В.А. БУНЬКО
А.М. ЭРПЕРТ
В.И. ЖУКОВИЦКИЙ

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА В ГОРНО-ТЕХНИЧЕСКИХ РАСЧЕТАХ

Допущено Министерством высшего и среднего специального образования УССР в качестве учебного пособия для студентов горных специальностей вузов

Киев — Донецк
Головное издательство
издательского объединения
«Вища школа»
1986

33.1я73
Б91

УДК 681.3.06 : 622

Вычислительная техника в горно-технических расчетах/В. А. Бунько, А. М. Эрперт, В. И. Жуковицкий.— К.; Донецк: Вища шк. Головное изд-во, 1986.— 160 с.

В учебном пособии рассмотрено применение электронной вычислительной техники в горной промышленности. Изложены основы программирования на алгоритмическом языке высокого уровня ФОРТРАН-IV в операционной системе ЕС ЭВМ. Содержатся сведения о программной реализации численных методов, используемых в горно-технических расчетах, а также модели, алгоритмы и программы для решения на ЭВМ типовых задач подземной и открытой разработки месторождений полезных ископаемых.

Для студентов горных вузов, изучающих курс «Программирование и расчеты на ЭВМ».

Табл. 14. Ил. 73. Библиогр.: 14 назв.

Подраздел 1 написал В. А. Бунько, подразделы 2, 3, 6, 7 — А. М. Эрперт, подразделы 4, 5 — В. И. Жуковицкий

Рецензенты: кафедра «Вычислительная техника в инженерных и экономических расчетах» Донецкого политехнического института (заведующий кафедрой кандидат технических наук, доцент *В. Я. Спорыхин*), заведующий кафедрой «Вычислительная техника» Криворожского горно-рудного института доктор технических наук, профессор *Э. Г. Файнштейн*

Редакция общетехнической литературы при Донецком государственном университете

Зав. редакцией *М. Х. Тахтаров*

Б $\frac{2501000000-307}{M211(04)-86}$ 213—86

ОГЛАВЛЕНИЕ

1. СОВРЕМЕННАЯ ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА В ГОРНОМ ДЕЛЕ

1.1. Техническая характеристика современных ЭВМ

Начало истории развития отечественной электронной вычислительной техники следует отнести к 1951 г., когда в нашей стране была создана первая ЭВМ — малая электронная счетная машина (МЭСМ). Ее структура и основные схемные решения, ставшие классическими, были положены в основу других машин, разработанных в последующие годы: БЭСМ-1, БЭСМ-2, «Стрела», М-20, «Минск-1», «Урал-1», «Урал-4». Эти машины теперь относят к ЭВМ первого поколения. Их элементной базой были электронные лампы.

Появление полупроводниковых приборов и быстрые темпы совершенствования технологии их изготовления способствовали созданию в 60-е гг. ЭВМ второго поколения. Применение полупроводниковых приборов позволило повысить надежность ЭВМ, снизить потребляемую ими мощность, уменьшить габариты и применить принципиально новые архитектурные решения. К ЭВМ второго поколения относятся БЭСМ-4, БЭСМ-6, М-220, М-222, «Урал-11», «Минск-22», «Минск-32» и др. Они обладают большим объемом оперативной памяти, хорошим быстродействием (от 5 тыс. до 1 млн. операций в секунду), однако имеют серьезные недостатки: программную несовместимость различных типов универсальных ЭВМ; большое количество разнотипных внешних устройств; необходимость для каждого типа машин собственного программного обеспечения и др.

Качественный и количественный скачок в развитии вычислительной техники произошел в конце 60-х гг. К этому времени был освоен выпуск интегральных микросхем, которые стали элементной базой ЭВМ третьего поколения, имеющих техническую и программную совместимость. Эти вычислительные машины получили название ЕС ЭВМ (единая система ЭВМ).

Основными особенностями ЕС ЭВМ (табл. 1.1) являются: использование интегральных схем в качестве элементной базы; информационная и программная совместимость различных моделей; наличие развитой системы программного обеспечения; возможность работы в мультипрограммном режиме; наличие большого набора внешних устройств, позволяющих использовать ЭВМ в различных областях; возможность расширения и усовершенствования моделей ЕС ЭВМ.

В начале 70-х гг. странами социалистического содружества был налажен выпуск ЕС ЭВМ первой очереди: ЕС-1020, ЕС-1030, ЕС-1040, ЕС-1050 и др. Затем на основе их модернизации были созданы модели с улучшенными параметрами: ЕС-1022, ЕС-1032, ЕС-1033. Во второй половине 70-х гг. начат серийный выпуск ЕС ЭВМ второй очереди. Эти машины (ЕС-1035, ЕС-1045, ЕС-1055, ЕС-1060, ЕС-1065) характеризуются повышенными технико-экономическими показателями.

Дальнейшее развитие вычислительной техники связано с созданием больших интегральных схем (БИС). Применение БИС позволяет

Таблица 1.1. Техническая характеристика ЕС ЭВМ

Модель	Быстродействие, тыс. операций в секунду	Максимальный объем оперативной памяти, К	Количество каналов		Модель	Быстродействие, тыс. операций в секунду	Максимальный объем оперативной памяти, К	Количество каналов	
			мультиплексных	селекторных				мультиплексных	селекторных
1020	10...20	256	1	2	1033	180	512	1	3
1030	60	512	1	3	1035	140	1024	1	4
1040	400	1024	1	6	1045	540	4096	2	5
1050	500	1024	1	6	1055	450	2048	2	4
1022	80...90	512	1	2	1060	1300	8192	2	6
1032	180	512	1	3	1065	4500	16 384	2	11

реализовать в одном корпусе микросхемы целые подсистемы ЭВМ (устройства, блоки), например запоминающие устройства, микропроцессоры и т. п. В связи с этим появилась возможность создания ЭВМ с новыми принципами организации, в том числе и мультипроцессорных. Введя в состав ЭВМ несколько процессоров, можно организовать процесс с распараллеливанием вычислений и получить такое быстродействие, которое практически недостижимо при однопроцессорной конфигурации ЭВМ. К примеру, производительность вычислительного комплекса «Эльбрус-2», выполненного на основе БИС, составляет более 100 млн. операций в секунду.

Одним из основных направлений научно-технического прогресса в области вычислительной техники является широкое применение микропроцессоров и создание на их базе встроенных систем автоматического управления и микро-ЭВМ. В нашей стране серийно выпускаются микро-ЭВМ «Электроника-60», «Электроника К1-10», СМ1800, VEFORMIKA, персональная ЭВМ СДГ-01 и др. Они снабжены дисплеями, устройствами ввода, вывода, печати и позволяют решать широкий класс научных, инженерных и экономических задач. Программное обеспечение микро-ЭВМ дает возможность применять язык АССЕМБЛЕР, а также алгоритмические языки высокого уровня — ПЛ/М, БЕЙСИК, ФОРТРАН, ПАСКАЛЬ. При малых габаритах (для размещения достаточно письменного стола) по производительности и математическим возможностям микро-ЭВМ сравнимы с универсальными ЭВМ второго поколения (например, «Минск-22»), а в некоторых случаях даже выше.

Коммунистическая партия и Советское государство постоянно уделяют внимание развитию средств вычислительной техники и их широкому использованию в управлении производством, технологическими процессами, научных исследованиях и других сферах деятельности. Обширная программа дальнейшего развития вычислительной техники и ее применения в народном хозяйстве намечена в Основных направлениях экономического и социального развития СССР на 1986—1990 годы и на период до 2000 года. Запланировано обеспечить рост объема производства вычислительной техники в 2—2,3 раза, высокими темпами наращивать масштабы применения современных высокопроизводительных ЭВМ всех классов, продолжить создание и повысить эффектив-

ность работы вычислительных центров коллективного пользования, интегрированных банков данных, сетей обработки и передачи информации.

Большой вклад в развитие вычислительной техники и создание теоретических основ ее построения и применения внесли видные советские ученые С. А. Лебедев, А. И. Берг, В. М. Глушков, Ю. А. Базилевский, А. П. Ершов, А. И. Китов, Б. И. Рамеев, А. М. Ларионов, С. А. Майоров и др. Благодаря усилиям ученых и инженеров вычислительная техника в нашей стране является одной из наиболее динамично развивающихся отраслей современной техники.

1.2. Основные устройства ЕС ЭВМ

Логическая структура ЕС ЭВМ показана на рис. 1.1. Основным устройством ЭВМ является процессор, содержащий вычислительное устройство (*ВУ*), оперативную память (*ОП*), мультиплексный (*МК*) и селекторные (*СК*) каналы, пульт управления. В состав *ВУ* входит арифметико-логическое устройство (*АЛУ*), которое выполняет операции арифметической и логической обработки данных. Для управления этими операциями, а также обмена данными между *АЛУ*, *ОП* и внешними устройствами (*ВнУ*) предусмотрено устройство управления (*УУ*). В основной памяти хранятся числовые данные, программа решаемой задачи и специальная программа, которая управляет процессом выполнения программы пользователя.

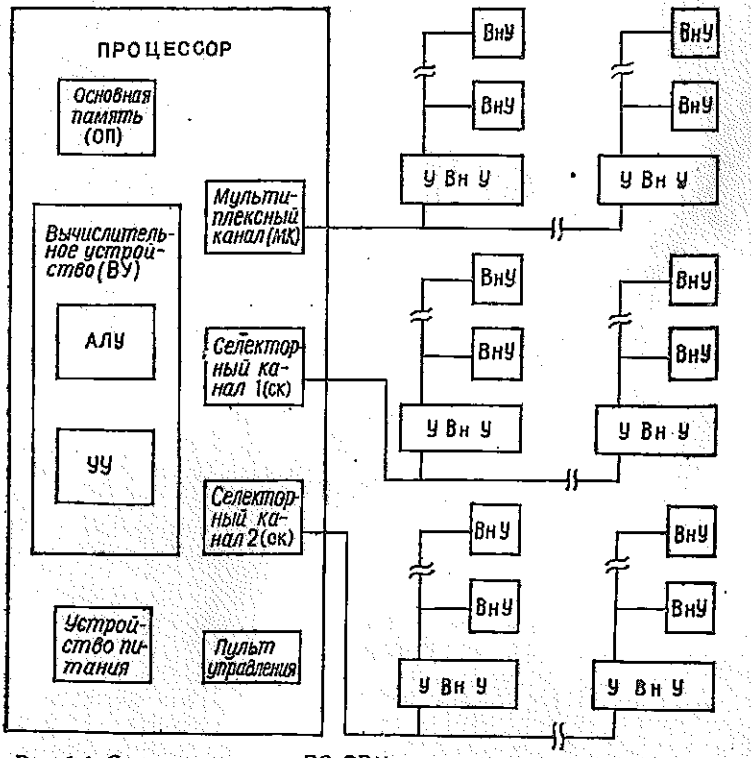


Рис. 1.1. Структурная схема ЕС ЭВМ

Пульт управления предназначен для индикации состояния процессора, приведения ЭВМ в исходное состояние перед решением задачи, проверки при ремонте ЭВМ.

К внешним устройствам относятся медленнодействующие (пультовая пишущая машинка, устройства ввода с перфокарт, перфолент и вывода на перфокарты и перфоленты, печатающие устройства) и быстродействующие (накопители на магнитных лентах, на магнитных дисках).

Пультовая пишущая машинка обеспечивает ввод в ЭВМ директив, инициирующих выполнение задания, приостановку решения задачи, снятие задания и другие операции. В процессе работы ЭВМ печатает на пишущей машинке служебную информацию, содержащую сведения о ходе выполнения задания.

Устройства ввода с перфокарт и перфолент служат для считывания информации, закодированной в виде пробивок, преобразования ее в электрические сигналы и передачи последних в ЭВМ. С помощью этих устройств в ЭВМ вводятся программы и числовой материал решаемых задач, а также специальная информация, управляющая процессом обработки задач.

Устройства вывода на перфокарты и перфоленты преобразуют передаваемую из ЭВМ информацию в виде электрических сигналов в информацию в виде пробивок на перфокартах и перфолентах.

Печатающие устройства предназначены для преобразования информации, выводимой из ЭВМ в виде электрических сигналов, в систему печатных знаков на бумаге. Устройство позволяет печатать на бумаге цифры, буквы русского и латинского алфавитов, специальные символы.

Накопители на магнитных лентах и магнитных дисках применяют для записи, хранения и выдачи больших массивов информации. Они используются в качестве внешней памяти ЭВМ.

Устройства управления внешними устройствами *(УВнУ)* подключаются к процессору через мультиплексный и селекторные каналы ввода-вывода. Мультиплексный канал обеспечивает обмен информацией между процессором и медленнодействующими внешними устройствами. Он работает в мультиплексном или монопольном режиме. В мультиплексном режиме *МК* может осуществлять обмен информацией между процессором и несколькими внешними устройствами (до 112). В монопольном режиме *МК* соединяет с процессором только одно внешнее устройство. Селекторный канал в каждый момент времени осуществляет обмен информацией между процессором и одним внешним устройством. Первый *СК* повышенной пропускной способности подключен к внешней памяти на магнитных дисках, а второй — к внешней памяти на магнитных лентах. К одному *СК* можно подсоединить до 8 *УВнУ*.

1.3. Применение ЭВМ для управления на горных предприятиях

В горной промышленности ЭВМ широко используются в системах управления технологическими процессами, производственной и экономической деятельностью предприятий и объединений.

Электронно-вычислительные машины являются технической базой отраслевых автоматизированных систем управления (ОАСУ-уголь, ОАСУ-руда).

На горных предприятиях с высоким уровнем механизации и автоматизации производственных процессов ЭВМ применяются для оперативно-диспетчерского управления. Они осуществляют сбор и обработку информации о работе отдельных машин, комплексов и участков шахты (карьера), управление технологическими процессами с целью стабилизации или оптимизации режимов работы, регистрацию и классификацию статистической информации о работе и простоях технологического оборудования, цехов, участков предприятия и др. Примерами таких систем могут служить разработанные в Гипроуглеавтоматизации системы оперативно-диспетчерского управления шахтами «Октябрьская» и «Лутугинская-Северная» в Донбассе, «Прогресс» и «Северная» в Подмосковном угольном бассейне, разрезом «Томусинский 3/4» в Кузбассе. Много лет функционирует система оперативного управления транспортом на карьере Ингулецкого горно-обогатительного комбината, которая позволяет осуществлять поставку руды на обогатительную фабрику в режиме стабилизации качества полезного ископаемого.

Большинство производственных объединений угольной промышленности и крупных горно-обогатительных комбинатов оснащено автоматизированными системами управления производством (АСУП). Основными функциями АСУП являются: повышение оперативности информации, улучшение ее использования; получение объективных данных от предприятий и цехов для анализа и оценки выполнения плана; управление производственно-хозяйственной деятельностью предприятий; планирование технико-экономических показателей работы предприятий с применением методов оптимизации и др.

Применение ЭВМ на всех уровнях управления позволяет повысить оперативность и достоверность информации о работе горных предприятий и отдельных звеньев, использовать экономико-математические методы для оптимизации их деятельности и управления технологическими процессами, что в конечном итоге обеспечивает существенное улучшение технико-экономических показателей.

1.4. Использование ЭВМ в горно-технических расчетах

Горно-технические расчеты и научные исследования в области горной техники и технологии связаны с решением задач, характеризующихся большим объемом вычислений, что объясняет эффективность и целесообразность использования ЭВМ. Задачи, решаемые с применением ЭВМ, можно условно разделить на четыре типа: инженерно-технические расчеты; технико-экономические расчеты; моделирование процессов горной технологии и организации производства; автоматизированное проектирование шахт, карьеров и обогатительных фабрик.

Наиболее представительными являются задачи первого типа. Так, расчеты на ЭВМ выполняют при обработке статистических данных

о качественно-технологических показателях полезного ископаемого по результатам геологического и оперативного опробования, при выполнении горно-геометрического анализа шахтных и карьерных полей, при исследовании и расчете надежности отдельных звеньев технологического оборудования и транспортно-технологических схем, расчете паспортов вскрышных работ при открытой разработке горизонтальных месторождений, параметров и показателей усреднения аккумулирующих емкостей и др.

К технико-экономическим расчетам относятся задачи оптимального размещения добычи угля в бассейнах и планирования его поставок, определения оптимальных направлений и способов концентрации производства на шахтах, распределения объема добычи горного предприятия по отдельным участкам при соблюдении ограничений по качеству полезного ископаемого и пропускной способности технологических звеньев. Большинство задач этого типа по математической формулировке являются задачами математического программирования. Например, задача определения оптимального состава рудной шихты, поступающей на обогатительную фабрику, относится к задачам линейного программирования. Для условий современных горно-обогатительных комбинатов такая задача должна содержать 30—70 ограничений при 40—150 неизвестных. Очевидно, что ее решение в приемлемые сроки возможно только с помощью ЭВМ.

Математическое моделирование процессов горной технологии и организации производства занимает видное место в научных и прикладных исследованиях. В основе методов математического моделирования лежит идея аналитического синтеза элементов производственного процесса и математического эксперимента. Математической моделью объекта является его описание в виде математических зависимостей и логических условий, которыми выражается взаимосвязь между входными и выходными параметрами объекта и его собственными характеристиками. Моделирование применяют для выбора из множества альтернативных решений оптимального.

В области организации и управления горными работами наибольшее распространение получили математические модели, описательным аппаратом которых является математическое программирование. При исследовании технологических процессов применяют имитационное моделирование, которое позволяет по известным законам распределения поведения элементов, составляющих систему, воссоздать процесс функционирования системы в целом. В настоящее время для анализа и исследования объектов горной технологии используются следующие модели: оптимального управления очистным забоем; функционирования шахтного транспорта; функционирования автомобильного транспорта на карьере при циклично-поточной технологии горных работ и др.

Опыт работы научно-исследовательских и проектных институтов горного профиля показал, что ЭВМ можно эффективно использовать при проектировании горных предприятий, поскольку этот процесс требует сложных и трудоемких расчетов, анализа и сопоставления вариантов. Первоначально ЭВМ применялись для решения частных задач проектирования: горно-геометрического анализа месторождения,

расчетов вентиляционной сети, проветривания, определения параметров вскрытия и подготовки шахтного (карьерного) поля, выбора технологической схемы транспорта и др. В последние годы во многих проектных организациях и научно-исследовательских институтах ведутся работы по созданию комплексных систем автоматизированного проектирования (САПР), позволяющих использовать ЭВМ на всех стадиях проектирования, начиная от технико-экономического обоснования и кончая выпуском проектной документации.

2. АЛГОРИТМИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

2.1. Алгоритм и его свойства

Процесс подготовки и решения задачи на ЭВМ включает следующие этапы.

1. *Постановка задачи* — четкая и однозначная формулировка задачи с составлением перечня исходных данных, достаточных для ее решения, а также выходных данных с указанием формы их представления.

2. *Разработка алгоритма решения* — выбор и обоснование метода решения задачи и составление детальной схемы алгоритма, определяющей последовательность вычислений.

3. *Составление программы решения задачи* — представление алгоритма в форме, удобной для передачи его содержания ЭВМ. На этом этапе схема алгоритма записывается в виде набора машинных кодов или одним из алгоритмических языков. Для ввода программы в ЭВМ полученный текст наносится на соответствующие носители информации (перфокарты, перфолента, магнитная лента и т. п.).

4. *Отладка программы* — выявление ошибок, допущенных на предыдущих этапах, и их устранение. На данном этапе получают контрольный просчет, т. е. решение одного из вариантов задачи (возможно упрощенного), и проводят детальный анализ результатов.

5. *Решение задачи* — получение на ЭВМ требуемых результатов по отлаженной программе.

Первый этап задачи выполняет специалист в соответствующей области науки или техники, второй, третий и четвертый — программист (зачастую постановщиком задачи и программистом является одно и то же лицо), а пятый — операторы, обслуживающие вычислительную машину.

Разработка алгоритма — один из наиболее ответственных этапов. От его квалифицированного исполнения зависит качество программы и эффективность использования времени вычислительной машины.

Под алгоритмом понимают систему однозначно понимаемых предписаний, при выполнении которых получают решение поставленной задачи. Для пояснения приведенного определения рассмотрим пример алгоритма нахождения наименьшего числа из множества n чисел $\{x_1, x_2, \ldots, x_n\}$. Алгоритм опишем в виде последовательности шагов (предписаний), каждый из которых содержит словесное описание осуществляемых на данном шаге действий. Обозначим min — значе-

ние искомого наименьшего числа, l — текущий номер элемента множества X. Тогда алгоритм запишется следующим образом:

Шаг

1 Принять $\min = x_1$
2 Принять $l = 2$
3 Если $\min > x_l$, то $\min = x_l$
4 Увеличить l на единицу
5 Если $l \leqslant n$, перейти к шагу 3
6 Конец

Пусть $n = 3$, $X = \{10, 5, 7\}$. В данном случае последовательность действий по приведенному алгоритму такая:

Шаг

1 $\min = 10$
2 $l = 2$
3 $\min > 5$, принимаем $\min = 5$
4 $l = 3$
5 $l = n$, переходим к шагу 3
3 $\min < 7$ ($\min$ не изменяется)
4 $l = 4$
5 $l > n$ (к шагу 3 не переходим)
6 Конец ($\min = 5$)

Любой алгоритм должен обладать свойствами:

детерминированности (однозначности) — применение алгоритма к одним и тем же исходным данным должно всегда приводить к одному и тому же результату;

массовости — возможность выбора исходных данных из некоторого множества значений, т. е. алгоритм должен обеспечивать решение любой задачи из некоторого класса однотипных задач;

результативности — получение искомого результата после конечного числа шагов.

Поясним свойства алгоритма на примерах. Свойство детерминированности проявляется в том, что многократное повторение описанного выше алгоритма нахождения минимального числа из множества применительно к множеству $X = \{10, 5, 7\}$ всегда даст один и тот же результат: $\min = 5$.

Для объяснения свойства массовости рассмотрим определенный интеграл:

$$F = \int_a^b x^3 dx. \qquad (2.1)$$

Алгоритм решения этой задачи можно представить формулой

$$F = (b^4 - a^4)/4. \qquad (2.2)$$

Следовательно, формула (2.2) приемлема для вычисления определенного интеграла (2.1) при любых значениях a и b ($b \geqslant a$).

Чтобы проиллюстрировать свойство результативности алгоритма, рассмотрим пример вычисления суммы членов бесконечного сходя-

щегося ряда:

$$S = \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^k}{k!} + \cdots = \sum_{k=1}^{\infty} \frac{x^k}{k!}. \quad (2.3)$$

Эта формула не удовлетворяет свойству результативности, поскольку для вычисления суммы S требует бесконечное число шагов. Результативность алгоритма, описываемого формулой (2.3), можно обеспечить, если добавить дополнительное условие: накопление суммы членов ряда следует прекратить, когда значение очередного слагаемого $x^k / k!$ станет меньше наперед заданной достаточно малой величины.

Для описания алгоритма применяют различные способы. Наиболее распространенными из них являются описания: словесное в виде последовательности предписаний, формульное и с помощью схем.

2.2. Запись алгоритма

Для передачи содержания алгоритма вычислительной машине его записывают в виде последовательности команд или на специальных алгоритмических языках (ФОРТРАН, АЛГОЛ, КОБОЛ и др.).

Язык команд. Команда — это инструкция, которая представляется цифровым кодом (набором цифр) и содержит все сведения, необходимые для выполнения одного арифметического или логического действия. Последовательность команд, описывающая некоторый вычислительный процесс, называется *программой.*

Программа и числовые данные, записанные цифровым кодом, хранятся в памяти ЭВМ. Память состоит из пронумерованных ячеек, в каждой из которых помещается одна команда или одно число. Номер ячейки с хранящимся цифровым кодом называют *адресом* этого кода.

Команды в программе располагаются в той последовательности, в какой они должны выполняться. Их последовательность может изменяться с помощью специальных команд безусловной или условной передачи управления. Расшифровка и исполнение ЭВМ последовательности команд программы называется *программным управлением вычислительным процессом.*

Команда ЭВМ содержит код операции и адресную часть. Первый из них указывает, какую операцию следует выполнить над операндами (числами, участвующими в операции). В адресную часть входят адреса операндов и результатов операций в памяти ЭВМ. В зависимости от структуры адресной части различают одно-, двух- и трехадресные команды. Каждая ЭВМ оперирует с командами определенной адресности. Например, ЭВМ серии «Урал» являются одноадресными, «Минск» и ЕС — двухадресными, а типа М-20 (М-220, БЭСМ-4 и др.) — трехадресными.

Структура трехадресной команды показана на рис. 2.1. Адреса a_1 и a_2 указывают, из каких ячеек памяти берутся операнды, а адрес a_3 — в какую ячейку будет помещен результат операции. В ЭВМ второго поколения код операции кодируется двухразрядным числом со знаком «плюс» или «минус», адреса — четырехразрядными числами.

Код операции (КОП)	Адрес 1 (a_1)	Адрес 2 (a_2)	Адрес 3 (a_3)

Рис. 2.1. Структура трехадресной команды

Таким образом, команда по способу изображения представляет собой многоразрядный цифровой код со знаком.

Пусть кодам операций соответствуют такие значения: +10 — сложение; +20 — вычитание; +30 — умножение; +40 — деление; —00 — останов. Предположим, что число 5,7 записано в ячейке № 1000, а число 12,3 — в ячейке № 2500.

Код команды + 10 1000 2500 1900 (КОП = +10, $a_1 = 1000$, $a_2 = 2500$, $a_3 = 1900$) ЭВМ расшифрует следующим образом: из ячеек памяти № 1000 и 2500 извлечь операнды (числа 5,7 и 12,3 соответственно), сложить их и результат операции поместить в ячейку памяти № 1900. Следовательно, после выполнения команды в ячейку памяти № 1900 запишется число 18,0.

Рассмотрим методику составления программы для ЭВМ с указанными кодами операций. Допустим, необходимо выполнить вычисления по формуле $R = (4{,}7 \cdot 8{,}2^2 - 25{,}3) : (17{,}1 + 9{,}7)$. Процесс составления программы в данном случае делится на два этапа.

Первый этап. Для исходных, промежуточных чисел и окончательного результата выделяются ячейки в памяти ЭВМ. Исходные данные должны быть помещены в соответствующие ячейки перед началом решения задачи. Окончательные результаты (в нашем примере R) извлекаются из памяти после окончания вычислений. Следует иметь в виду, что при записи числа в ячейку памяти ее предыдущее содержимое стирается, поэтому одну и ту же ячейку можно использовать для хранения нескольких последовательно получаемых промежуточных результатов.

Для нашего примера размещение числовой информации в памяти ЭВМ может быть выполнено следующим образом:

Число	4,7	8,2	25,3	17,1	9,7	R	Промежуточные результаты		
Адрес	1000	1001	1002	1003	1004	1005	1006	1007	...

Второй этап. Каждое из действий, которые необходимо выполнить для решения задачи, кодируется в виде отдельной команды. Последовательность расположения команд определяется приоритетом выполняемых действий. Программа для рассматриваемого примера приведена в табл. 2.1.

Программа состоит из шести команд. Номер команды обозначает номер ячейки памяти, в которой хранится цифровой код данной команды. Отметим, что в ячейки № 1006 и 1007 дважды производилась запись промежуточного результата. В конечном итоге в ячейку № 1006 помещен числитель, а в ячейку № 1007 — знаменатель вычисляемого выражения. Команда останова не имеет операндов и результата вычислений, поэтому ее адресная часть заполняется нулями.

Алгоритмические языки. На ранних этапах развития вычислительной техники программирование велось на машинных языках. Их применение делает процесс программирования очень трудоемким. При

Таблица 2.1. Программа на языке команд

Номер команды	КОП	a_1	a_2	a_3	Пояснения
0001	+30	1001	1001	1006	8,2×8,2, запись в ячейку 1006
0002	+30	1000	1006	1007	$4{,}7\times8{,}2^2$, запись в ячейку 1007
0003	+20	1007	1002	1006	$4{,}7\times8{,}2^2-25{,}3$, запись в ячейку 1006
0004	+10	1003	1004	1007	17,1 + 9,7, запись в ячейку 1007
0005	+40	1006	1007	1005	Вычисление R, запись в ячейку 1005
0006	−00	0000	0000	0000	Останов

составлении программы необходимо запоминать, в каких ячейках памяти ЭВМ хранятся исходные данные и промежуточные результаты. Программа обычно имеет громоздкую запись, что затрудняет проверку и поиск ошибок. Кроме того, программа, составленная для машины определенного типа, не пригодна для использования на машинах других типов.

Трудности программирования на машинных языках вызвали необходимость создания алгоритмических языков, которые лишены основных недостатков машинных языков и позволяют записывать алгоритм решения задачи в компактной форме.

Алгоритмические языки подразделяются на машинно-ориентированные (автокоды) и машинно-независимые. Автокоды дают возможность составлять программы в символической записи. Сущность программирования на автокоде состоит в том, что в командах вместо конкретных кодов операций и адресов записываются буквенно-цифровые символы, наименования переменных, констант, обозначения типов операций и др. Программу, записанную на автокоде, называют *символической.* Непосредственно символическую программу ЭВМ выполнять не может, ее необходимо перевести на машинный язык, для чего символические коды операций и адреса следует преобразовать в числовые коды и числовые адреса. Этот перевод осуществляет специальная программа-переводчик, которая именуется «транслятор».

При программировании на автокоде программист избавлен от необходимости составлять таблицу распределения числовой информации в памяти ЭВМ; эту функцию берет на себя транслятор. Кроме того, программа в символической записи нагляднее программы на машинном языке, ее проще читать, проверять и отлаживать. Недостатком символической программы является то, что она составляется для машин одного типа, из-за чего автокоды и относятся к машинно-ориентированным языкам.

Машинно-независимые языки лишены указанного недостатка и позволяют использовать одну и ту же программу для решения задач на машинах разных типов. В основу машинно-независимых языков положен формульно-словесный способ записи алгоритмов: часть указаний задается в виде формул, а часть, определяющая тип величин и организацию вычислительного процесса,— словами. Запись на таких языках близка к разговорному, что существенно упрощает процесс написания и проверки программы. Очевидно, что непосредственно по программе на машинно-независимом языке ЭВМ решать задачу

не может: как и для автокодов в данном случае также необходим перевод на машинный язык соответствующей ЭВМ, который выполняет транслятор. Следовательно, уменьшение затрат времени и усилий на составление программы, связанное с применением алгоритмических языков, приводит к увеличению затрат машинного времени в процессе ее использования.

В настоящее время создано и используется большое количество специализированных и универсальных алгоритмических языков. Среди них наиболее широко распространены ФОРТРАН, АЛГОЛ-60 (для решения математических задач), КОБОЛ (программирование экономических задач) и PL/1 — универсальный алгоритмический язык.

2.3. Символы схем алгоритмов

Перед написанием программы на автокоде или на алгоритмическом языке структура алгоритма должна быть представлена графически, т. е. в виде структурной схемы. Для этого используется набор блочных символов, показанных на рис. 2.2.

Символ «*Обработка*» обозначает операции, результатом выполнения которых является вычисление значения переменной; соответствующие вычисления записываются внутри прямоугольника.

Символ «*Проверка*» позволяет изображать разветвление вычислительного процесса в зависимости от значения некоторого условия: если записанное внутри символа условие выполняется, осуществляется переход на следующий блок по соединительной линии, над которой написано «Да»; а если не выполняется, то переход происходит по соединительной линии, над которой написано «Нет».

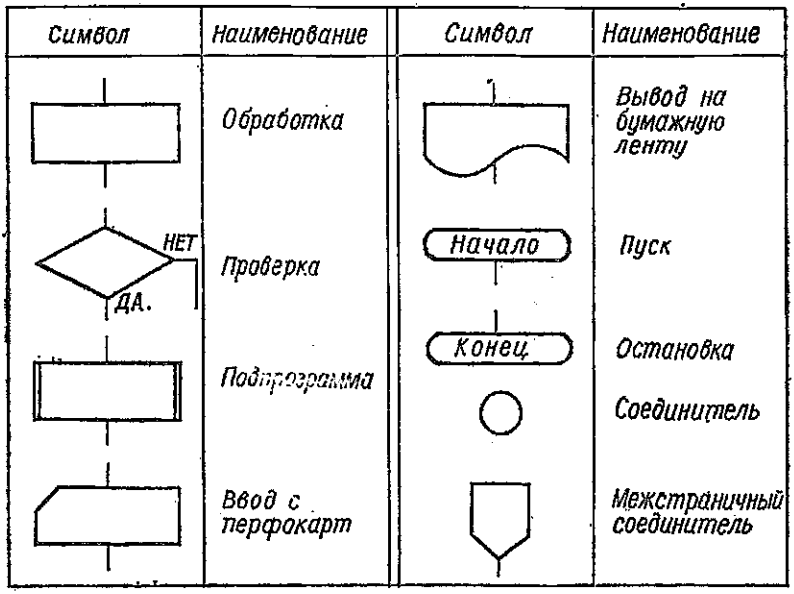


Рис. 2.2. Блочные символы схем алгоритмов

Символ *«Подпрограмма»* обозначает алгоритм, оформляемый в виде отдельной единицы — подпрограммы, к которой производится обращение из разрабатываемой программы.

Символ *«Ввод с перфокарт»* предназначен для ввода в память ЭВМ исходных данных, с которыми будут выполняться вычисления при решении задачи.

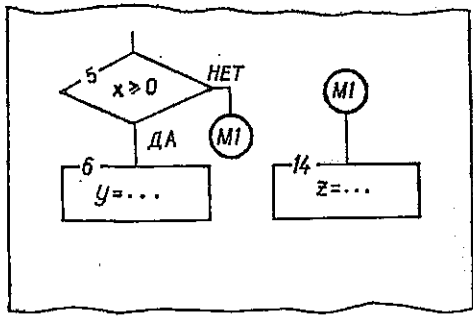


Рис. 2.3. Пример применения символа «Соединитель»

Символ *«Вывод на бумажную ленту»* обозначает печатание на бумажной ленте результатов вычислений.

Символом *«Пуск-останов»* изображают начало, конец или прерывание процесса решения задачи.

Символ *«Соединитель»* применяют для обозначения связи прерванных линий, которые соединяют блоки, помещенные на одном листе. Концы прерываемых линий помечают одинаковыми метками (цифрами, буквами). Пример применения символа «Соединитель» показан на рис. 2.3: если $x < 0$, из блока *5* происходит переход по прерванной соединительной линии на блок *14*.

Символ *«Межстраничный соединитель»* применяют для той же цели, что и символ «Соединитель», но при соединении блоков, расположенных на разных листах схемы алгоритма. Внутри символа указывается номер листа и блока, с которого (на который) поступает соединительная линия (рис. 2.4).

При направлении линии сверху вниз или слева направо стрелка не ставится; в остальных случаях ее направление должно быть отмечено стрелкой.

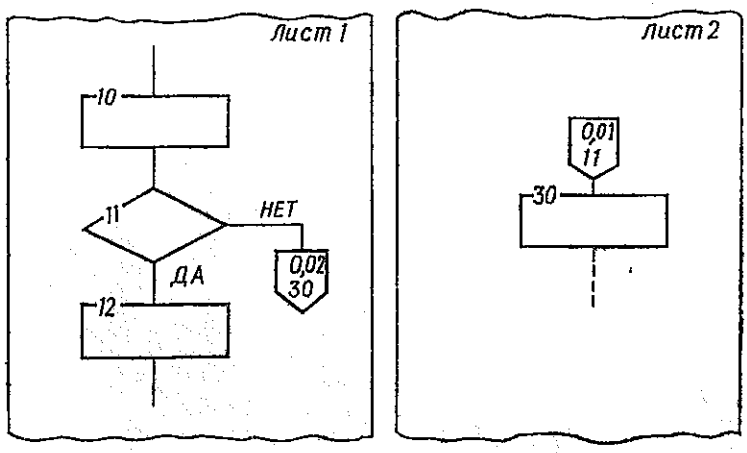


Рис. 2.4. Пример применения символа «Межстраничный соединитель»

Различают укрупненные и рабочие схемы алгоритмов. В *укрупненных* отражается структура вычислительного процесса без пооперационной детализации. Такая схема состоит из относительно небольшого числа блоков, каждый из которых предусматривает определенный объем вычислений, связанных с одним из этапов решения задачи. Каждый блок укрупненной схемы, в свою очередь, может быть изображен укрупненной схемой алгоритма, в которой последовательность действий описана с большей степенью детализации. Количество уровней представления указанной схемы зависит от сложности задачи и, как правило, не превышает двух-трех. В *рабочей* схеме алгоритма отражается детализация вычислительного процесса вплоть до отдельных операций. По ней составляется программа.

2.4. Составление схем алгоритмов

Вычислительные процессы могут быть последовательного, разветвляющегося и циклического типов. В алгоритмах любой сложности можно выделить фрагменты перечисленных типов, которые могут размещаться последовательно или быть вложенными друг в друга.

Последовательным называется вычислительный процесс, в котором действия выполняются в порядке записи. Примером такого процесса может служить алгоритм решения совместной и определенной системы двух линейных алгебраических уравнений относительно двух неизвестных.

Пусть дана система уравнений

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = b_1; \\ a_{21}x_1 + a_{22}x_2 = b_2. \end{cases}$$

Определитель системы

$$D = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12}.$$

Для совместной и определенной системы $D \neq 0$, при неизвестных

$$D_1 = \begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix} = b_1a_{22} - b_2a_{12},$$

$$D_2 = \begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix} = a_{11}b_2 - a_{21}b_1.$$

Рассматриваемая система уравнений решается по формулам $x_1 = D_1/D$; $x_2 = D_2/D$.

Схема алгоритма показана на рис. 2.5.

Разветвляющимся называется вычислительный процесс, последовательность действий в котором зависит от значения условия. Примером в данном случае может служить вычисление выражения (рис. 2.6)

$$y = \begin{cases} \sqrt{x|a-b|}, & x \geqslant 0, \\ \sqrt{-x|a-b|}, & x < 0. \end{cases}$$

Разветвление вычислений выполняется с помощью блочного символа «Проверка». Количество разветвлений зависит от количества проверяемых условий.

Предположим, из трех чисел a, b и c надо найти наибольшее и вычислить его квадрат. Обозначим $x = \max\{a, b, c\}$, $y = x^2$. Схема алгоритма решения этой задачи показана на рис. 2.7. Вычислительный процесс имеет два блочных символа «Проверка», с помощью которых реализуются три разветвления. При удовлетворении обоих условий $x = a$. Если второе условие (блок *5*) выполняется, а первое (блок *3*) нет, то $x = b$, если же не выполняется второе условие, то $x = c$. Перед блоками проверки x принимает значение a, поэтому первая проверка сравнивает числа a и b, а вторая — большее из них с числом c.

Циклический — это такой вычислительный процесс, при котором многократно повторяются вычисления по одним и тем же математическим зависимостям при различных значениях входящих в них величин. Фрагмент схемы алгоритма, реализующий данный процесс, называется ц и к л о м, блочные символы схемы алгоритма, содержащие повторяющиеся вычисления,— о б л а с т ь ю ц и к л а, а аргумент выражения, изменяющийся при каждом повторении вычислений,— п а р а м е т р о м ц и к л а.

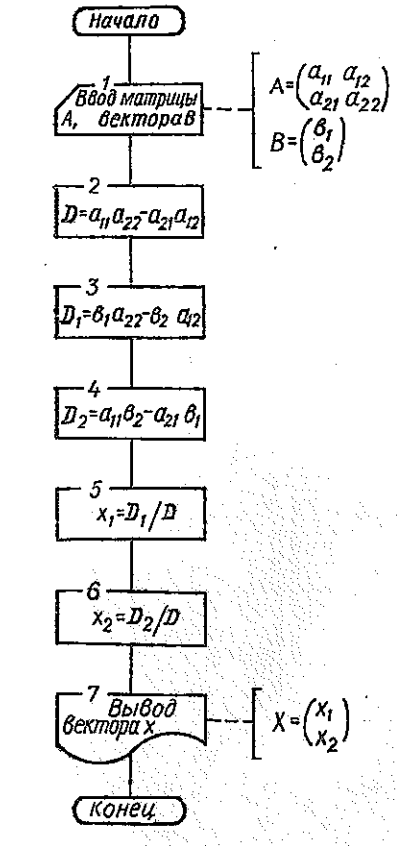
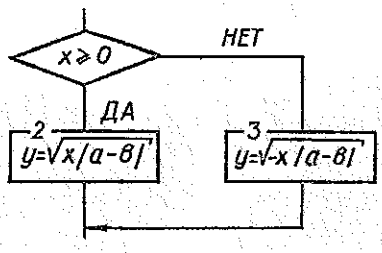


Рис. 2.5. Схема алгоритма последовательного вычислительного процесса

Для пояснения приведенных определений рассмотрим пример. Пусть надо вычислить и отпечатать значения функции $F = 4{,}5x^2 + 2{,}1x + 0{,}7$ при $x = 0{,}1;\ 0{,}2;\ \ldots;\ 1{,}5$. Очевидно, что для решения этой задачи необходимо повторить вычислительный процесс пятнадцать раз, изменяя аргумент x от 0,1 до 1,5 с шагом 0,1. Чтобы организовать циклический процесс, зададим аргументу x начальное значение 0,1, а затем будем увеличивать его с заданным шагом. Каждое текущее значение сравнивается с конечным. Если $x \leqslant 1{,}5$, вычисления повторяются, ес-

Рис. 2.6. Фрагмент схемы алгоритма разветвляющегося вычислительного процесса



ли же аргумент стал больше конечного значения, вычисления следует прекратить.

Схема алгоритма описанного циклического вычислительного процесса изображена на рис. 2.8. Параметром цикла в данном примере является аргумент x. Блок *1* формирует начальное значение аргумента, блок *4* увеличивает его на величину шага, а блок *5* сравнивает с конечным значением. Область цикла образуют блоки *2*, *3*. Блок *4* содержит запись, которая с математической точки зрения кажется некорректной. В действительности знак «=» интерпретируется не как математическое равенство, а как действие присваивания. Под присваиванием подразумевают вычисление выражения, находящегося справа от знака «=», и присваивание результата переменной, стоящей слева. Так, в записи $x = x + 0{,}1$ в блоке *4* переменная x слева и справа от знака равенства имеет разные значения, отличающиеся друг от друга на 0,1. Следовательно, запись корректна.

В зависимости от способа выхода из цикла различают вычислительные процессы типа арифметической прогрессии и итеративного типа. Первый из них характеризуется тем, что количество значений параметра цикла и, следовательно, количество повторений вычислений в области цикла известно. К такому типу относится пример вычисления значений функции F,

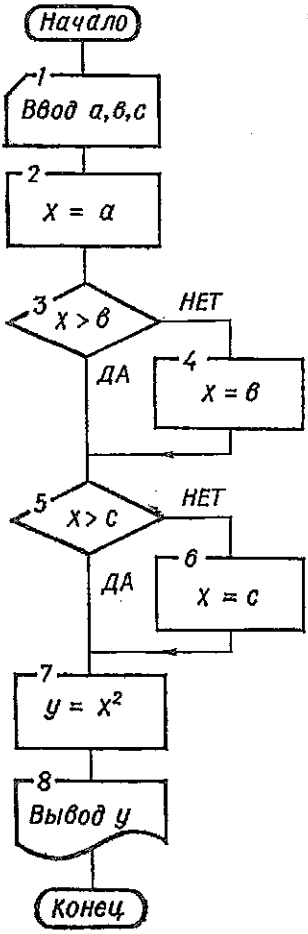


Рис. 2.7. Схема алгоритма вычисления квадрата наибольшего из трех чисел

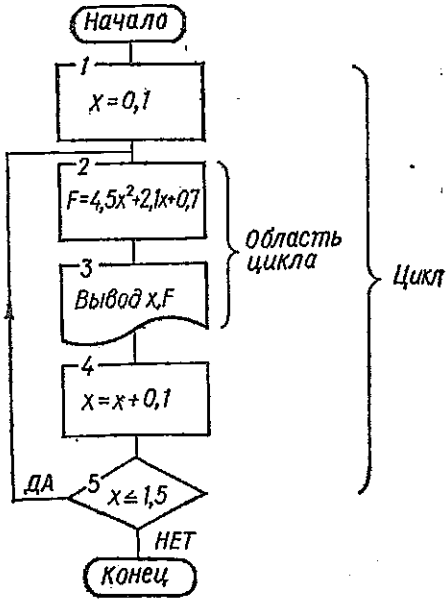


Рис. 2.8. Схема алгоритма циклического вычислительного процесса

поскольку количество повторений вычислений (в этом примере 15) и значение параметра цикла x известны из постановки задачи.

В циклах типа арифметической прогрессии параметром цикла может быть индекс, т. е. целочисленная величина, идентифицирующая номер элемента некоторого массива.

Пример. Предположим, на участке рудного месторождения взято пятьдесят проб для определения содержания полезного компонента в руде. Результаты опробования представим массивом A_1, A_2, A_3, ..., A_{50}. Необходимо установить среднее значение $A_{ср}$ и дисперсию D содержания полезного компонента. Искомые величины вычисляем по формулам:

$$A_{ср} = \sum_{i=1}^{50} A_i/50; \quad D = \sum_{i=1}^{50} A_i^2/50 - A_{ср}^2.$$

Чтобы составить схему алгоритма, обозначим: $S_1 = \sum_{i=1}^{50} A_i$, $S_2 = \sum_{i=1}^{50} A_i^2$.

Сумму вычисляем в такой последовательности: сначала принимаем ее равной нулю, затем прибавляем к ней i-е слагаемое ($i = 1, 2, \ldots, 50$) и результат опять присваиваем сумме. Очевидно, что прибавление i-го слагаемого следует производить пятьдесят раз. При каждом повторении этого действия индекс слагаемого i увеличивается на единицу. Для значения S_1 последовательность действий можно представить следующим образом:

$$S_1 = 0;$$
$$S_1 = (0) + A_1, \quad i = 1;$$
$$S_1 = (A_1) + A_2, \quad i = 2;$$
$$S_1 = (A_1 + A_2) + A_3, \quad i = 3;$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$S_1 = (A_1 + \cdots + A_{49}) + A_{50}, \quad i = 50.$$

Из примера видно, что в качестве первого слагаемого (взятого в скобки) можно использовать предыдущее значение S_1. Параметром цикла служит индекс i. Поскольку суммы S_1 и S_2 определяются по одному и тому же индексу, их вычисление можно совместить в одной области цикла.

Схема алгоритма, реализующего вычисление значений $A_{ср}$ и D, показана на рис. 2.9. В данной схеме цикл образуют блочные символы *4—8*, область цикла состоит из блоков *5, 6*. Для описания организации цикла потребовалось три блочных символа: блок *4* — задание начального значения параметра цикла; блок *7* — организация приращения параметра цикла; блок *8* — проверка цикла на завершение. Все циклы типа арифметической прогрессии имеют такую организацию.

Чтобы уменьшить размеры схемы алгоритма (что особенно важно для вложенных циклов), модифицируем графическое изображение циклического вычислительного процесса таким образом. В начале циклического фрагмента схемы поместим перечень всех значений параметра цикла, а под ним — область цикла. Штриховой линией соединим список значений параметра цикла с последним блочным символом области цикла, указывая тем самым область, на которую распространяется действие данного параметра. Схема алгоритма вычисления среднего значения и дисперсии с модифицированным графическим изображением цикла показана на рис. 2.10.

Помимо введения вспомогательного блочного символа, используемого в качестве заголовка цикла (блок *3*), в схеме изменен способ нумерации блоков. Все блоки области цикла имеют двузначную нумерацию: первая цифра определяет подчиненность блока соответствую-

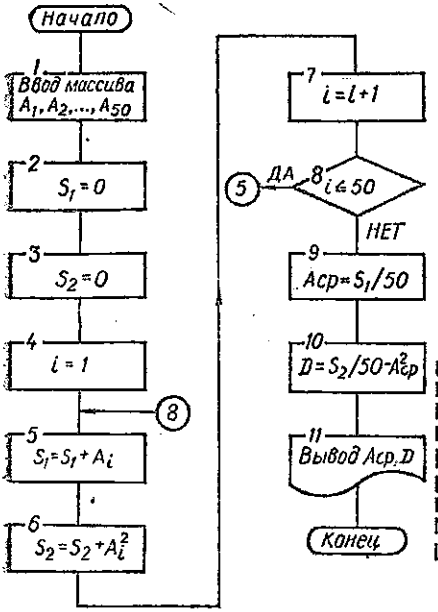


Рис. 2.9. Схема алгоритма циклического вычислительного процесса типа арифметической прогрессии

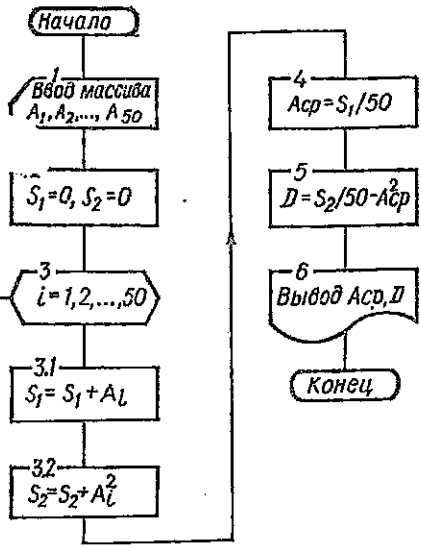


Рис. 2.10. Схема алгоритма циклического вычислительного процесса типа арифметической прогрессии с модифицированным графическим изображением цикла

щему заголовку цикла, а вторая — порядковый номер блока в области цикла. Читается циклический фрагмент данной схемы так: блоки области цикла (блоки *3.1* и *3.2*) следует выполнить многократно при всех значениях параметра i, указанных в блоке заголовка цикла (блок *3*). Приведенная нумерация не является обязательной; можно использовать и последовательную нумерацию всех блоков.

В рассмотренных примерах циклических вычислительных процессов область цикла представляет собой последовательный вычислительный процесс. Однако она может быть разветвляющимся или циклическим вычислительным процессом. В последнем случае получаем схему алгоритма с вложенными циклами, которую называют ц и к л о м в ц и к л е. Математическими объектами, требующими при алгоритмизации применения структур с вложенными циклами, являются матрицы. В качестве примера рассмотрим следующую задачу.

Для определения содержания полезного компонента в руде на участке горизонтально залегающего марганцово-рудного месторождения, разрабатываемого открытым способом, была проведена разведка прямоугольной сеткой скважин. Результаты опробования представлены матрицей

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,30} \\ A_{2,1} & A_{2,2} & \dots & A_{2,30} \\ & & \dots & \\ A_{20,1} & A_{20,2} & \dots & A_{20,30} \end{pmatrix}.$$

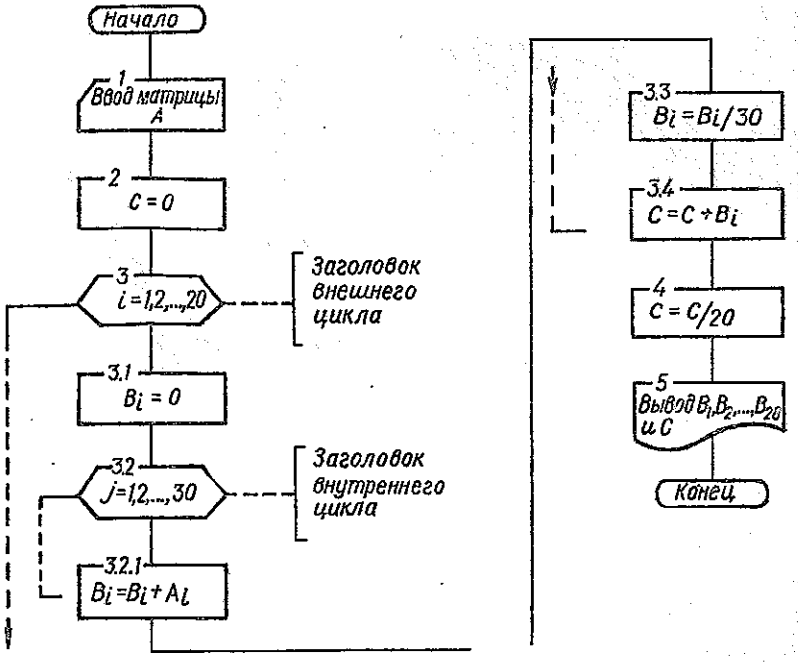


Рис. 2.11. Схема алгоритма циклического вычислительного процесса типа цикл в цикле

Шаг сетки скважин равен ширине заходки. Пробы, принадлежащие одной заходке, помещены в одной строке матрицы. Таким образом, участок будет отработан двадцатью заходками (матрица имеет двадцать строк) по тридцать проб в каждой (тридцать столбцов). Необходимо установить среднее содержание полезного компонента в руде для каждой заходки $B = (B_1, B_2, \ldots, B_{20})$, $B_i = \sum_{j=1}^{30} A_{ij}/30$, а также по участку в целом $C = \sum_{i=1}^{20} B_i/20$.

Очевидно, что для выбора строк матрицы потребуется цикл с параметром i ($i = 1, 2, \ldots, 20$). В области этого цикла из каждой строки матрицы следует выбирать поочередно элементы всех столбцов, для чего понадобится внутренний цикл с параметром j ($j = 1, 2, \ldots, 30$). Схема алгоритма для решения приведенной задачи изображена на рис. 2.11. Вначале в нем организуется накопление суммы слагаемых, а затем деление этой суммы на количество слагаемых. В отличие от предыдущего примера для сумм не введены вспомогательные переменные; промежуточные значения результатов вычислений хранятся в тех же ячейках B_i, $i = \overline{1,20}$, и C, в которые после завершения соответствующих циклов будут помещены окончательные результаты. Переменной C присваивается нулевое значение перед внешним циклом, а переменным B_i — перед внутренним.

Циклический вычислительный процесс итеративного типа характеризуется тем, что выход из цикла происходит при выполнении некоторого условия, на значение которого влияют вычисления, выполняемые в области цикла. Указанный процесс можно представить обобщенной схемой алгоритма, показанной на рис. 2.12. Параметру цикла присваивается значение, определяемое арифметическим выражением (блок *1*), после чего проверяется выполнение некоторого логического условия (блок *2*). Если оно не выполняется, то производятся вычисления, задаваемые блочными символами области цикла (блок *3*), и процесс повторяется до тех пор, пока логическое условие не будет удовлетворено.

Пример 1. Определим квадратный корень из числа x по итерационной формуле Ньютона. Обозначим $y = \sqrt{x}$. Итерационная формула Ньютона записывается так:

$$y = \lim_{n \to \infty} y_n,$$

где $y_n = 0{,}5\left(y_{n-1} + \frac{x}{y_{n-1}}\right)$, $y_0 = 1$.

Поскольку количество итераций должно быть конечным, вычисления прекращаются, когда значение y_n, полученное на n-й итерации, будет отличаться от предыдущего (y_{n-1}) не более чем на заданную малую величину ε, т. е. $|y_n - y_{n-1}| \leqslant \varepsilon$.

Согласно итерационной формуле Ньютона

$$\Delta y = y_n - y_{n-1} = 0{,}5\left(\frac{x}{y_{n-1}} - y_{n-1}\right);$$

$$y_n = y_{n-1} + \Delta y.$$

Полученное значение Δy прибавляется к y_{n-1} до тех пор, пока оно не станет меньше наперед заданной величины ε. Фрагмент схемы алгоритма изображен на рис. 2.13.

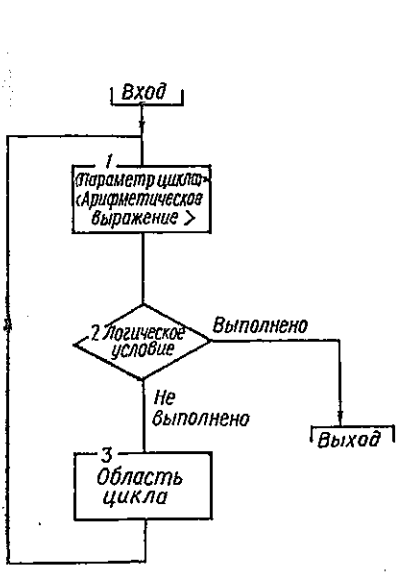


Рис. 2.12. Обобщенная схема алгоритма циклического вычислительного процесса итеративного типа

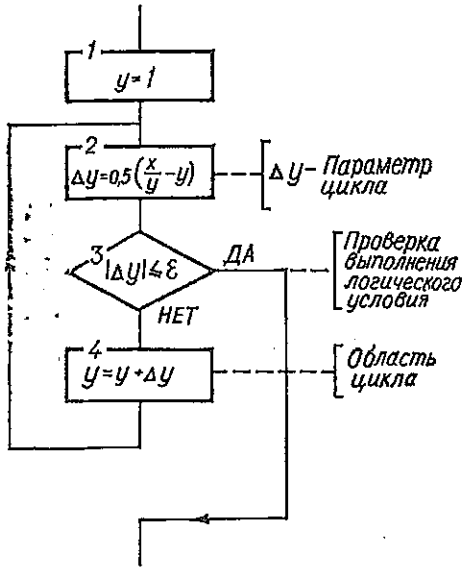


Рис. 2.13. Фрагмент схемы алгоритма вычисления квадратного корня из числа по итерационной формуле Ньютона

Пример 2. Вычислим значение $\sin x$ путем разложения в бесконечный сходящийся ряд

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Накопление суммы S членов ряда производим до тех пор, пока очередное слагаемое по абсолютной величине не станет меньше наперед заданной положительной величины ε.

Обозначим n-е слагаемое ряда через v_n, которое можно вычислить по следующей итерационной формуле:

$$v_n = v_{n-1} \frac{-x^2}{(2n-1)(2n-2)},$$

$$n = 2, 3, \dots; \quad v_1 = x.$$

Фрагмент схемы алгоритма вычисления $\sin x$ показан на рис. 2.14. В блоках *1—3* формируются начальные значения переменных, необходимые для последующих вычислений. Параметр цикла v вычисляется в блоке *4* по итерационной формуле. В блоке *5* проверяется логическое условие, при выполнении которого циклический процесс прерывается. Накопление суммы слагаемых ряда и формирование номера очередного слагаемого выполняется в блоках *6* и *7*.

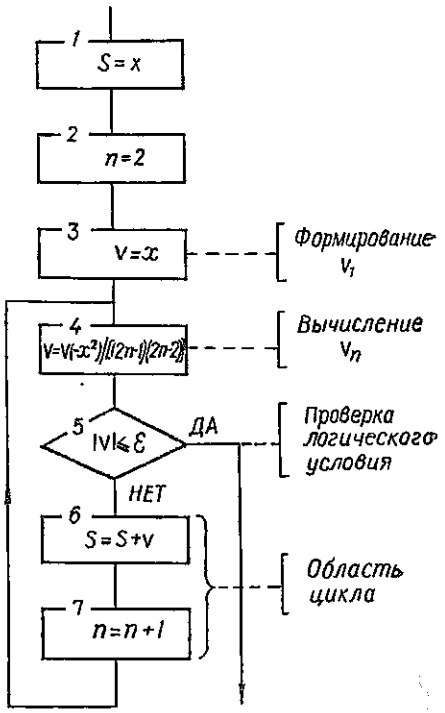


Рис. 2.14. Фрагмент схемы алгоритма вычисления $\sin(x)$ путем разложения в ряд

3. АРИФМЕТИЧЕСКИЕ ОСНОВЫ ЦИФРОВОЙ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

3.1. Системы счисления

Все операции в ЭВМ выполняются над кодами, посредством которых представлены числа и символы. Для физического отображения отдельных разрядов кода применяют электронные устройства, имеющие на выходе два уровня напряжения — высокое и низкое. Им ставят в соответствие два символа — 1 и 0. Такие устройства получили название двоичных. Совокупность двоичных устройств, представляющих многоразрядный двоичный код, называют *регистром*.

Система счисления — это способ представления чисел цифровыми знаками (цифрами). Количество цифр, используемых для записи чисел, называют основанием системы счисления. Например, в десятичной системе для записи чисел используется десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Основание этой системы счисления — число 10.

Число можно представить в позиционной форме, в которой значение (вес) цифры зависит от ее положения (позиции) в ряду цифр, изображающих число. Так, в десятичном числе 55,5 первая цифра слева означает количество десятков, вторая — количество единиц, третья — количество десятых долей единицы. Основание системы

счисления показывает, во сколько раз изменится вес цифры при перестановке ее на соседнюю позицию.

Кроме того, число можно записать в развернутой форме, т. е. в виде суммы парных произведений коэффициентов числа на степени основания, равные весу соответствующей позиции:

$$a_n a_{n-1} \ldots a_1 a_0, a_{-1} a_{-2} \ldots a_{-m} = a_n S^n + a_{n-1} S^{n-1} + \cdots$$
$$\cdots + a_1 S^1 + a_0 S^0 + a_{-1} S^{-1} + a_{-2} S^{-2} + \cdots + a_{-m} S^{-},$$

где a_i — коэффициент, стоящий в i-м разряде; S — основание системы счисления.

Например, число 1975,36 в развернутой форме записывается как $1 \cdot 10^3 + 9 \cdot 10^2 + 7 \cdot 10^1 + 5 \cdot 10^0 + 3 \cdot 10^{-1} + 6 \cdot 10^{-2}$. При переходе от развернутой формы к позиционной записываются последовательно коэффициенты при степенях оснований, причем запятая ставится после коэффициента с весом S^0: $2 \cdot 10^2 + 6 \cdot 10^1 + 1 \cdot 10^0 + 3 \cdot 10^{-1} + 0 \cdot 10^{-2} + 7 \cdot 10^{-3} = 261{,}307$.

Задаваясь различными основаниями S, можно получать различные системы счисления. Рассмотрим некоторые из них, применяемые в вычислительной технике.

Двоичная система счисления. Для изображения чисел в этой системе используют цифры 0 и 1, основание $S = 2$. Чтобы записать десятичное число в двоичной системе, можно использовать развернутую форму записи исходного числа по основанию $S = 2$. Например, $13_{(10)} = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1101_{(2)}$, $27_{(10)} = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11\,011_{(2)}$.

Для перевода целого десятичного числа в двоичную систему счисления его следует разделить на основание двоичной системы. Полученный результат снова делится на основание двоичной системы, и так до тех пор, пока частное не станет меньше основания S. Число в двоичной системе запишется в виде последнего частного и остатков деления, начиная с последнего. Переведем, к примеру, десятичное число 117 в двоичную систему счисления: $117 : 2 = 58$. (1); $58 : 2 = 29$. (0); $29 : 2 = 14$. (1); $14 : 2 = 7$. (0); $7 : 2 = 3$. (1); $3 : 2 = 1$. (1). Выписывая последовательно последнее частное и остатки деления (взятые в скобки), получаем: $117_{(10)} = 111\ 01\ 01_{(2)}$.

При переводе в двоичную систему правильной десятичной дроби ее умножают на основание $S = 2$, затем на это же основание умножают дробные части получающихся произведений. Число в двоичной системе образуют последовательно записанные целые части полученных произведений, начиная с первого. Например, при переводе числа 0,125 в двоичную систему записываем: $0{,}125 \cdot 2 = (0), 25$; $0{,}25 \cdot 2 = (0){,}5$; $0{,}5 \cdot 2 = (1), 0$. Последовательно выписывая целые части полученных произведений (взятые в скобки), получаем: $0{,}125_{(10)} = 0{,}001_{(2)}$.

Не всякую правильную десятичную дробь можно перевести в двоичную конечной длины, поэтому следует ограничиваться разумным количеством двоичных разрядов. Для примера возьмем число 0,67: $0{,}67 \cdot 2 = (1), 34$; $0{,}34 \cdot 2 = (0), 68$; $0{,}68 \cdot 2 = (1), 36$; $0{,}36 \cdot 2 = (0){,}72$; $0{,}72 \cdot 2 = (1), 44$; $0{,}44 \cdot 2 = (0), 88$; $0{,}88 \cdot 2 = (1), 76$; $0{,}76 \times$

$\times 2 = (1), 52, \ldots$ С точностью до восьми значащих цифр получим $0{,}67_{(10)} = 0{,}10101011 \ldots_{(2)}$.

В смешанных дробях переводят отдельно целые и отдельно дробные части по приведенным правилам, а затем записывают совместно: $117{,}125_{(10)} = 111\ 01\ 01{,}001_{(2)}$.

Таблица 3.1. Перевод восьмеричных цифр в двоичную систему счисления

Восьмеричная цифра	Двоичная триада	Восьмеричная цифра	Двоичная триада
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

Восьмеричная система счисления. Для изображения чисел в этой системе используют цифры 0,1,2,3,4,5,6,7, основание $S = 8$. Перевод десятичных чисел в восьмеричную систему производится по тем же правилам, что и для двоичной системы, но с основанием $S = 8$. Например, перевод числа 117 записывается следующим образом: $117 : 8 = 14.\ (5);\ 14 : 8 = (1).\ (6)$. Получаем $117_{(10)} = 165_{(8)}$.

Восьмеричная система счисления применяется в вычислительной технике как вспомогательная для более компактного представления двоичных чисел при обмене информацией между ЭВМ и внешними устройствами, а также между ЭВМ и оператором. Каждому разряду числа в восьмеричной системе можно поставить в соответствие трехразрядное двоичное число — триаду, и наоборот (табл. 3.1). Таким образом, перевод восьмеричных чисел в двоичную систему счисления производится заменой каждой цифры восьмеричного числа соответствующей двоичной триадой. При обратном переводе двоичные триады заменяются их восьмеричным эквивалентом, причем разбиение двоичного числа на триады осуществляется вправо и влево от запятой. Например, $3641_{(8)} = 011\ 110\ 100\ 001_{(2)}$; $11\ 010\ 110{,}111\ 010\ 1_{(2)} = (0)11\ 010\ 110{,}111\ 010\ 1(00)_{(2)} = 326{,}724_{(8)}$. Для получения полных триад двоичное число можно дополнять нулями слева от целой части и справа от дробной, отчего значение числа не изменяется.

Шестнадцатиричная система счисления. В этой системе числа изображаются с помощью шестнадцати символов (цифр), основание $S =$

Таблица 3.2. Шестнадцатиричные цифры и их десятичные эквиваленты

Шестнадцатиричные цифры	Десятичные эквиваленты	Шестнадцатиричные цифры	Десятичные эквиваленты
0	0	8	8
1	1	9	9
2	2	*A*	10
3	3	*B*	11
4	4	*C*	12
5	5	*D*	13
6	6	*E*	14
7	7	*F*	15

Таблица 3.3. Перевод шестнадцатиричных цифр в двоичную систему счисления

Шестнадцатиричная цифра	Двоичная тетрада	Шестнадцатиричная цифра	Двоичная тетрада
0	0000	8	1000
1	0001	9	1001
2	0010	*A*	1010
3	0011	*B*	1011
4	0100	*C*	1100
5	0101	*D*	1101
6	0110	*E*	1110
7	0111	*F*	1111

= 16. Поскольку в десятичной системе счисления имеется только десять цифровых символов, то для обозначения остальных применяют заглавные латинские буквы A, B, C, D, E и F. Соответствие шестнадцатиричных символов десятичным эквивалентам приведено в табл. 3.2. Перевод чисел в шестнадцатиричную систему счисления осуществляется по тем же правилам, что и для двоичной системы, но с основанием $S = 16$. К примеру, перевод числа 2654 записывается как 2654 : 16 = 165. (14); 165 : 16 = (10).(5). В результате получаем $2654_{(10)} = A\ 5\ E_{(16)}$, поскольку $10_{(10)} = A_{(16)}$, $14_{(10)} = E_{(16)}$.

Шестнадцатиричная система счисления тоже применяется в вычислительной технике как вспомогательная, для более компактного представления двоичных чисел при обмене информацией между процессором ЭВМ, внешними устройствами и оператором. Каждой цифре шестнадцатиричной системы счисления можно поставить в соответствие четырехразрядное двоичное число — тетраду, и наоборот (табл. 3.3).

Перевод шестнадцатиричных чисел в двоичную систему счисления производится заменой каждой цифры исходного числа двоичной тетрадой: $C5F2_{(16)} = 1100\ 0101\ 1111\ 0010_{(2)}$. При обратном переводе двоичные тетрады заменяются их шестнадцатиричными эквивалентами, причем разбиение двоичного числа на тетрады осуществляется вправо и влево от запятой: $1101\ 0111{,}1000\ 0011\ 1011_{(2)} = D\ 7{,}83\ B_{(16)}$.

Двоично-десятичный код. Поскольку все символы в ЭВМ представлены в двоичном коде, то для изображения десятичных цифр применяется двоично-десятичный код: каждая десятичная цифра заменяется эквивалентной ей двоичной тетрадой: $469{,}75_{(10)} = 010001101001{,}01110101_{(2-10)}$.

Десятичные числа переводятся в двоично-десятичный код механически на устройствах подготовки данных перед вводом в ЭВМ. Двоично-десятичный код числа не соответствует представлению этого числа в двоичной системе счисления, в которой ЭВМ выполняет математические операции, поэтому до решения задачи числа по специальной программе переводятся вычислительной машиной в двоичную систему. Результаты решения также по специальной программе переводятся из двоичной системы счисления в двоично-десятичный код, после чего поступают на устройства вывода результатов, которые представляют их в десятичной системе счисления.

3.2. Представление чисел в памяти ЭВМ

Как уже отмечалось, память ЭВМ состоит из отдельных пронумерованных ячеек, содержащих одинаковое количество двоичных разрядов. Обращение можно производить ко всей ячейке, но не к ее части.

В ЭВМ второго поколения данные представлены в виде машинных слов; для каждого слова отводится ячейка памяти с определенным числом двоичных разрядов (например, у ЭВМ «Минск-22» машинное слово имеет длину 37 двоичных разрядов). Длина любого числа равна длине машинного слова. В ЕС ЭВМ память состоит из ячеек, содержащих по восемь двоичных разрядов (байт). Данные могут занимать

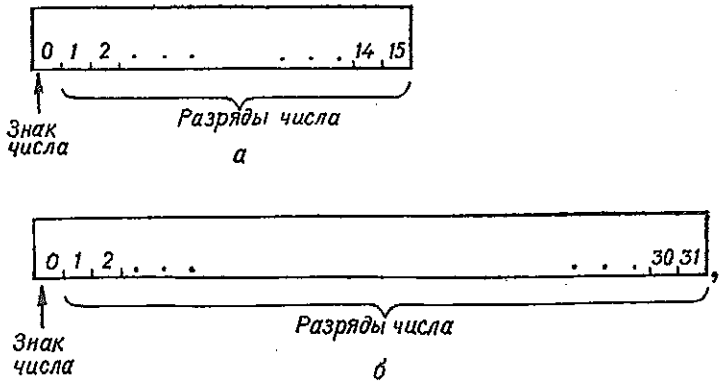


Рис. 3.1. Форматы чисел для ЕС ЭВМ с фиксированной запятой при длине числа:
а — 2 байта; *б* — 4 байта

в памяти одну ячейку (один байт) или несколько смежных (несколько байтов). Для чисел в ЕС ЭВМ используются поля памяти длиной 2 (полуслово), 4 (слово) или 8 (двойное слово) байтов.

В ЭВМ применяют естественную (с фиксированной запятой) и нормальную (с плавающей запятой) формы представления чисел. Форма числа с *фиксированной запятой* — это запись числа в виде целой и дробной частей, отделенных друг от друга запятой; место запятой зафиксировано и для каждой части числа отведено определенное количество разрядов. В ЕС ЭВМ числа с фиксированной запятой могут иметь длину 2 или 4 байта, запятая находится после младшего разряда (рис. 3.1). Таким образом, естественная форма используется для представления величин целого типа, максимальные абсолютные значения которых составляют $2^{15} - 1$ при длине 2 байта и $2^{31} - 1$ при длине 4 байта.

Форма числа с *плавающей запятой* имеет вид произведения двух сомножителей:

$$X = mS^p,$$

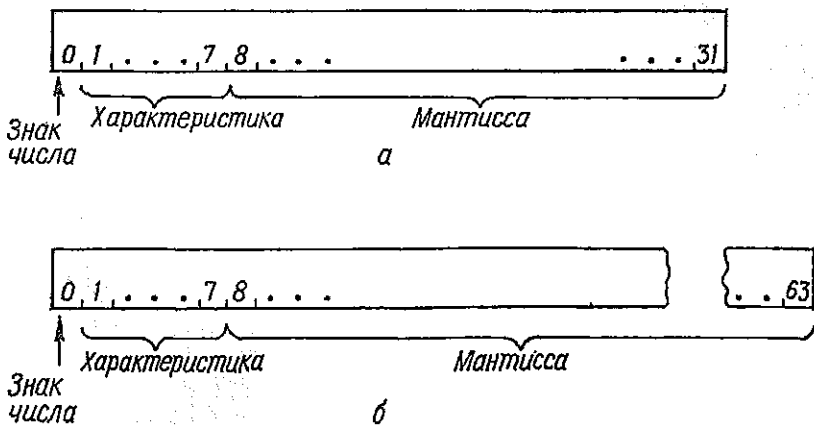


Рис. 3.2. Форматы чисел для ЕС ЭВМ с плавающей запятой при длине числа:
а — 4 байта; *б* — 8 байтов

где m — мантисса числа (цифровая часть числа с запятой, помещенной перед старшей значащей цифрой, $m < 1$); S — основание системы счисления; p — порядок числа, т. е. целое число, показывающее, на сколько разрядов вправо (если $p > 0$) или влево (если $p < 0$) следует переместить запятую в мантиссе, чтобы получить исходное число X.

Например, $456{,}2 = 0{,}4562 \cdot 10^3$, $m = 0{,}4562$, $p = 3$; $0{,}371 = 0{,}371 \cdot 10^0$, $m = 0{,}371$, $p = 0$; $0{,}00259 = 0{,}259 \cdot 10^{-2}$, $m = 0{,}259$, $p = -2$.

В ЕС ЭВМ числа с плавающей запятой могут иметь длину 4 или 8 байтов, порядок числа заменяется характеристикой $x = p + 64$ (рис. 3.2). Диапазон значений чисел составляет от 10^{-78} до 10^{75}. Числа представлены с точностью до 7 значащих десятичных разрядов при длине 4 байта и 16 — при длине 8 байтов.

Помимо чисел вычислительные машины ЕС могут оперировать также с различными символами. Каждому символу выделяется один байт памяти. Если необходимо оперировать с цепочкой символов (строкой), в памяти ЭВМ выделяется соответствующее количество байтов.

4. ОСНОВЫ ПРОГРАММИРОВАНИЯ НА АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ ФОРТРАН

4.1. Назначение и развитие языка ФОРТРАН

Язык программирования ФОРТРАН предназначен для описания алгоритмов и решения на ЭВМ инженерных и научно-технических задач. Благодаря ряду преимуществ (простота и выразительность средств языка, эффективность оттранслированных программ, универсальность операторов для обращения к внешним устройствам и др.) он получил широкое распространение.

Название языка образовано из двух английских слов for(mula) tran(slator), что означает «переводчик формул». Это один из первых алгоритмических языков, созданный для программирования задач вычислительной математики. Начальная версия языка разработана сотрудниками фирмы IBM (International Business Machines Corp., США) в 1954 г., а в 1957 г. начал работать первый транслятор. В дальнейшем язык ФОРТРАН непрерывно развивался и улучшался. В 1962 г. его усовершенствованный вариант ФОРТРАН-IV был стандартизирован в США и ряде других стран.

При последующей модернизации языка были сняты некоторые ограничения, присущие стандартному ФОРТРАНу, что упростило программирование, сделало его более доступным и удобным для пользователя. Именно эта модернизированная версия, реализованная, в частности, в операционной системе ЕС ЭВМ, изложена в настоящем учебном пособии.

4.2. ФОРТРАН-программа

Программа, записанная на языке ФОРТРАН, состоит из одного или нескольких модулей. Первым в ФОРТРАН-программе размещают головной модуль, после которого в произвольной после-

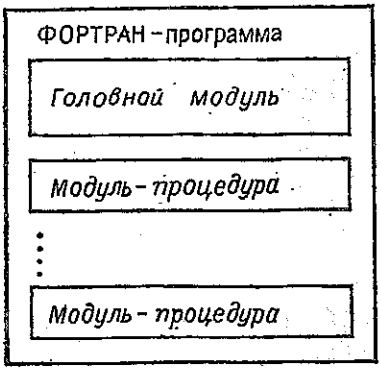


Рис. 4.1. Структура ФОРТРАН-программы

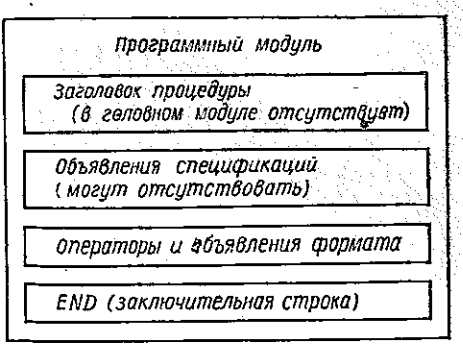


Рис. 4.2. Структура программного модуля

довательности располагаются другие модули-процедуры — внешние функции и подпрограммы (рис. 4.1). Каждый программный модуль включает предложения и комментарии.

Предложения делятся на выполняемые (операторы) и невыполняемые (объявления). Операторы определяют действия, которые необходимо выполнить при решении задачи, а объявления описывают свойства и способы редактирования данных, являются заголовками внешних процедур, указывают размещение данных и т. п. Любое предложение может быть помечено меткой — целым числом без знака. Наличие метки дает возможность ссылаться на это предложение, обращаться к нему из других предложений данного программного модуля.

Комментарии — это текст, написанный символами ДКОИ [1] и используемый для пояснений к программе. Комментарии не оказывают никакого влияния на выполнение программы.

Структура программного модуля показана на рис. 4.2. Если модуль является внешней процедурой, то в начале помещается ее заголовок, после которого следует тело модуля. В головном модуле заголовок отсутствует. Тело модуля включает спецификации (при необходимости) и раздел операторов, после которого записывается заключительная строка — ключевое слово END, указывающее конец текста программного модуля.

Операторы программы выполняются с первого оператора головного модуля в последовательности их записи до тех пор, пока не встретится оператор, указывающий на необходимость прервать естественную последовательность и перейти к помеченному оператору, вычислить функцию, обратиться к подпрограмме или повторить выполнение определенной группы операторов.

Для записи предложений ФОРТРАН-программы используются цифры (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), заглавные буквы латинского алфавита и специальные символы (+ — / * = , . ' ( ) $ ␣). Коммен-

[1] ДКОИ — двоичный код обмена информацией — содержит латинские и русские буквы, арабские цифры, а также ряд других символов.

Таблица 4.1. Перечень ключевых слов языка ФОРТРАН

Ключевое слово	Перевод	Ключевое слово	Перевод
CALL	Вызвать	IF	Если
COMMON	Общий	INTEGER	Целый
COMPLEX	Комплексный	LOGICAL	Логический
CONTINUE	Продолжать	PRINT	Печатать
DIMENSION	Размерность	READ	Читать
DO	Выполнять	REAL	Вещественный
DOUBLE PRECISION	Двойная точность	RETURN	Вернуться
END	Конец	STOP	Стоп
EXTERNAL	Внешний	SUBROUTINE	Подпрограмма
FORMAT	Формат	WRITE	Писать
FUNCTION	Функция		
GO TO	Идти к		

тарии могут содержать также буквы русского алфавита и другие символы ДКОИ.

В состав предложений на языке ФОРТРАН входят также своеобразные неделимые символы — ключевые слова (табл. 4.1), знаки логических операций, операций отношения и логические константы.

Составленная ФОРТРАН-программа наносится на перфокарты и вводится в ЭВМ для трансляции и исполнения. Стандартная перфокарта содержит 80 колонок. Пробивкой в колонке одного или нескольких отверстий кодируется тот или иной символ (рис. 4.3).

Для удобства перфорации ФОРТРАН-программа записывается на специальном бланке (рис. 4.4). Каждое предложение начинается с новой строки и помещается в позициях 7—72. Если предложение не вмещается в одну строку, его переносят на следующую с пометкой в 6-й позиции каким-либо символом (кроме нуля). Встречающиеся в операторах пробелы транслятором игнорируются. Метка (если она есть) помещается в позициях 1—5.

Комментарии записываются в позициях 2—72, при этом в первой позиции помещается символ С. Строки с комментариями размещаются

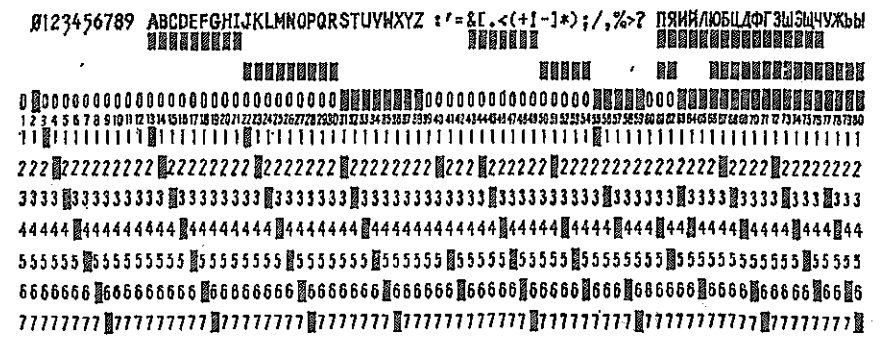


Рис. 4.3. Кодирование символов на перфокарте

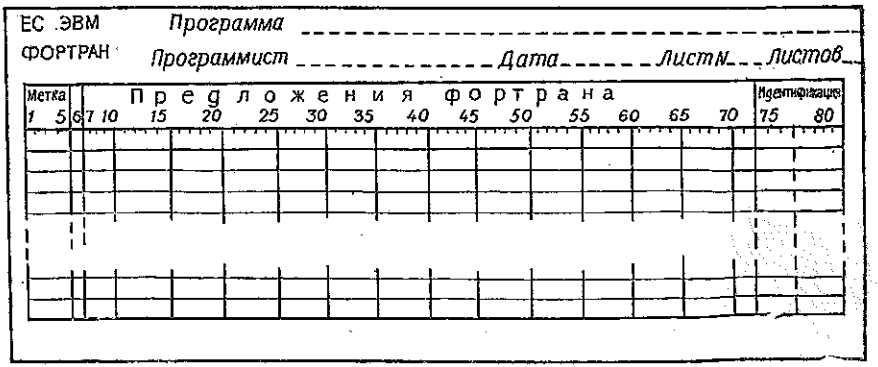


Рис. 4.4. Бланк для записи ФОРТРАН-программы

в любом месте программы. Записи в позициях 73—80 (как и комментарии) транслятором не обрабатываются, их можно использовать для нумерации строк (карт) или других целей.

4.3. Данные

Данными в алгоритмических языках называют информацию, представленную символами и обрабатываемую при выполнении программы. В языке ФОРТРАН используют данные трех видов: константы, переменные и массивы. Каждый из этих видов может быть следующего типа: целый (INTEGER), вещественный (REAL), двойной точности (DOUBLE PRECISION), комплексный (COMPLEX), логический (LOGICAL). Константы могут быть также текстового типа.

Константы — это данные, которые не изменяются в процессе вычислений.

Целая числовая константа (целое число) представляет собой десятичное число, не содержащее дробной части и записанное в виде последовательности цифр. Перед числом может стоять знак «плюс» или «минус». Отсутствие знака свидетельствует, что число положительное. Например: +17 —54 2403 0

В ЕС ЭВМ целая числовая константа занимает четыре байта памяти, причем старший разряд отводится под знак числа (рис. 4.5). Следовательно, наибольшее число, которое можно записать в память ЭВМ, составляет $2^{31} - 1 = 2147483647$.

Вещественная числовая константа (вещественное число) — это положительное или отрицательное десятичное число, содержащее целую и дробную части.

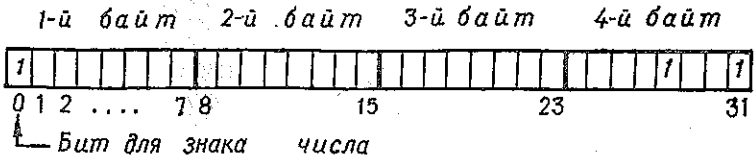


Рис. 4.5. Представление в ЭВМ числовой константы целого типа. (Для примера показана запись числа —9.)

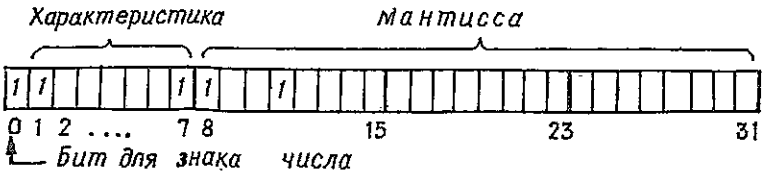


Рис. 4.6. Представление вещественного числа в памяти ЕС ЭВМ. (Для примера показана запись числа $-9 = -0{,}5625 \cdot 16^1$.)

Вещественное число в форме F (fixed — фиксированный) записывается в виде последовательности цифр и точки, которая отделяет дробную часть числа. Если целая или дробная часть числа отсутствует, она может быть опущена. Например:

$$+12.6 \quad -781.31 \quad 3.1415926 \quad 15. \quad .1$$

Вещественное число в форме E (exponent — показатель степени) состоит из мантиссы и десятичного порядка. Мантисса — это целое или вещественное число в форме F. Порядок обозначается буквой E, за которой следует целое число — не более двух цифр со знаком или без него:

Запись в форме E	—5.4E3	.21E+05	17.1E—3	2.3E—7	1E6
Обычная запись	—5400	21 000	0,0171	$2{,}3 \cdot 10^{-7}$	10^6

В памяти ЭВМ вещественное число записывается в виде порядка и мантиссы независимо от формы представления. При этом старший двоичный разряд четырехбайтного слова отводится под знак числа (рис. 4.6), следующие семь — для порядка[1], а остальные 24 — для мантиссы. Диапазон абсолютных значений чисел, с которыми может работать ЭВМ, составляет $5{,}4 \cdot 10^{-79} \ldots 7{,}2 \cdot 10^{75}$, а также нуль. Три байта мантиссы обеспечивают точность, соответствующую работе с семизначными десятичными числами.

Константы двойной точности отличаются тем, что при записи порядка числа в них вместо буквы E пишется буква D: .2718281828D1. В памяти ЭВМ константе двойной точности отводится восемь байтов, семь из которых занимает мантисса. Это эквивалентно десятичному числу с 17 значащими цифрами.

Комплексная константа записывается в виде упорядоченной пары вещественных чисел, разделенных запятой и заключенных в скобки:

Запись на ФОРТРАНе	*Обычная запись*
(2.8,—3.7)	$2{,}8 - i \cdot 3{,}7$
(16.2,.25E3)	$16{,}2 + i \cdot 250$
(1E2,1E2)	$100 + i \cdot 100$
(1D1,2D1)	$10 + i \cdot 20$

Логическая константа представляется в виде .TRUE. (истина) или .FALSE. (ложь). В памяти ЭВМ она занимает четыре байта.

[1] В разряды 1—7 записывается не порядок p, а характеристика $p' = p + 64$, что технически реализуется более просто. При $p_{max} = 127$ порядок (при основании $N = 16$) может находиться в пределах $-64 < p < 63$.

Текстовая константа — это любая последовательность символов ДКОИ. Допускаются две формы записи текстовых констант:

последовательность символов, записанная после кода wH (где w — количество символов в текстовой константе), например:

25H ␣ РЕШЕНИЕ ␣ УРАВНЕНИЯ ␣ F (X) = 0

последовательность символов, заключенная в апострофы:

'␣ РЕШЕНИЕ ␣ УРАВНЕНИЯ ␣ F (X) = 0'

Число символов в текстовой константе не должно превышать 255.

Переменные — это данные, которые в процессе выполнения программы могут принимать различные значения. Переменные обозначают символическими именами, образуемыми из букв и цифр. Имя может содержать до шести символов и должно начинаться с буквы: K, X1, Q4A, TEMP26, ALFA.

Тип переменных может быть указан в объявлениях. Отсутствие последних предполагает неявное указание типа. При этом если символическое имя начинается с букв I, J, K, L, M или N, оно связывается с целым типом, в остальных случаях — с вещественным. Например, переменные I, K2, NUM — целого типа, а ALFA, Q31, VEL — вещественного.

В памяти ЭВМ каждой переменной отводится определенное число байтов: по четыре — переменным целого, вещественного и логического типа и по восемь — переменным двойной точности и комплексным.

Массив — это упорядоченный набор данных, обозначенных одним именем. Имя массива образуется по тем же правилам, что и имя переменной.

В ФОРТРАНе используются одномерные (векторы), двумерные (матрицы) и трехмерные массивы. Если трансляция программы осуществляется в операционной системе ЕС ЭВМ, число измерений массива может достигать семи.

Компоненты набора данных, образующих массив, называются элементами массива. Каждый элемент массива обозначается именем, после которого в скобках помещается список индексов: LS (3), Q (2, 6), Z (1,3,N).

Индексы отделяют друг от друга запятыми. Количество индексов в списке равно числу измерений (размерности) массива. Индексом может быть целая константа (без знака), целая переменная или арифметическое выражение. Например, LS (1 + 3 * K), R4 (5, M — 1). Индекс принимает только целое значение, большее нуля. Если в результате вычисления арифметического выражения получается вещественное число, оно автоматически преобразуется в целый тип, дробная часть при этом отбрасывается.

Для каждого массива в ЭВМ отводится соответствующее количество байтов памяти. Длина каждого элемента задается объявлением (явным или неявным), а количество элементов массива определяется верхними границами его индексов. Если, например, двумерный массив R4 содержит 5 строк и 14 столбцов, а для каждого элемента отводится четыре байта памяти, то общий объем памяти для этого массива составит $5 \cdot 14 \cdot 4 = 280$ байт.

Верхние границы индексов каждого используемого в программе массива должны быть указаны в объявлении массивов, которое помещается в начале программного модуля, перед разделом операторов. Объявление массивов записывается так:

DIMENSION $ом_1$ $ом_2$, ..., $ом_n$

где DIMENSION — ключевое слово ФОРТРАНа; $ом_i$ — описание i-го массива.

Каждое описание массива состоит из имени и списка границ, заключенного в скобки:

имя (список границ)

Список границ состоит из одного или нескольких элементов — целых чисел [1], разделенных запятыми. Количество элементов в списке равно размерности массива, а величина каждого из них указывает верхнюю границу по соответствующему измерению; нижняя граница установлена равной единице.

Пример объявления массивов:

DIMENSION ␣ LS(20), ␣ Q(10, 30), ␣ R4(5,14), ␣ Z(6, 6, 3)

Эта запись указывает транслятору, что в программе используются четыре массива: одномерный LS, двумерные Q и R4, трехмерный Z. В скобках после имени каждого массива указаны верхние границы индексов.

Тип переменных, являющихся элементами массива, определяется как при неявном указании — по первой букве имени массива. Длина каждого элемента составляет четыре байта. В памяти ЭВМ элементы массива размещаются в такой последовательности, при которой индексы изменяются следующим образом: сначала увеличивается первый индекс; второй возрастает на единицу каждый раз, когда заканчивается изменение первого, и т. д. Например, элементы матрицы

$$B = \begin{pmatrix} b_{1.1} & b_{1.2} & b_{1.3} \\ b_{2.1} & b_{2.2} & b_{2.3} \\ b_{3.1} & b_{3.2} & b_{3.3} \end{pmatrix}$$

размещаются в памяти ЭВМ в такой последовательности: $b_{1.1}$, $b_{2.1}$, $b_{3.1}$, $b_{1.2}$, $b_{2.2}$, $b_{3.2}$, $b_{1.3}$, $b_{2.3}$, $b_{3.3}$, т. е. по столбцам.

4.4. Объявления типа

Объявление типа используют, чтобы подтвердить неявное указание типа или изменить его, закрепив за каждым именем определенный тип данных: целый, вещественный, двойной точности, комплексный или логический; кроме того, объявление может содержать информацию о структуре массивов.

Объявление типа — это запись вида

тип v_1, v_2, ..., v_n

[1] В модулях-процедурах границы индексов могут быть указаны целыми переменными.

где *тип* — одно из ключевых слов, являющихся описателями типа INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL; v_i — имя переменной, массива, функции или описание массива.

Как и другие спецификации, объявления типа размещают в программном модуле перед разделом операторов (см. рис. 4.2). Например, в объявлениях типа

```
INTEGER, I, S, Q2, L
REAL K, MASSA, A(5, 12)
```

объявлены переменные целого типа I, S, Q2 и L, вещественные переменные K и MASSA, а также двумерный вещественный массив A, т. е. матрица, имеющая 5 строк и 12 столбцов. Все переменные имеют стандартную длину — четыре байта.

Если ФОРТРАН-программа ориентирована на трансляцию в операционной системе ЕС ЭВМ, то в описаниях типа можно указывать также длину и начальные значения.

4.5. Стандартные математические функции

Для вычисления часто встречающихся математических функций в языке ФОРТРАН существует набор *стандартных функций*. При обращении к ним в соответствующем месте программы (в операторе) ставится *указатель функции*, который записывается следующим образом:

f (*аргументы*)

где *f* — символическое имя функции. Аргументом (фактическим параметром) может быть не только число или переменная, но любое арифметическое выражение. Если аргументов несколько, они отделяются друг от друга запятой.

Приведем примеры указателей стандартных функций:

Обычная запись	*Запись на ФОРТРАНе*
ln 3,7	ALOG (3.7)
arctg h	ATAN (H)
$\sqrt{2aS}$	SQRT (2*A*S)
$e^{-\alpha t}$	EXP (—ALFA*T)
lg | sin ωt |	ALOG10 (ABS (SIN (W*T)))

Аргументы тригонометрических функций должны быть выражены в радианах. Если угол задан в градусах, необходимо выполнить соответствующие преобразования. Например, для вычисления sin 21° 30′ и cos 23°46′ следует записать

```
SIN (21.5 * 0.01745329)
COS (23 * 1.745329E — 2 + 46 * 2.908882E — 4)
```

Обратные тригонометрические функции вычисляются в радианах для главных значений. Кроме того, можно получить значение угла в градусах. Так, для вычисления в градусной мере функции

Таблица 4. 2. Стандартные функции языка ФОРТРАН

Вычисляемое значение	Указатель функции	Тип и длина		Примечания
		функции	аргумента	
$\sqrt{x}$	SQRT (X)	R4	R4	$x \geqslant 0$
$\sqrt{x}$	DSQRT (X)	R8	R8	$x \geqslant 0$
ln x	ALOG (X)	R4	R4	$x > 0$
lg x	ALOG10 (X)	R4	R4	$x > 0$
e^x	EXP (X)	R4	R4	$x < 174{,}6$
sin x	SIN (X)	R4	R4	$x < 2^{18}\pi$
cos x	COS (X)	R4	R4	$x < 2^{18}\pi$
tg x	TAN (X)	R4	R4	$x < 2^{18}\pi$
ctg x	COTAN (X)	R4	R4	$x < 2^{18}\pi$
arcsin x	ARSIN (X)	R4	R4	$\lvert x \rvert \leqslant 1$
arccos x	ARCOS (X)	R4	R4	$\lvert x \rvert \leqslant 1$
arctg x	ATAN (X)	R4	R4	—
arctg $\frac{x_1}{x_2}$	ATAN2 (X1, X2)	R4	R4	Кроме $x_1 = x_2 = 0$
sh x	SINH (X)	R4	R4	$\lvert x \rvert < 175{,}3$
ch x	COSH (X)	R4	R4	$\lvert x \rvert < 175{,}3$
th x	TANH (X)	R4	R4	$\lvert x \rvert < 175{,}3$
$\lvert x \rvert$	ABS (X)	R4	R4	—
$\lvert x \rvert$	IABS (X)	I4	I4	—
Re (x)	REAL (X)	R4	C8	Получение действительной части комплексного числа
Im (x)	AIMAG (X)	R4	C8	Получение мнимой части комплексного числа
$x_1 + ix_2$	CMPLX (X1, X2)	C8	R4	Получение комплексного числа из двух вещественных

arctg 0,53 записывают

57.29578*ATAN (0.53)

В ФОРТРАНе предусмотрено и вычисление функций с двойной точностью. Наиболее употребительные стандартные функции приведены в табл. 4.2.

4.6. Арифметические выражения

Арифметическое выражение — аналог алгебраической формулы — представляет собой компактную запись, указывающую, в какой последовательности должны быть выполнены арифметические действия для получения искомого результата.

Результатом вычисления арифметического выражения является числовое данное (целое, вещественное или комплексное число). Строятся арифметические выражения из операндов, знаков арифметических операций и скобок. Операндом может быть числовая константа, числовая переменная (простая или с индексом), указатель функции (стандартной, внутренней или внешней) или заключенное в скобки арифметическое выражение. Частным случаем арифметического выражения является один операнд.

Знаками арифметических операций являются: + (сложение), —(вычитание), / (деление), * (умножение), ** (возведение в степень).

При составлении арифметических выражений необходимо руководствоваться следующими правилами.

1. Арифметическое выражение записывается в одну строку. Например:

Обычная запись	*Запись на ФОРТРАНе*
$\frac{ab}{c}$	A*B/C или A/C*B
$\frac{a}{b} + c$	A/B + C

2. Два знака арифметических операций нельзя записывать рядом:

Обычная запись	*Запись на ФОРТРАНе*
$\frac{z}{-4}$	Z/(—4), а не Z/—4
x^{-2}	X**(—2), а не X**—2

3. Действия в арифметических выражениях выполняются в соответствии с общепринятым приоритетом: вычисление функций, возведение в степень, умножение и деление, сложение и вычитание.

4. Операции с одинаковым приоритетом выполняются в порядке их записи, т. е. слева направо. Исключение составляет операция возведения в степень. Например:

Запись на ФОРТРАНе	*Обычная запись*
A*B/C*D	$\frac{ab}{c}d$
A*B/C/D	$\frac{ab}{cd}$
A**B**C	$a^{(b^c)}$, а не $(a^b)^c$

5. Изменение последовательности вычислений задается скобками:

Запись на ФОРТРАНе	*Обычная запись*
A*B/(C*D)	$\frac{ab}{cd}$
A + B/(C + D)	$a + \frac{b}{c+d}$
(A + B)/(C + D)	$\frac{a+b}{c+d}$

6. Операнды в арифметическом выражении могут быть разного типа и различной длины. Результат принимает тип, определяемый следующим образом:

Тип операнда	*Тип результата*
COMPLEX и REAL	COMPLEX
COMPLEX и INTEGER	COMPLEX
REAL и INTEGER	REAL

Длина результата равна наибольшей из длин операндов.

7. Результат операции деления будет целого типа, если оба операнда целого типа; дробная часть в данном случае отбрасывается (не округляется). Например, когда M = 8, при записи M/3 результат равен 2 (INTEGER), а при M/3.0 — 2,666667 (REAL).

8. Возведение в степень выполняется следующим образом:

степень I — целого типа, основание A — целого, вещественного или комплексного типа:

$$A{**}I = \begin{cases} A*A*A* \ \ldots \text{ (I раз)} & \text{если } I>0; \\ 1. & \text{если } I=0; \\ 1/(A*A*A* \ \ldots \text{ (I раз))} & \text{если } I<0 \text{ и } A\neq 0; \\ \text{Не определено} & \text{если } I\leqslant 0 \text{ и } A=0; \end{cases}$$

степень B — вещественного типа; основание A — целого или вещественного типа:

$$A{**}B = \begin{cases} e^{B \ln A} & \text{если } A>0; \\ 0. & \text{если } A=0 \text{ и } B=0; \\ \text{Не определено} & \text{если } A=0 \text{ и } B<0 \text{ или} \\ & \text{если } A<0. \end{cases}$$

Комплексная степень недопустима. Примеры записи арифметических выражений:

Обычная запись	*Запись на ФОРТРАНе*
$\dfrac{a-b}{a+\dfrac{dx}{x-y}}$	(A — B)/(A + D * X/(X — Y))
$\sin^2 x + \cos x^2$	SIN (X) * * 2 + COS (X * * 2)
$\dfrac{a_i^{nx}}{\sqrt{a_i+1}}$	A (I) * * (N * X)/SQRT (A (I) + 1)
$\dfrac{\ln (\operatorname{arctg} y)^2}{S_{m,2}}$	ALOG (ATAN (Y) * * 2)/S (M, 2)
$\dfrac{1}{3}\sqrt{b^2+(x_i+8{,}3)^2}$	SQRT (B * B + (X (I) + 8.3) * * 2)/3
$\dfrac{\sqrt[3]{x^2}}{x+e^{-\sqrt{x}}}$	X * * (2./3.)/(X + EXP (— SGRT (X)))
$\dfrac{\sin x_{k,i}^2}{\operatorname{tg}\dfrac{\pi}{k}}$	SIN (X (K, I) * * 2)/TAN (3.141593/K)

4.7. Логические выражения

Логические выражения строятся из логических операндов, знаков логических операций и скобок.

Операнды логического выражения: логическая константа (.TRUE. или .FALSE.), логическая переменная (величина, принимающая одно

Таблица 4.3. Знаки логических операций и операций отношения

Запись на ФОРТРАНе	Название операции	Математическая запись	Название операции на английском языке
	Логические операции		
.NOT.	НЕ — логическое отрицание	$\neg$	Not
.AND.	И — логическое умножение	$\wedge$	And
.OR.	ИЛИ — логическое сложение	$\vee$	Or
	Операции отношения		
.LT.	Меньше	$<$	Less than
.LE.	Меньше или равно	$\leqslant$	Less or equal
.EQ.	Равно	$=$	Equal
.NE.	Не равно	$\neq$	Not equal
.GE.	Больше или равно	$\geqslant$	Greater or equal
.GT.	Больше	$>$	Greater than

Таблица 4.4. Результаты выполнения логических операций НЕ, И, ИЛИ

Значение операнда		Результат операции		
A	B	.NOT.A	A.AND.B	A.OR.B
.FALSE.	.FALSE.	.TRUE.	.FALSE.	.FALSE.
.FALSE.	.TRUE.	.TRUE.	.FALSE.	.TRUE.
.TRUE.	.FALSE.	.FALSE.	.FALSE.	.TRUE.
.TRUE.	.TRUE.	.FALSE.	.TRUE.	.TRUE.

из двух значений,—.TRUE. или .FALSE.), отношение (два арифметических выражения, разделенных знаком операции отношения), логическое выражение, заключенное в скобки. Частным случаем логического выражения является один операнд.

Используемые в ФОРТРАНе знаки логических операций и операций отношения приведены в табл. 4.3, а результат выполнения логических операций — в табл. 4.4.

Вычисления в логическом выражении выполняются слева направо в соответствии с приоритетом в такой последовательности операций: арифметические (с учетом их приоритета); отношения; НЕ; И; ИЛИ.

Изменение последовательности задается скобками. Результатом вычисления логического выражения является логическое данное (.TRUE. или .FALSE.).

Логические переменные должны быть объявлены. Они имеют длину 1 или 4 байта. Если в объявлениях длина не указана, то она устанавливается стандартной — 4 байта.

Примеры логических выражений:

Обычная запись	*Запись на языке ФОРТРАН*
$\varepsilon < 10^{-6}$	EPS.LT.1E — 6
$x^2 \geqslant z + 10 \wedge y < 2{,}5 + z$	X**2.GE.Z + 10.AND.Y.LT.2.5 + Z
$x \leqslant 0 \vee x > 4{,}7$ (рис. 4.7, *а*)	X.LE.0.0.OR.X.GT.4.7
$-3{,}2 < x \leqslant 5$ (рис. 4.7, *б*)	X.GT.—3.2.AND.X.LE.5.0

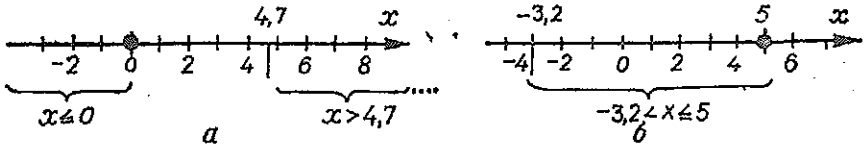


Рис. 4.7. Иллюстрация к примерам:
а — $x \leqslant 0 \vee x > 4{,}7$; *б* — $-3{,}2 < x \leqslant 5$

4.8. Операторы присваивания, перехода, останова

Оператор присваивания служит для вычисления арифметического или логического выражения и присваивания полученного значения некоторой переменной. Он записывается так:

$$v = s$$

где v — имя переменной или элемента массива; s — арифметическое или логическое выражение.

Символ «=» называют знаком присваивания.

В отличие от обычного равенства оператор присваивания описывает вычислительный процесс, протекающий во времени: вначале вычисляется значение s, записанное в правой части, а затем оно присваивается переменной v.

Различают арифметический и логический операторы присваивания. В арифметическом переменная v может быть целого, вещественного или комплексного типа. Если результат вычисления арифметического выражения s имеет тип или длину, не совпадающую с типом или длиной переменной v, то перед присваиванием полученное значение преобразуется к типу или длине переменной v.

В логическом операторе присваивания переменная v должна быть логического типа и ей присваивается значение логического выражения s.

Пример 1. После выполнения оператора присваивания M = M + 1 переменная M получит значение на единицу больше, чем до его выполнения. Пример наглядно показывает принципиальную разницу между оператором присваивания и равенством: подобное равенство вообще не имеет смысла.

Пример 2. Пусть дан оператор

```
A (J) = 5.64/SQRT (BT * * 2 + R)
```

Значение арифметического выражения присваивается j-му элементу массива A; значения переменных BT, R и J должны быть определены до выполнения оператора.

Пример 3. В операторе присваивания

```
L = X.LT.2.5
```

Логической переменной L присваивается значение .TRUE., если $x < 2{,}5$, и .FALSE. в противном случае.

Операторы перехода (или операторы GO TO) предназначены для того, чтобы при необходимости прервать естественную последовательность выполнения операторов и указать тот оператор-преемник, с которого должно быть продолжено выполнение программы.

Чаще других используется б е з у с л о в н ы й оператор перехода, который записывается так:

GO TO m

где m — метка оператора, который выполняется после оператора GO TO. Например, после оператора GO TO 15 будет выполняться оператор, помеченный меткой 15.

Метки локализованы в пределах программного модуля, поэтому переход допустим только к помеченному оператору, находящемуся в том же программном модуле, что и оператор GO TO.

Иногда возникает необходимость передать управление на тот или иной оператор в зависимости от значения некоторой переменной. В данном случае применяется в ы ч и с л я е м ы й оператор перехода:

GO TO (m_1, m_2, ..., m_n), i

где m_1, m_2, ..., m_n — метки операторов; i — имя переменной целого типа.

Этот оператор передает управление оператору, помеченному меткой m_i, если значение переменной i находится в пределах $i = 1...n$; в остальных случаях ($i \leqslant 0$ или $i > n$) выполняется следующий за ним оператор. Например, при N = 2 после оператора

GO TO (4, 10, 7), N

выполняется оператор с меткой 10, однако при N = 4 осуществляется переход к оператору, записанному после GO TO.

Оператор останова используется для прекращения вычислений и записывается в виде

STOP или STOP k

где k — число целого типа (не более пяти цифр).

Если вычисления прекращаются оператором STOP k, то он печатается на пультовой машинке и по числу k можно определить место в программе, где произошел останов.

Прекращение вычислений происходит также при достижении заключительной строки END. При этом транслятор сам формирует команду останова.

4.9. Условные операторы

Условные операторы (или операторы IF) предназначены для описания разветвляющихся вычислительных процессов, выполняемых различно в зависимости от значения выражения, входящего в состав оператора IF. Существует два вида условных операторов: арифметический и логический.

Условный арифметический оператор записывается следующим образом:

IF (a) m_1, m_2, m_3

где IF — ключевое слово; a — арифметическое выражение целого или вещественного типа; m_1, m_2, m_3 — метки операторов.

При выполнении этого оператора вначале вычисляется значение арифметического выражения a, после чего управление передается

Рис. 4.8. Графическое изображение условного арифметического оператора

Рис. 4.9. Схема алгоритма вычисления корней квадратного уравнения

одному из операторов с меткой: m_1 при $a<0$; m_2, если $a=0$; m_3 при $a>0$ (рис. 4.8).

Пример 1. Составить ФОРТРАН-программу для вычисления корней квадратного уравнения $ax^2+bx+c=0$.

Корни этого уравнения вычисляются по правилу: если дискриминант $D=b^2-4ac$ положителен ($D>0$), то корни вещественные разные:

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}, \tag{4.1}$$

$$x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}; \tag{4.2}$$

при $D=0$ корни вещественные равные:

$$x_1=x_2=-\frac{b}{2a}; \tag{4.3}$$

если $D<0$, корни сопряженные комплексные:

$$x_1=-\frac{b}{2a}+i\frac{\sqrt{4ac-b^2}}{2a}, \tag{4.4}$$

$$x_2=-\frac{b}{2a}-i\frac{\sqrt{4ac-b^2}}{2a}. \tag{4.5}$$

Схема алгоритма вычисления корней квадратного уравнения показана на рис. 4.9. Алгоритм начинается со ввода исходных данных: a, b и c (блок *1*). Затем вычисляется дискриминант D (блок *2*). В зависимости от его значения осуществляется переход к блокам *4*, *8* или *9*. Если $D>0$, вычисляются вещественные корни по формулам (4.1) и (4.2), а затем переменной N присваивается значение 0, которое свидетельствует о том, что ответы — вещественные числа (блоки *4*, *5*). Блок *6* выводит на печать результаты x_1, x_2 и признак N. После печати результатов происходит останов (блок *7*).

При $D=0$ значения x_1 и x_2 вычисляются по формуле (4.3), после чего управление передается блоку *5*. Если $D<0$, осуществляется переход к блоку *9* и вычисляются вещественная (r) и мнимая (s) части комплексных корней по формулам (4.4) и (4.5), а переменной N присваивается значение 1. Затем печатаются результаты (r, s) и признак (N), указывающий, что полученные числа — вещественная и мнимая части сопряженных комплексных корней (блок *10*). Эта ветвь программы также завершается остановом (блок *11*).

В ФОРТРАН-программе оператор ввода записывается как

READ *m*, *список*

а оператор вывода

PRINT *m*, *список*

Здесь READ, PRINT — ключевые слова ФОРТРАНа; *m* — метка объявления формата, в котором указано, как представлены числа на перфокартах (при вводе) и на бумажной ленте (при выводе); *список* содержит имена переменных и массивов, отделенных друг от друга запятой.

Объявления формата — FORMAT (...) — можно помещать в любом месте программы (см. рис. 4.2) [1].

ФОРТРАН-программа, соответствующая алгоритму (см. рис. 4.9), записывается следующим образом:

```
C ПРОГРАММА ВЫЧИСЛЕНИЯ КОРНЕЙ
C       КВАДРАТНОГО УРАВНЕНИЯ
        READ 1, A, B, C
        D = B ** 2 — 4 * A * C
        IF (D) 30,20,10
C  ВЫЧИСЛЕНИЕ ВЕЩЕСТВЕННЫХ КОРНЕЙ
   10   X1 = (— B + SQRT (D))/2/A
        X2 = (— B — SQRT (D))/2/A
   15   N = 0
        PRINT 2, X1, X2, N
        STOP
C  ВЫЧИСЛЕНИЕ РАВНЫХ ВЕЩЕСТВЕННЫХ КОРНЕЙ
   20   X1 = — B/2/A
        X2 = X1
        GO TO 15
C  ВЫЧИСЛЕНИЕ КОМПЛЕКСНЫХ КОРНЕЙ
   30   R = — B/2/A
        S = SQRT (— D)/2/A
        N = 1
        PRINT 2, R, S, N
        STOP
    1   FORMAT (. . .)
    2   FORMAT (. . .)
        END
```

Условный логический оператор имеет вид

IF (*b*) *s*

где *b* — логическое выражение; *s* — любой оператор, кроме операторов IF и DO.

При выполнении этого оператора сначала вычисляется, а затем анализируется значение логического выражения *b*. Если оно истинно (.TPUE.), внутренний оператор *s* выполняется, а если ложно (.FALSE.) — не выполняется. Схема действия оператора показана на рис. 4.10. В том случае, когда *b* — отношение, на выходах блока вместо .TRUE. пишут «да», а вместо .FALSE.— «нет».

Пример 2. Составить фрагмент программы, преобразующей массив $A(20) = a_1, a_2, \ldots, a_{20}$ так, чтобы каждый его элемент, превышающий величину q, уменьшался

[1] Запись FORMAT (...) условная, в дальнейшем она будет приводиться с указанием формата чисел.

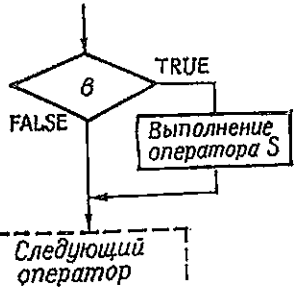


Рис. 4.10. Схема, иллюстрирующая выполнение условного логического оператора

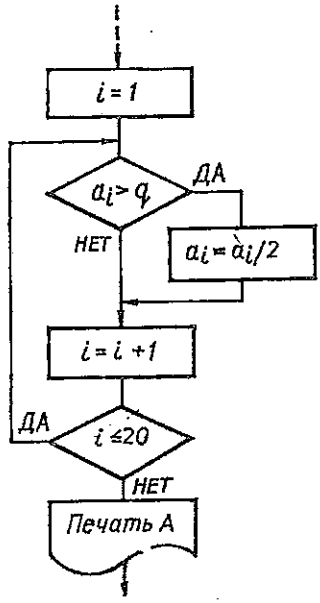


Рис. 4.11. Схема к примеру 2

вдвое. Схема алгоритма показана на рис. 4.11. Ему соответствует следующая программа:

```
      ...
      I = 1
   12 IF (A (I).GT.Q) A (I) = A (I)/2
      I = I + 1
      IF (I.LE.20) GO IO 12
      PRINT 40,A
      ...
```

Часто при истинности логического выражения необходимо выполнить одну группу операторов, а при ложности — другую. В этом случае фрагмент программы составляют по схеме алгоритма, изображенной на рис. 4.12.

Пример 3. Составить программу для вычисления корней квадратного уравнения $ax^2 + bx + c = 0$, используя логический оператор IF. Схема алгоритма показана на рис. 4.13.

При выполнении блока *2* определяется не только дискриминант D, но и переменная $r = -b/(2a)$, которая является одним из слагаемых при вычислении вещественных корней (см. формулы (4.1), (4.2)), а также вещественной частью комплексных корней (см. формулы (4.4), (4.5)). Разветвление процесса осуществляет блок *3*. Если $D < 0$, вычисляется переменная s, а затем печатаются вещественная (r) и мнимая (s) составляющие сопряженных комплексных корней. При $D \geqslant 0$ сначала определяется значение вспомогательной переменной $s = \sqrt{D}/(2a)$, а затем — значения корней x_1 и x_2 (блок *4*). Результаты вычисления корней и признак их вещественности ($N = 0$) печатаются (блок *5*). Данному алгоритму соответствует такая программа:

```
C ВЫЧИСЛЕНИЕ КОРНЕЙ КВАДРАТНОГО УРАВНЕНИЯ
      READ 1,A,B,C
      D = B*B — 4*A*C
      R = —B/2/A
      IF (D.LT.0) GO TO 10
      S = SQRT (D)/2/A
      X1 = R + S
      X2 = R — S
      N = 0
C ПЕЧАТЬ ВЕЩЕСТВЕННЫХ КОРНЕЙ
      PRINT 2, X1, X2, N
      STOP
   10 S = SQRT (—D)/2/A
      N = 1
```

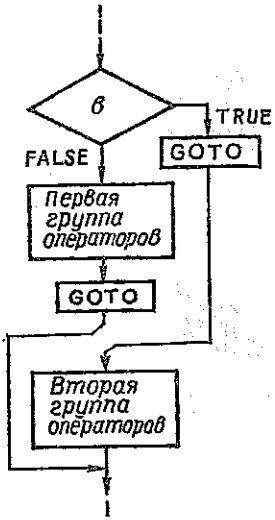


Рис. 4.12. Схема алгоритма выполнения одной из двух групп операторов в зависимости от значения логического выражения *b*

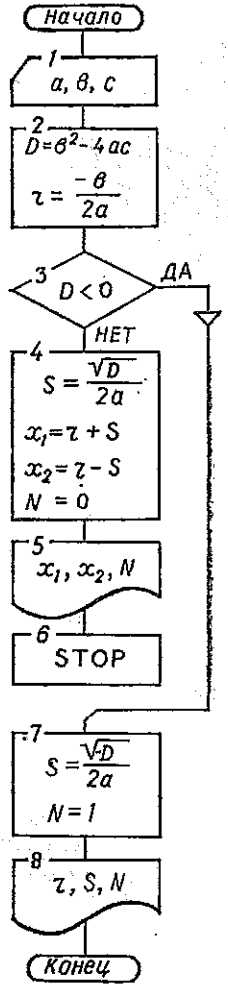


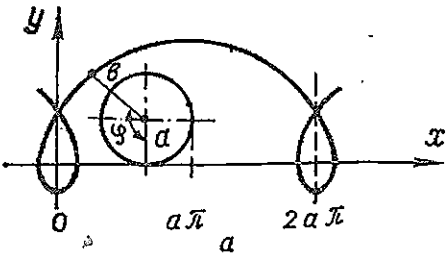
Рис. 4.13. Схема алгоритма вычисления корней квадратного уравнения

```
C ПЕЧАТЬ ВЕЩЕСТВЕННОЙ И МНИМОЙ
C СОСТАВЛЯЮЩИХ СОПРЯЖЕННЫХ
C КОМПЛЕКСНЫХ КОРНЕЙ
      PRINT 2, R, S, N
   2∂ STOP
    1 FORMAT (...)
    2 FORMAT (...)
      END
```

Пример 4. Составить ФОРТРАН-программу для вычисления текущих координат циклоиды (рис. 4.14, *а*) по формулам $x = a\varphi - b \sin \varphi$, $y = a - b \cos \varphi$. Отпечатать координаты двадцати точек при изменении φ от 0 до 2π. Схема алгоритма изображена на рис. 4.14, *б*.

С помощью блоков *2* и *3* вычисляется шаг Δφ изменения угла φ и задается его начальное значение (φ = 0). Блок *6* проверяет логическое условие. При $\varphi \geqslant 2\pi$ происходит останов. Если вычисления не завершены, то переменной φ присваивается новое значение, которое больше предыдущего на величину шага Δφ. После этого вычисляются (блок *4*) и печатаются (блок *5*) текущие значения координат циклоиды *x* и *y*. Описанный алгоритм реализуется следующей программой:

```
      READ 1∂, A, B
      DFI = 2 * 3.141593/19
      FI = ∂.
    5 X = A * FI — B * SIN (FI)
      Y = A — B * COS (FI)
      PRINT 2∂, X, Y
      IF (FI .GE. 6.2832) STOP
      FI = FI + DFI
      GO TO 5
   1∂ FORMAT (. . .)
   2∂ FORMAT (. . .)
      END
```



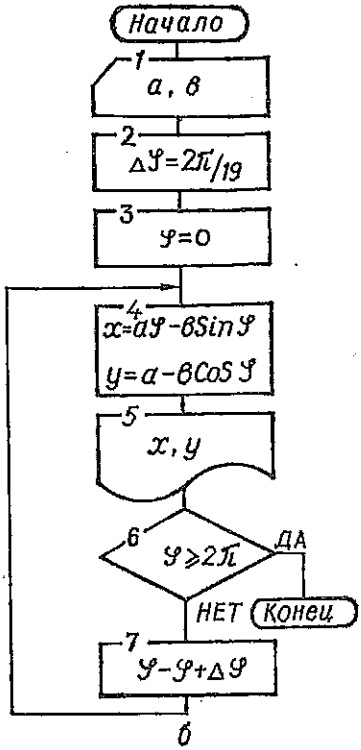


Рис. 4.14. Вычисление текущих координат циклоиды:

а — графическое изображение циклоиды при $b > a$; *б* — схема алгоритма

Пример 5. Составить программу для вычисления функции $y = e^x$. Она как стандартная имеется в библиотеке ФОРТРАНа (см. табл. 4.2), однако составление такой программы наглядно иллюстрирует использование условного оператора для организации циклических вычислений.

Значение e^x можно найти как сумму бесконечного ряда:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

В реально реализуемых алгоритмах вычисления прекращают, когда очередное слагаемое по абсолютному значению становится меньше некоторой наперед заданной малой величины ε, например меньше 10^{-7}.

Схема алгоритма вычисления функции $y = e^x$ показана на рис. 4.15. Очередное слагаемое, начиная со второго, образуется из предыдущего (блок *3*), причем начальное значение присваивается переменной s с помощью блока *2*. В ячейке памяти, отведенной для переменной y, накапливается сумма ряда (блок *4*). Когда очередное слагаемое s по абсолютному значению становится меньше величины ε (такой контроль осуществляет блок *5*), переменная y как результат вычисления выводится на печать. Если же заданная точность не достигнута, выполняется блок *6*. При этом переменная i (множитель, образующий факториал $k!$ в знаменателе очередного слагаемого) увеличивается на единицу, после чего осуществляется переход к блоку *3* для выполнения очередного цикла.

Программа вычисления значения e^x с точностью $\varepsilon = 10^{-7}$ записывается так:

```
C ВЫЧИСЛЕНИЕ Y = E**X
    READ 10,X
    Y = 1.
    I = 1
    S = 1.
  3 S = S*X/I
    Y = Y + S
    IF (ABS (S).LT.1E — 7) GO TO 7
    I = I + 1
    GO TO 3
  7 PRINT 20,X,Y
    STOP
```

```
10 FORMAT (...)
20 FORMAT (...)
   END
```

4.10. Оператор цикла

Оператор цикла (или оператор DO) используется для организации циклических вычислений, т. е. для выполнения некоторой части программы заданное число раз. Записывается оператор цикла следующим образом:

$$\text{DO } m \; i = n_1, \; n_2, \; n_3$$

где DO — ключевое слово ФОРТРАНа; m — метка последнего (закрывающего) оператора повторяющейся части программы; i — имя переменной целого типа (управляющая переменная); n_1, n_2, n_3 — управляющие параметры, задающие соответственно начальное, конечное значения управляющей переменной и шаг ее изменения.

Управляющим параметром может быть целое число (без знака) или переменная целого типа. Значения n_1, n_2, n_3 должны быть положительными и неизменными при выполнении цикла.

Операторы, помещаемые в программе после оператора DO (включая и закрывающий с меткой m), образуют повторяющуюся часть программы, называемую *телом оператора цикла* или *областью* DO. Последним в области DO может быть любой оператор, кроме тех, которые вызывают останов или передачу управления (STOP, GO TO, RETURN, арифметический IF, логический IF с одним из перечисленных). Если алгоритм требует, чтобы последним был один из запрещенных операторов, то цикл завершают оператором

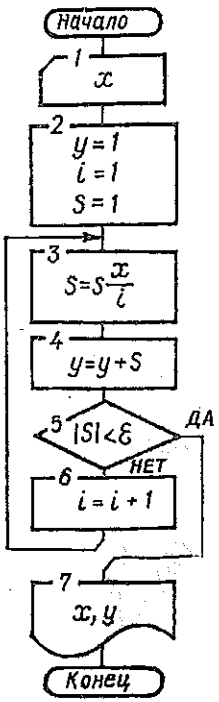
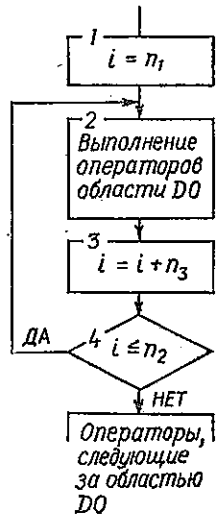


Рис. 4.15. Схема алгоритма для вычисления $y = e^x$

Рис. 4.16. Схема, иллюстрирующая выполнение оператора цикла



продолжения

m CONTINUE

Он не вызывает никаких действий и служит лишь для помещения метки.

Оператор цикла выполняется следующим образом (рис. 4.16). Переменной i присваивается значение n_1 (блок *1*) и выполняются операторы области DO (включая закрывающий оператор с меткой m). Затем к управляющей переменной добавляется шаг n_3 (блок *3*), и новое значение i сравнивается с конечным значением n_2 (блок *4*). Если $i \leqslant n_2$, снова выполняются операторы области DO, и так до тех пор, пока переменная i не превысит значения n_2. После этого управление передается оператору, следующему за помеченным меткой m (выход из области DO). Очевидно, что при $n_2 \leqslant n_1$ операторы области DO выполняются один раз.

Вход в область DO возможен только через оператор DO, а выход — или при завершении цикла, или с помощью операторов GO TO, IF и др. В последнем случае управляющая переменная i сохраняет значение, присвоенное ей перед входом в область DO.

Внутри области DO могут находиться другие операторы цикла, при этом их области действия должны целиком располагаться в области действия внешнего оператора DO, как показано на рис. 4.17. Такая конструкция носит название гнезда DO или вложенных циклов. В области действия внутреннего оператора цикла также могут быть операторы DO.

Рассмотрим использование операторов цикла на примерах.

Пример 1. Для вычисления вектора C $(c_1, c_2, \ldots, c_{16})$, равного сумме векторов A и B, оператор цикла записывается так:

```
...
   DO 10 I = 1,16
10    C (I) = A (I) + B (I)
...
```

Пример 2. Фрагмент программы для вычисления следа матрицы Q (25, 25) по формуле $S = \sum_{i=1}^{25} Q_{i,i}$

```
...
   S = 0
   DO 15 I = 1,25
15    S = S + Q (I,I)
...
```

Оператором $S = 0$ присваивается начальное значение (нуль) переменной S, которая используется для накапливания суммы элементов, стоящих на главной диагонали квадратной матрицы Q.

Пример 3. Вычисление факториала *N !*

```
...
   NF = 1
   DO 5 I = 1,N
5  NF = NF*I
...
```

Пример 4. Составить фрагмент программы для определения минимального числа (n) элементов массива A $(a_1, a_2, \ldots, a_{100})$, сумма которых, начиная с первого элемента, превышает заданное число R; при $\sum_{i=1}^{100} a_i < R$ присвоить $n = 101$.

Фрагмент программы можно записать следующим образом:

```
      DIMENSION A (100)
      ...
      S = 0.
      DO 7 I = 1,100
          S = S + A (I)
          IF (S.GT.R) GO TO 8
7         CONTINUE
      N = 101
      GO TO 9
8     N = I
9     ...
```

Сумма накапливается в ячейке памяти, отведенной для переменной S. Очередной элемент массива добавляется оператором присваивания, находящимся в области DO. Последним в ней должен быть оператор IF. Он осуществляет выход из цикла, как только сумма S становится больше величины R. Поскольку такое завершение области DO недопустимо, последним записывают оператор CONTINUE. При выходе из цикла по оператору GO TO 8 переменной n присваивается текущее значение управляющей переменной I. Если же сумма элементов массива меньше числа R, циклические вычисления завершаются добавлением последнего (a_{100}) элемента массива A. После естественного выхода из цикла переменной N присваивается константа 101. Такое присваивание необходимо, поскольку по правилам языка ФОРТРАН значение параметра i по

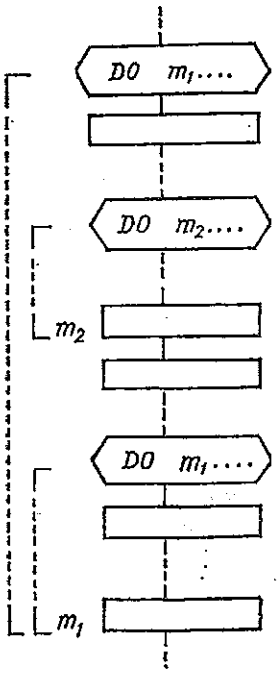


Рис. 4.17. Вложенные циклы

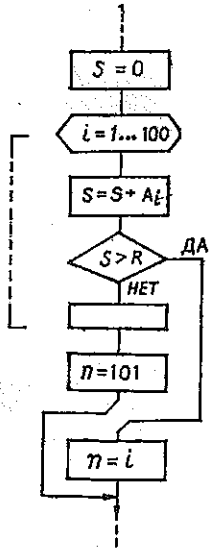


Рис. 4.18. Оператор цикла с областью DO, оканчивающейся оператором продолжения

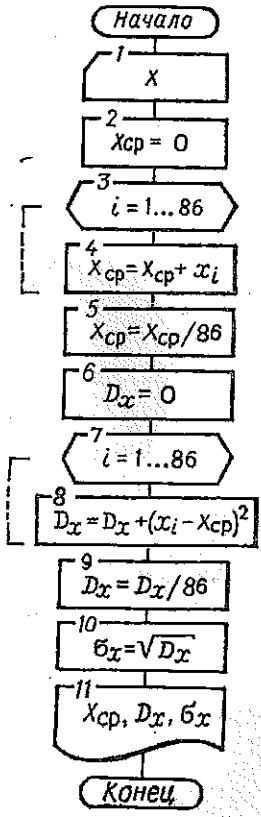


Рис. 4.19. Схема алгоритма для вычисления $X_{ср}$, D_x и σ_x

окончании выполнения оператора DO не определено[1]. Схема алгоритма показана на рис. 4.18.

Пример 5. Составить программу для вычисления среднего значения $X_{ср}$, дисперсии D_x и среднего квадратического отклонения σ_x для 86 элементов массива X по формулам:

$$X_{ср} = \frac{1}{n}\sum_{i=1}^{n} x_i; \quad D_x = \frac{1}{n}\sum_{i=1}^{n}(x_i - X_{ср})^2; \quad \sigma_x = \sqrt{D_x}.$$

Схема алгоритма изображена на рис. 4.19. После выхода из первого цикла (блоки *3* и *4*) переменная $X_{ср}$ содержит сумму всех элементов массива X. Только после деления на 86 (блок *5*) значение этой переменной станет равным $X_{ср}$. Аналогично вычисляется дисперсия (блоки *6*—*9*). Данной схеме соответствует программа:

```
C  СТАТИСТИЧЕСКИЕ РАСЧЕТЫ
        DIMENSION X (86)
        READ 5, X
C ВЫЧИСЛЕНИЕ СРЕДНЕГО
        XCP = 0.
        DO 10 I = 1, 86
  10        XCP = XCP + X (I)
        XCP = XCP/86.
C ВЫЧИСЛЕНИЕ ДИСПЕРСИИ
        DX = 0.
        DO 20 I = 1,86
  20        DX = DX + (X (I) — XCP) ** 2
        DX = DX/86.
C ВЫЧИСЛЕНИЕ СРЕДНЕГО КВАДРАТИ-
C       ЧЕСКОГО ОТКЛОНЕНИЯ
        SGX = SQRT (DX)
C ПЕЧАТЬ ОТВЕТОВ
        PRINT 7, XCP, DX, SGX
      5 FORMAT (. . .)
      7 FORMAT (. . .)
        STOP
        END
```

В операторах DO после числа 86 точка не ставится, поскольку это целая константа. В операторах присваивания после числа 86 десятичная точка поставлена, так как здесь должна быть вещественная константа. Если точки нет, необходимое преобразование выполнит транслятор.

Пример 6. Составить программу для вычисления суммы квадратов всех элементов матрицы A размером 10 × 20 по формуле

$$S_{кв} = \sum_{i=1}^{10}\sum_{j=1}^{20} A_{i,j}^2.$$

Схема алгоритма показана на рис. 4.20. Ей соответствует такая программа:

```
C    СУММА ЭЛЕМЕНТОВ МАТРИЦЫ
       DIMENSION A (10, 20)
       READ 1, A
       SK = 0.
       DO 5 I = 1, 10
            DO 5 J = 1, 20
   5           SK = SK + A (I, J) ** 2
       PRINT 2, SK
     1 FORMAT (. . .)
     2 FORMAT (. . .)
       END
```

[1] Некоторые трансляторы составляют такие машинные программы, в которых после естественного выхода из цикла значение параметра i равно $n_2 + 1$.

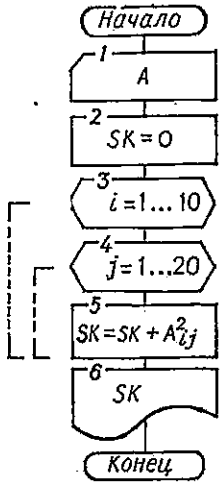


Рис. 4.20. Схема к примеру 6 для вычисления суммы квадратов элементов матрицы A

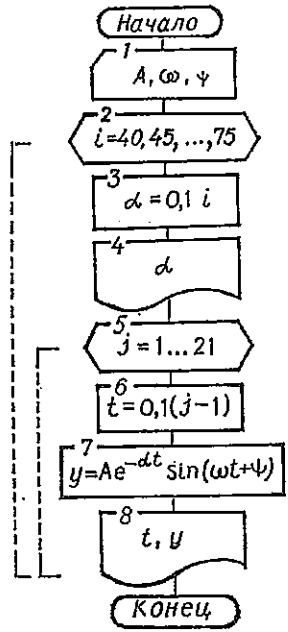


Рис. 4.21. Схема алгоритма к примеру 7

После ввода матрицы и присваивания начального значения переменной SK (блоки *1*, *2*) организован цикл с управляющей переменной i. В области DO находится еще один оператор цикла, в котором изменяется управляющая переменная j. Сначала при $i = 1$ параметр j принимает значения 1, 2, 3, ..., 20, и каждый раз выполняется оператор *5*, т. е. осуществляется суммирование:

$$\sum_{j=1}^{20} A_{1,j}^2 = A_{1.1}^2 + A_{1.2}^2 + A_{1.3}^2 + \cdots + A_{1.20}^2.$$

Затем переменная i принимает значение 2, и к накопленной сумме добавляется $A_{2.1}^2 + A_{2.2}^2 + A_{2.3}^2 + \ldots + A_{2.20}^2$. И так до тех пор, пока i не примет значение 10. После очередного добавления единицы управляющая переменная становится больше n_2. При этом выполнение внешнего (а значит, и внутреннего) оператора цикла оканчивается (см. рис. 4.20).

Пример 7. Составить программу для вычисления и печати значений функции $y = Ae^{-\alpha t} \sin(\omega t + \psi)$ при $t = 0; 0{,}1; 0{,}2; \ldots; 2$ и $\alpha = 4; 4{,}5; 5; \ldots; 7{,}5$.

Поскольку t и α — числа вещественные, а управляющая переменная целого типа, необходимо задать ей такие значения, чтобы затем, умножая или деля их на вещественную константу, получить требуемый ряд чисел. Например, можно принять $i = 8, 9, 10, \ldots, 15$ и вычислить $\alpha = 0{,}5i$. Аналогично образуется переменная t, однако, поскольку управляющая переменная должна быть больше нуля, следует осуществить смещение. Например, $j = 1, 2, 3, \ldots, 21$, $t = 0{,}1\ (j - 1)$. ФОРТРАН-программа с вложенными циклами записывается так:

```
      READ 1, A, W, PSI
      DO 10 I = 40, 75, 5
         AL = 0.1 * I
         PRINT 2, AL
         DO 10 J = 1,21
            T = 0.1 * (J - 1)
            Y = A * EXP (- AL * T) *
     *          SIN (W * T + PSI)
   10    PRINT 2, T, Y
```

```
1  FORMAT (. . .)
2  FORMAT (. . .)
   STOP
   END
```

Схема, соответствующая этой программе, показана на рис. 4.21.

4.11. Ввод и вывод данных

Обмен данными между оперативной памятью ЭВМ и внешними устройствами осуществляется с помощью операторов ввода (READ) и вывода (WRITE, PRINT). Каждый из них содержит ссылку на объявление формата. В объявлении указывается, в каком виде представлены данные на перфокартах (или другом носителе) при вводе или в каком виде они должны быть записаны на внешнем носителе (в частности, на бумажной ленте) при выводе.

Операторы ввода и вывода. Для *ввода* информации с внешнего носителя (например, с перфокарты) в оперативную память ЭВМ используется оператор

READ $(k, m)c$

где k — целое число без знака, идентифицирующее устройство ввода; m — метка объявления формата; c — список данных, подлежащих вводу (список ввода).

Для совместимости программ, написанных на разных версиях языка ФОРТРАН, допустима запись

READ m, c

Этот оператор осуществляет чтение информации с системного (при стандартном назначении с перфокарточного) устройства ввода.

Вывод информации из оперативной памяти и запись ее на внешний носитель производится с помощью оператора

WRITE $(k, m)c$

где c — список данных, подлежащих выводу (список вывода).

Допустима также запись

PRINT m, c

для печати результатов на бумажной ленте (вывод на системное устройство печати).

При стандартном назначении приведенным целым числам k соответствуют следующие внешние устройства: $k = 5$ — ввода с перфокарт, $k = 6$ — вывода на АЦПУ[1], $k = 7$ — вывода на перфокарты.

В список ввода-вывода могут входить имя переменной (элемент списка), элемент массива (элемент списка), имя массива (группа элементов списка), список с циклом [2] (группа элементов списка). Элементы в списке ввода-вывода отделяются друг от друга запятой.

Если список представлен именем массива, то вводятся все его элементы. В памяти ЭВМ элементы массива располагаются в последовательности, указанной выше. Очевидно, что в такой же последо-

[1] АЦПУ — алфавитно-цифровое печатающее устройство.

[2] Список с циклом часто называют неявным DO.

вательности вводимые числа должны быть отперфорированы и на картах.

Список с циклом используют для передачи части массива, а иногда и всего массива. При этом элементы массива могут передаваться в последовательности, отличающейся от расположения их в памяти ЭВМ (например, вывод матрицы по строкам).

Список с циклом имеет вид

$$(b,\ i = n_1,\ n_2,\ n_3)$$

где b — список ввода-вывода; i, n_1, n_2, n_3 — управляющая переменная и управляющие параметры.

Список b, в свою очередь, может содержать другой список с циклом. Такими конструкциями пользуются для передачи многомерных массивов.

Значения управляющих параметров n_1, n_2, n_3 могут быть введены оператором READ, в списке которого содержатся элементы неявного DO, причем ввод управляющих параметров должен предшествовать вводу массива.

Ввод или вывод элементов списка с циклом выполняется в такой последовательности. Управляющей переменной i присваивается значение n_1 и читаются (записываются) все элементы списка. Затем к переменной i добавляется шаг n_3. Если $i \leqslant n_2$, то ввод (вывод) всех элементов списка неявного DO повторяется (с новым значением i), и так до тех пор, пока выполняется условие $i \leqslant n_2$.

Рассмотрим примеры записи операторов ввода и вывода.

Пример 1. Оператор ввода со списком без неявного DO:

```
      DIMENSION A (10),S (3,5)
      READ (5,3) N,R,A (4),S
    3 FORMAT (...)
      .
      .
      .
```

С помощью оператора READ с перфокарт ($k = 5$) вводится 18 чисел: целое число N, вещественное число R, четвертый элемент массива A и 15 чисел двумерного массива S. Элементы последнего должны быть отперфорированы по столбцам: $S_{1,1}$, $S_{2,1}$, $S_{3,1}$, $S_{1,2}$, $S_{2,2}$, ..., $S_{1,5}$, $S_{2,5}$, $S_{3,5}$. Форму представления чисел и количество их на каждой карте указывают в спецификации формата.

Пример 2. Вывод группы элементов при помощи списка с циклом:

```
      DIMENSION X (100)
      .
      .
      .
      WRITE (6,10) (X (I),I = 1,15,2)
   10 FORMAT (...)
```

На АЦПУ ($k = 6$) выводятся элементы массива x_1, x_3, x_5, ..., x_{15} в виде, указанном объявлением формата с меткой 10.

Пример 3. Ввод двух одномерных массивов переменной длины с предварительным вводом управляющего параметра n_3. Вводимые элементы представлены именем переменной и двумя списками с циклами:

```
      DIMENSION X (100), Y (100)
      READ (5,8) N, (X (I), I = 1, N), (Y (I), I = 1, N)
    8 FORMAT (. . .)
      .
      .
```

Сначала вводится число N, затем N чисел массива X ($x_1, x_2, \ldots, x_N$), после чего N чисел массива Y ($y_1, y_2, \ldots, y_N$).

Если оператор ввода записать в виде

READ (5,8) N, (X (I), Y (I), I = 1, N)

то числовые данные массивов X и Y будут вводиться поочередно. Они должны быть отперфорированы на картах в такой последовательности: $x_1, y_1, x_2, y_2, x_3, y_3, \ldots, x_N, y_N$. В памяти ЭВМ массивы в обоих случаях располагаются одинаково: сначала элементы массива X, а затем элементы массива Y.

Пример 4. Вывод двумерного массива S (8,5) по строкам с использованием вложенных списков с циклами:

```
...
WRITE (6, 12) ((S (I, J), J = 1,5), I = 1,8)
12 FORMAT (...)
...
```

При использовании этого оператора переменная I принимает вначале значение 1 и выполняется вложенный список с циклом, т. е. выводятся элементы $S_{1,1}, S_{1,2}, \ldots, S_{1,5}$. Затем переменная I принимает значение 2 и выводятся элементы $S_{2,1}, S_{2,2}, \ldots, S_{2,5}$. Так повторяется до тех пор, пока не отпечатаются последние пять элементов массива ($S_{8,1}, S_{8,2}, \ldots, S_{8,5}$) при I = 8.

Объявления формата. Если операторы READ и WRITE обращаются к устройствам ввода с перфокарт ($k = 5$) или вывода на АЦПУ ($k = 6$), то связанные с ними объявления формата указывают:

при вводе — с каких колонок перфокарты должны считываться данные и форму их представления (числа целые или вещественные);

при выводе — в каком виде данные должны быть отпечатаны на бумажной ленте (числа целые или вещественные, количество знаков в дробной части, чисел в строке и т. д.).

Объявление формата записывается как

m FORMAT *сф*

Спецификация формата *сф* имеет вид

$$(// \ldots / on_1 \# on_2 \# on_3 \# \cdots \# on_{n-1} \# on_n / \ldots //)$$

где on_i — описатель поля (формат поля) или группа описателей полей; # — разделитель полей.

Наиболее часто применяют следующие описатели полей:

$$r\text{I}w,\ r\text{F}w.d,\ r\text{E}w.d,\ w\text{Н}литерал',\ 'литерал',\ w\text{X}$$

Здесь I, F, E, H, X обозначают коды преобразования; r — счетчик повторений, указывающий, сколько раз повторяется следующий за ним основной описатель поля [1]; w — ширина поля, т. е. количество позиций, в которых размещается число (включая его знак и десятичную точку), литерал или пробелы; d — количество позиций, отведенных в поле для дробной части числа.

Группа описателей полей — это несколько описателей (в частности, один), заключенных в круглые скобки. Перед скобками может стоять счетчик повторений.

Разделителями полей являются запятая или серия дробных черт (в частности, одна). Дробная черта одновременно используется для

[1] Отсутствие в описателе полей сивола r означает, что $r = 1$.

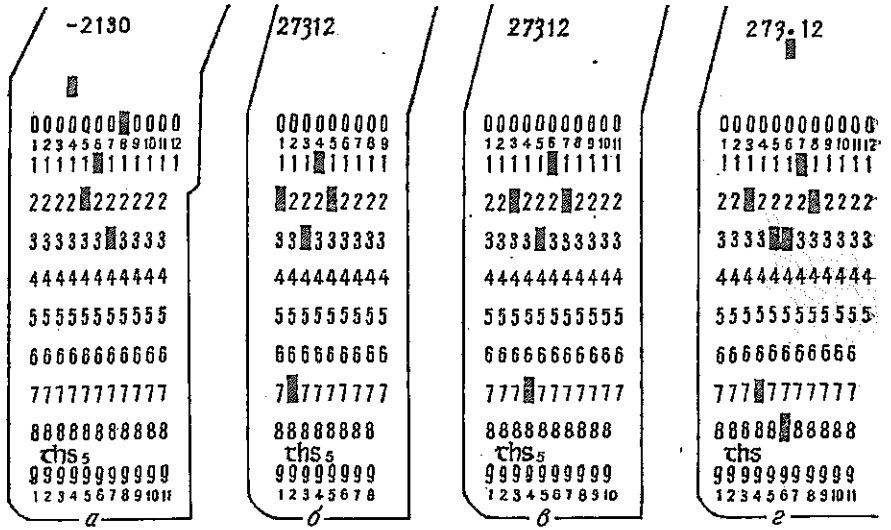


Рис. 4.22. Перфорация чисел:
а — — 2130 в формате I8; *б* — 273,12 в формате F5.2; *в* — 273,12 в формате F8.3; *г* — 273,12 в формате F1⌀.5 с отперфорированной точкой

разграничения форматных записей [1], поэтому две стоящие рядом дробные черты указывают на пустую запись (отсутствие символов в записи). Необходимо различать скобки группы описателей полей и скобки спецификации формата.

Приведем несколько примеров. Так, в объявлении

4 ␣ FORMAT (F1⌀.4, 6E16.5/)

спецификация формата (F1⌀.4, 6E16.5/) содержит два описателя поля, причем второй из них со счетчиком повторений $r = 6$.

В объявлении

52 ␣ FORMAT (3I6, F12.4/' ␣ A = ', F8.2,1⌀X, 'B = ', F8.2)

спецификация формата включает семь описателей полей, второй и третий разделены дробной чертой, остальные — запятыми. Описатель поля 3I6 означает I6, I6, I6.

Объявление формата

16 ␣ FORMAT (/5,4F1⌀. 3/2 (E14.4, 5F8.2))

содержит группу описателей полей 2 (E14.4, 5F8.2) со счетчиком повторений $r = 2$.

Рассмотрим использование описателей полей формата при вводе и выводе данных.

Описатель поля *r*I*w*. Он используется для передачи целых чисел. При *вводе* описатель I*w* указывает, что на карте отперфорировано целое число, занимающее *w* позиций. Это число должно располагаться

[1] Примерами форматных записей являются перфокарта или строка, выводимая на печать.

Таблица 4.5. Примеры перфорации чисел для ввода по спецификациям формата F*w.d* и E*w.d*

Вводимое число	Описатель поля	Колонки перфокарты												Дробную часть определяет
		1	2	3	4	5	6	7	8	9	10	11	12	
273,12	F5.2	2	7	3	1	2								*d*
	F8.3			2	7	3	1	2						*d*
	То же	2	7	3	.	1	2							Точка
	»		2	7	3	.	1	2						То же
—273,12	»		—	2	7	3	.	1	2					»
+273,12	F1β.5			2	7	3	1	2						*d*
	или													
	E1β.5													
	То же		2	7	3	1	2	E	+	β	3			*d*
	»	2	7	3	.	1	2							Точка
	»	β	.	2	7	3	1	2	E	β	3			То же
—273,12	»		—	2	7	3	.	1	2					»
$-4{,}5\cdot10^{7}$	»	—	4	5	0	0	0	0	0	0	.			»
	»		—	4	5					E	7			*d*
	»						4	.	5	E	7			Точка
10^{-8}	»						1	.	E	—	8			То же
	»							1	.	—	8			»

в правой части отведенного поля, поскольку пробелы интерпретируются как нули. Например, при вводе числа N = —2130 с использованием предложений

```
READ (5,6) N
6 FORMAT (I8)
```

оно должно быть отперфорировано так, как показано на рис. 4.22, *а*[1].

При *выводе* на АЦПУ целое число печатается в правой части поля шириной *w*, а знак «минус» — непосредственно перед числом. Например, если на печать выводится число N = —3705, причем с переменной N в списке вывода связан описатель поля I 1β в спецификации формата, то на бумажной ленте это число отпечатается следующим образом:

␣␣␣␣␣—3705␣␣␣

Описатель числового поля *r*F*w.d*. Его применяют для передачи чисел в форме F. При *вводе* этот описатель указывает, что на карте отперфорировано вещественное число в форме F, занимающее *w* позиций, причем для дробной части отводится *d* позиций в правой части поля. Например, для ввода числа A = 273,12 можно использовать описатель F 5.2 и отперфорировать число, как показано на рис. 4.22, *б*.

Количество позиций в отводимом поле может быть больше, чем необходимо для записи числа. Так, число 273,12 можно ввести, используя описатель поля F 8.3. Число в данном случае необходимо отперфорировать так, как показано на рис. 4.22, *в*.

Наличие на карте десятичной точки отменяет положение точки, описанное указателем дробной части *d*. На рис. 4.22, *г* показан один

[1] Для вводимой информации можно использовать все 80 колонок перфокарты, а не 72, как для строки ФОРТРАН-программы.

Таблица 4.6. Примеры вывода вещественных чисел по спецификациям формата F*w.d* и E*w.d*

Выводимое на АЦПУ число	Описатель поля	Позиция в строке бумажной ленты												Примечания
		1	2	3	4	5	6	7	8	9	10	11	12	
529,368	F1⌀.3				5	2	9	.	3	6	8			—
	F8.2			5	2	9	.	3	7					—
	F12.5				5	2	9	.	3	6	8	⌀	⌀	—
	F8.5	*	*	*	*	*	*	*	*					Число не размещается в отведенном поле
—0,9256	F8.5	—	⌀	.	9	2	5	6	⌀					—
529,368	E12.6	⌀	.	5	2	9	3	6	8	E		⌀	3	—
	E12.4			⌀	.	5	2	9	4	E		⌀	3	—
	E9.4	.	5	2	9	4	E		⌀	3				Для нуля перед десятичной точкой недостает одной позиции
$0{,}384 \cdot 10^{-6}$	E10.4	⌀	.	3	8	4	⌀	E	—	⌀	6			—
$-0{,}7624 \times \times 10^{-4}$	E8.4	*	*	*	*	*	*	*	*					Число не размещается в отведенном поле
	E12.4		—	⌀	.	7	6	2	4	E	—	⌀	4	—

из вариантов перфорации числа 273,12 в формате F 1⌀.5. Другие варианты перфорации чисел приведены в табл. 4.5.

При *выводе* числа в соответствии с описателем поля F*w.d* оно печатается в правой части отведенного поля шириной *w*, причем для дробной части отводится *d* позиций. Знак «минус» печатается непосредственно перед числом. Положительные числа печатаются без знака. Если число не размещается в отведенном поле, печатаются звездочки (*). Для примера в табл. 4.6 показаны некоторые варианты вывода чисел по формату F*w.d*.

Описатель числового поля *r*E*w.d*. Он используется для передачи вещественных чисел в форме E. При *вводе* этот описатель указывает, что на карте отперфорировано число в виде мантиссы и порядка, занимающее *w* позиций в правой части поля. В форме E обычно вводят очень большие или очень малые числа. Для размещения порядка резервируется четыре позиции, примыкающие к правой границе поля. Можно, однако, сократить запись порядка, опустив знак «плюс» (если порядок положителен) или букву E (в данном случае знак порядка обязателен). Однозначный порядок можно записать в одной позиции. Следовательно, равнозначны и допустимы записи порядка E + ⌀3, E⌀3, +⌀3, +3.

Мантиссу записывают перед порядком в виде числа с целой и дробной частями. Положение десятичной точки определяется или указателем *d*, или записью и перфорацией этой точки (действие указателя *d* при этом отменяется).

Необходимо отметить, что при вводе чисел действие описателей полей F*w.d* и E*w.d* одинаково, а форма представления числа (F или

E) определяется по тому, как оно отперфорировано — с порядком или без него (см. табл. 4.5).

При *выводе* в формате E$w.d$ число печатается в правой части отведенного поля шириной w таким образом: знак числа (для отрицательного), нуль, десятичная точка, d значащих цифр мантиссы, символ E, знак порядка (вместо знака «плюс» оставляется пробел), две цифры порядка. Следовательно, минимальный размер поля $w = d + 6$[1]. Целесообразно, однако, отводить для числа больше позиций, чтобы между отпечатанными числами оставались пробелы. Примеры чисел, выведенных в формате E$w.d$, приведены в табл. 4.6.

Описатели текстового поля *w*Н*литерал* и *'литерал'*. Они применяются для передачи символов, в частности текста.

Описатель поля *w*Н*литерал* указывает, что при выводе (вводе) передается текстовая информация (литерал), следующая непосредственно за кодом преобразования Н и содержащая w символов, включая пробелы. В литерале можно использовать любые символы ДКОИ. Вместо описателя *w*Н*литерал* можно использовать равноценный описатель текстового поля *'литерал'*, не требующий указания количества символов. Например, предложения ФОРТРАН-программы

```
      WRITE (6,8)
    8 FORMAT (' ␣ МЕТОД ␣ НЕ ␣ ПРИМЕНИМ')
```

вызывают вывод литерала, записанного в спецификации формата (без кавычек), и на АЦПУ печатается

МЕТОД ␣ НЕ ␣ ПРИМЕНИМ

Тот же результат будет достигнут при использовании объявления формата

8 ␣ FORMAT (18Н ␣ МЕТОД ␣ НЕ ␣ ПРИМЕНИМ)

Описатель поля пробелов *w*X. Его используют для пропуска w символов на внешнем носителе. Заполнение пробелами определенных участков строки, например между числами, улучшает наглядность выводимых на АЦПУ данных.

Взаимодействие спецификации формата со списком ввода чисел. При выполнении оператора ввода устанавливается соответствие между описателями числовых полей в спецификации формата (на который ссылается оператор READ) и элементами в списке данных, подлежащих вводу. Первому описателю поля (считая слева направо) соответствует первый элемент в списке ввода, второму (с учетом счетчика повторений r) — второй и т. д.[2] При этом данные читаются с перфокарт в соответствии с указанным описателем поля и вводятся в память ЭВМ. Например, если на карте отперфорировано

Символ	1	*8*	*8*	2	7	5	9	5	1	3	7	4	8	2	4	—	1			
Колонка	1	2		4		6		8		10		12		14		16		18		20

[1] При недостатке места нуль перед десятичной точкой не печатается.

[2] Здесь рассматривается только передача данных с кодами преобразования I, F и E.

и фрагмент программы, осуществляющий ввод, записан в виде

```
      READ (5,40) K,R,D1,ALFA
40    FORMAT (I3,F4.2,F5.1,E5.3)
```

то в память, отведенную для переменных K, R, D1 и ALFA, записываются соответственно числа: 100; 27,59; 5137,4 и 0,0824. Целесообразнее, однако, использовать объявление формата

```
40    FORMAT (I5, 3F10.3)
```

и отперфорировать числа так:

Символ			1	0	0	2	7	.	5	9				5	1	3	1	.	4			0	.	0	8	2	4			
Колонка	1	2		4		6		8		10		12		14		16		18		20		22		24		26		28		30		32

Этим достигается большая наглядность и простота перфорации и проверки карты.

Количество элементов в списке ввода и количество основных описателей полей в спецификации формата могут быть неодинаковыми. При этом действует правило:

если описателей полей больше, чем элементов в списке ввода, лишние описатели игнорируются;

если описателей полей меньше, чем элементов в списке ввода, просмотр описателей (для указанного выше соответствия) продолжается с начала последней группы описателей полей (с учетом счетчика повторений), а при ее отсутствии — с начала спецификации формата. Это повторяется до тех пор, пока не будет исчерпан весь список ввода.

Описатели полей, разделенные запятыми, указывают на принадлежность вводимой информации к одной записи: данные читаются с одной перфокарты. Если в спецификации формата встречается дробная черта, означающая конец записи, то считывание данных продолжается с новой перфокарты (следующей по порядку в колоде). Переход к новой перфокарте происходит также при достижении закрывающей скобки спецификации формата (не скобки группы форматов!).

Пример 5. Необходимо ввести часть двумерного массива Q. Число строк M и столбцов N не постоянно, но не превышает 12. Каждый элемент массива — вещественное число, имеющее не более шести десятичных разрядов (целая и дробная части). Ввод массива может быть осуществлен при такой записи:

```
      DIMENSION Q (12, 12)
      READ (5, 16) M, N, ((Q (I, J), J = 1, N), I = 1, M)
16    FORMAT (2I2/(10F8.3))
```

На первой карте в формате *I2* перфорируют два целых числа: M и N. Отделяющая этот формат дробная черта означает конец записи, т. е. переход к новой перфокарте. На последующих картах в формате F8.3 перфорируют по десять чисел — элементов массива Q. Если вводимых чисел больше десяти, то по закрывающей скобке спецификации формата осуществляется переход к новой перфокарте, а просмотр спецификации формата продолжается с последней группы описателей полей: (10F8.3). Числа на картах перфорируют по строкам [1], причем на последней карте может быть менее десяти чисел. Оставшиеся описатели полей в этом случае игнорируются.

[1] Если список с циклом записать в виде ((Q (I, J), I = 1, M), J = 1, N) или ((Q (J, I), J = 1, M), I = 1, N), то элементы массива нужно перфорировать по столбцам.

Таблица 4.7 Управление продвижением бумаги

Управляющий символ	Интервал продвижения бумаги перед печатью	Примечания
Пробел	Одна строка	То же действие вызывает любой другой символ, кроме 0, 1 или +
0	Две строки	
1	До начала следующей страницы	Количество строк в странице определяется устройством печати
+	Отсутствует	Печать на той же строке

Допустим, необходимо ввести массив

$$Q = \begin{vmatrix} 3{,}7 & 12{,}5 & -2{,}9 & 0{,}83 \\ 5{,}2 & 0{,}7 & 16{,}1 & 4{,}4 \\ -1{,}1 & 6{,}2 & 8{,}4 & 3{,}7 \end{vmatrix}.$$

Для этого карты перфорируют следующим образом:

Номер перфокарты	Номер колонки									
	1	9	17	25	33	41	49	57	65	73 80
1	3.4									
2	3.7	12.5	—2.9	0.83	5.2	0.7	16.1	4.4	—1.1	6.2
3	8.4	3.7								

Взаимодействие спецификации формата со списком вывода данных[1]. При выполнении операторов WRITE и PRINT список вывода взаимодействует со спецификацией формата так же, как при вводе данных, однако некоторые особенности требуют пояснений.

Выводимые на АЦПУ данные сначала записываются в специальный буферный регистр[2], и только когда в спецификации формата встречается дробная черта (конец записи) или достигается закрывающая скобка спецификации, вся запись, сформированная в регистре, печатается на АЦПУ в виде одной строки.

Формирование новой записи в буферном регистре начинается в следующих случаях: при прохождении открывающей скобки спецификации формата; после дробной черты; при очередном возвращении к началу последней группы описателей полей; если буферный регистр заполнен до предела, а подлежащая выводу информация не исчерпана.

Первый символ записи, сформированной в буферном регистре, не печатается, а используется для управления продвижением бумаги (табл. 4.7).

Если при просмотре списка форматов встречаются описатели полей *w*X, *w*Н*литерал* или *'литерал'*, никакие элементы в списке вывода им в соответствие не ставятся, но при этом в буферный регистр записывается *w* пробелов (*w*X) или литерал (*w*Н*литерал*, *'литерал'*).

[1] Здесь рассматривается только вывод на АЦПУ.

[2] Буферный регистр рассчитан на запись 129 символов.

Пример 6. Составить программу для вычисления корней квадратного уравнения $ax^2 + bx + c = 0$. На печать выдавать информацию в виде

␣␣ИСХОДНЫЕ ДАННЫЕ

␣␣A = *число*␣␣␣␣B = *число*␣␣␣␣C = *число*

␣

␣␣ОТВЕТЫ

␣␣X1 = *число*␣␣␣␣␣X2 = *число*

или, если корни сопряженные комплексные, после строки со словом «ОТВЕТЫ» печатать

␣␣X1 = *число* + I *число*

␣␣X2 = *число* — I *число*

Для решения поставленной задачи можно использовать программу, приведенную в примере 3 подразд. 4.9, но внести в нее следующие изменения: исключить операторы N = 0 и N = 1, заменить операторы ввода и вывода, а также заполнить спецификацию формата.

Предложения программы, организующие ввод и вывод данных:

```
C ВВОД И ВЫВОД К ПРОГРАММЕ ВЫЧИСЛЕНИЯ
C КОРНЕЙ КВАДРАТНОГО УРАВНЕНИЯ
      READ (5,1) A,B,C
    1 FORMAT (3F10.3)
        WRITE (6,2) A,B,C
    2 FORMAT ('   ИСХОДНЫЕ ДАННЫЕ'/
     *'   A =',F8.3,'    B =',F8.3,
     *'    C =',F8.3//'   ОТВЕТЫ')
      D =
      ...
C ПЕЧАТЬ ВЕЩЕСТВЕННЫХ КОРНЕЙ
      WRITE (6,3) X1,X2
    3 FORMAT ('   X1 =',E12.5,6X,'X2 =',E12.5)
      ...
C ПЕЧАТЬ КОМПЛЕКСНЫХ КОРНЕЙ
      WRITE (6,4) R,S,R,S
    4 FORMAT ('   X1 =',E12.5,' + I',E12.5/
     *        '   X2 =',E12.5,' — I',E12.5)
      END
```

Оператор READ вводит исходные данные A, B, C, которые должны быть отперфорированы на одной карте. Каждой переменной отводится десять колонок. Оператор

WRITE (6.2) A,B,C

печатает введенные данные в соответствии со спецификацией формата: сначала печатается текст ␣␣ИСХОДНЫЕ␣ДАННЫЕ, затем осуществляется переход к новой записи (дробная черта). Печать строки с исходными данными (в заданном виде) производится по концу записи — первая дробная черта после описателя поля F 8.3. Поскольку между двумя дробными чертами нет описателей полей, бумага продвигается на одну чистую строку [1]. По концу следующей записи (закрывающая скобка спецификации формата) со сдвигом еще на одну строку (пробел в первой позиции записи) печатается слово ␣␣ОТВЕТЫ с двумя пробелами перед ним. По два пробела остается также перед словами ИСХОДНЫЕ ДАННЫЕ и A =, так как первый из трех пробелов, имеющихся в описателе поля, не выводится, а управляет продвижением бумаги (см. табл. 4.7).

Оператор WRITE (6,3) X1,X2 в формате E12.5 печатает в одну строку вещественные корни уравнения. Выбор такого формата обусловлен тем, что диапазон, в котором могут находиться значения корней, не известен.

Комплексные корни печатаются в две строки, так как объявлением формата с меткой 4 предусмотрен вывод двух записей.

[1] Количество пустых строк зависит от количества проставленных дробных черт.

4.12. Внутренняя функция

При решении некоторых задач возникает необходимость вычислений в разных местах программы по одной и той же формуле, но с различными аргументами. В языке ФОРТРАН предусмотрена возможность помещать такую формулу в начале программной единицы в виде *объявления внутренней функции* (см. рис. 4.2), а затем обращаться к ней по имени (см. подразд. 4.5).

Объявление внутренней функции записывается следующим образом:

$$f(a_1, a_2, \ldots, a_n) = s$$

где f — имя внутренней функции; a_1, a_2, ..., a_n — символические имена (в частности, одно), называемые формальными параметрами; s — арифметическое или логическое выражение.

Имя функции выбирают с учетом требований, сформулированных в подразд. 4.3. Тип результата определяется по первой букве имени (неявно) или явным объявлением (см. подразд. 4.4). В качестве формальных параметров используются имена простых переменных, которые не обязательно должны быть уникальными и могут совпадать с именами переменных, используемых в программе.

Выражение s может содержать кроме формальных параметров константы, имена переменных, используемых в данной программной единице, и указатели функций.

Примеры объявлений внутренних функций:

```
FC (X) = 2 * X * * 3 + R
ALIS (S, Q, X) = SQRT (S + 3.5 * Q * SIN (X)) + S * Q
```

На внутреннюю функцию ссылаются из арифметических (или логических) выражений с помощью указателя этой функции. Он состоит из имени функции, за которым в скобках следует список фактических параметров. В качестве последних можно использовать арифметические (или логические) выражения, в частности константы, переменные (простые или с индексами), указатели других функций. Фактические параметры должны быть согласованы с формальными. Это означает, что их количество, порядок следования и тип должны быть одинаковыми.

При ссылке на внутреннюю функцию выражение s вычисляется с подстановкой в него значений фактических параметров, после чего полученное значение передается в то место программы, откуда была произведена ссылка. Например, если в операторе присваивания встречается ссылка на внутреннюю функцию ALIS

```
D = Q + 2.3 * ALIS (EXP (R), R, W * T + PSI)
```

он выполняется как оператор

```
D = Q + 2.3 * SQRT (EXP (R) + 3.5 * R * SIN (W * T + PSI) + EXP (R) * R)
```

Пример 1. В разных местах программы присвоить переменным U и P значения

$$u = 2(a^2 + b)^3 + R,$$

$$p = \frac{2s^3 + R}{2(\operatorname{tg} x + 0{,}8)^3 + R} + s.$$

Воспользовавшись объявлением FC (X), можно записать:

```
U = FC (A * * 2 + B)
. . .
P = FC (S)/FC (TAN (X) + 0.8) + S
```

Когда при выполнении программы встретится указатель функции с именем FC, по нему будет найдено соответствующее объявление, и в формулу $(2x^3 + R)$ вместо формального параметра X будут подставлены фактические. При первом обращении к функции вместо X будет подставлено значение выражения $a^2 + b$, при втором s, а при третьем $\operatorname{tg} x + 0{,}8$. Переменная R не входит в число формальных параметров и, следовательно, является общей для всего программного модуля.

Пример 2. Составить ФОРТРАН-программу для решения дифференциального уравнения

$$\frac{dy}{dx} = f(x, y) = 1 + 0{,}2x \sin x - 0{,}8y^2 \qquad (4.6)$$

методом Рунге — Кутта с постоянным шагом h на интервале $[x_0; x_k]$ при заданных начальных условиях $x = x_0$, $y(x_0) = y_0$.

Классический метод Рунге — Кутта описывается системой соотношений

$$y_{m+1} = y_m + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \qquad (4.7)$$

где

$$k_1 = f(x_m, y_m), \qquad (4.8)$$

$$k_2 = f\left(x_m + \frac{h}{2},\ y_m + \frac{hk_1}{2}\right); \qquad (4.9)$$

$$k_3 = f\left(x_m + \frac{h}{2},\ y_m + \frac{hk_2}{2}\right); \qquad (4.10)$$

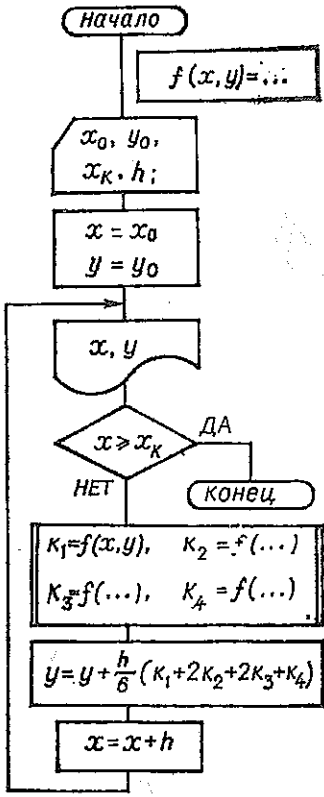


Рис. 4.23. Схема алгоритма интегрирования дифференциального уравнения (к примеру 2)

$$k_4 = f(x_m + h,\ y_m + hk_3); \qquad (4.11)$$

$$x_{m+1} = x_m + h. \qquad (4.12)$$

Каждая следующая точка решения (x_{m+1}, y_{m+1}) определяется по предыдущей (x_m, y_m), начиная от x_0, y_0. Для нахождения очередной ординаты y_{m+1} необходимо четыре раза вычислять значение функции $f(x, y)$ по одной и той же формуле (4.6), но с разными аргументами: (4.8) — (4.11). Целесообразность использования здесь внутренней функции очевидна.

Схема алгоритма показана на рис. 4.23, а соответствующая программа записывается так:

```
C ИНТЕГРИРОВАНИЕ ДИФФЕРЕНЦИАЛЬНЫХ
C УРАВНЕНИЙ МЕТОДОМ РУНГЕ — КУТТА
      REAL K1,K2,K3,K4
      F (X,Y) = 1 + 0.2*X*SIN (X) — 0.8*Y*Y
      READ (5,10) X0,Y0,XK,H
   10 FORMAT (4F10.4)
      X = X0
      Y = Y0
   20 WRITE (6,10) X,Y
      IF (X.GE.XK) STOP
      K1 = F (X,Y)
      K2 = F (X + H/2,Y + H*K1/2)
```

```
      K3 = F (X + H/2,Y + H*K2/2)
      K4 = F (X + H,Y + H*K3)
   С СЛЕДУЩИЕ ЗНАЧЕНИЯ X И Y
      Y = Y + H/6* (K1 + 2*K2 + 2*K3 + K4)
      X = X + H
      GO TO 20
      END
```

Первое предложение программы объявляет переменные k_1, k_2, k_3, k_4 вещественными. Объявление типа может отсутствовать, если имена переменных выбрать, например, такими: AK1, AK2, AK3, AK4. Второе предложение программы — объявление внутренней функции с именем F и формальными параметрами X и Y. Заменяя перфокарту с этим объявлением, можно интегрировать другие дифференциальные уравнения.

После ввода исходных данных и присваивания переменным X и Y начальных значений печатаются координаты первой точки (оператор с меткой 20). Если значение X меньше конечного (XK), то по формулам (4.8) — (4.11) вычисляются коэффициенты K1, ..., K4. При этом каждый раз происходит обращение к внутренней функции, поскольку в правой части операторов присваивания записаны указатели функции с именем F.

Переменной K1 присваивается значение F (X, Y). В данном случае имена фактических и формальных аргументов совпадают, что вполне допустимо [1]. При втором обращении к внутренней функции (для определения K2) фактическими параметрами являются арифметические выражения X + H/2 и Y + H*K1/2. И так далее. После определения коэффициентов K1, ..., K4 вычисляются координаты следующей точки по формулам (4.7) и (4.12). Новые значения присваиваются тем же переменным X и Y, поскольку предыдущие значения больше не нужны.

Оператор GO TO 20 передает управление на печать результатов, после чего, если X < XK, вычисляются координаты следующей точки, и так до тех пор, пока не отпечатается значение Y, соответствующее X = XK. При этом логическое выражение в операторе IF будет истинным и произойдет останов.

4.13. Внешние функции

Часто возникает необходимость вычислять функцию нескольких переменных не по одной формуле, а по алгоритму, который не реализуется одним оператором присваивания. В таких случаях используют не внутреннюю, а внешнюю функцию, т. е. самостоятельный программный модуль. В начале этого модуля должен быть помещен заголовок функции

$$t \text{ FUNCTION } f\ (a_1,\ a_2,\ \ldots,\ a_n)$$

где t — описатель типа; f — имя внешней функции, которое должно быть уникальным; a_1, a_2, ..., a_n — формальные параметры, которыми могут быть имена переменных, массивов и внешних процедур (обязательно наличие хотя бы одного параметра).

После заголовка следует тело модуля (см. рис. 4.2), в котором вычисляется значение функции. По завершении вычислений передача управления в вызывающий программный модуль осуществляется оператором возврата RETURN, который указывает на логический конец модуля-функции.

Ссылка на внешнюю функцию производится при помощи указателя функции. В арифметическом (или логическом) выражении, где

[1] Программа в остальном не изменится, если объявление внутренней функции записать в виде

$$\text{F (A, B)} = 1 + 0{,}2 * \text{A} * \text{SIN (A)} - 0{,}8 * \text{B} * \text{B}$$

в качестве операнда используют указатель функции, формальные параметры заменяют фактическими. Допустимые соответствия между формальными и фактическими параметрами в процедурах следующие:

Формальный параметр

Переменная (1)
Массив (2)
Имя внешней процедуры (3)

Фактический параметр

Выражение (в частности, константа, переменная, элемент массива, указатель функции) (1)
Массив, элемент массива (2)
Имя внешней процедуры (3)

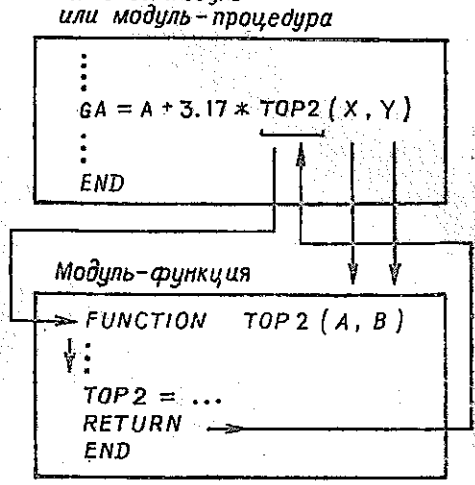


Рис. 4.24. Взаимодействие вызывающего программного модуля с внешней функцией

Взаимодействие между программным модулем со ссылкой на внешнюю функцию и соответствующим модулем-процедурой показано на рис. 4.24. Когда в вызывающем программном модуле встречается имя какой-либо внешней функции (в нашем примере TOP2), она отыскивается по указанному имени, и на место формальных параметров (A, B) передаются вычисленные значения фактических (X, Y). После этого выполняются операторы модуля-процедуры, а полученный результат присваивается имени функции. По достижении оператора RETURN вычисленное значение передается в программный модуль на место указателя функции, и выполнение оператора продолжается.

Внутри тела модуля-функции могут быть ссылки на другие внешние процедуры. Не допускается, однако, обращение функции к самой себе ни прямо, ни косвенно (через другие модули).

Пример 1. Написать оператор, присваивающий переменной значение, равное числу сочетаний из n элементов по k, т. е.

$$C_n^k = \frac{n!}{k!\,(n-k)!}.$$

Поскольку факториал необходимо вычислять трижды, целесообразно оформить для его вычисления внешнюю функцию:

```
C ГОЛОВНОЙ МОДУЛЬ
      INTEGER CNK
      . . .
      . . .
      CNK=NF (N)/NF (K)/NF (N — K)
      . . .
      . . .
      END
C
C ПРОЦЕДУРА-ФУНКЦИЯ
      FUNCTION NF (N)
      NF — 1
```

```
      DO 1 I = 1, N
    1    NF = NF * I
      RETURN
      END
```

Внешняя функция названа именем NF. Тип *f* не указан, так как имя неявно указывает на тип результата — целый. При выходе из оператора цикла переменная NF получает значение $n = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$.

В головном модуле трижды встречается ссылка на внешнюю функцию NF. При первой ссылке на место формального параметра N передается фактический, обозначенный тем же именем, при второй вычисляется значение *k*!, а при третьей вместо формального параметра N ставится значение фактического, равное N — K.

Если среди формальных параметров встречаются имена массивов, то они должны быть объявлены. Границы массивов в этих объявлениях могут быть заданы переменными целого типа, а не только целыми константами, как в головном модуле. Действительные границы массива передаются в процедуру, например, в виде фактических параметров. Для многомерных массивов следует передавать в модуль-процедуру те границы, которые объявлены в головном модуле, или задавать их теми же константами.

Пример 2. Предположим, в программе необходимо несколько раз вычислять среднее взвешенное значение по формуле

$$x_{\text{ср.в}} = \frac{a_1x_1 + a_2x_2 + \cdots + a_nx_n}{a_1 + a_2 + \cdots + a_n}. \qquad (4.13)$$

Внешняя функция с именем XCPB и операторы головного модуля со ссылками на нее записываются следующим образом:

```
C ГОЛОВНОЙ МОДУЛЬ
      DIMENSION A (20), B (20), V (20), W (20)
      READ ...
      ...
      R = XCPB (A, V, 7)
      ...
      K = ...
      ...
      X = 2. * XCPB (B, V, K + 1)/R
      ...
      Z = (XCPB (A, V, 10) + XCPB (B, W, 10))/2
      ...
      ...
      END
C
C МОДУЛЬ — ФУНКЦИЯ
      FUNCTION XCPB (A, X, N)
      DIMENSION A (N), X (N)
      H = 0.
      Z = 0.
      DO 10 I = 1, N
         H = H + A (I) * X (I)
   10    Z = Z + A (I)
      XCPB = H/Z
      RETURN
      END
```

Формальными параметрами функции XCPB являются массивы A, X и переменная N — верхняя граница формальных массивов. После объявления массивов и присваивания начальных значений переменным H и Z вычисляются числитель (H) и зна-

менатель (Z) формулы (4.13). Соответствующие суммы накапливаются с помощью оператора цикла. Затем имени функции присваивается среднее взвешенное значение. Перед заключительной строкой помещен оператор возврата RETURN. Схема описанного алгоритма показана на рис. 4.25.

В головном модуле объявлены четыре одномерных массива: A, B, V и W. Оператор R = XCPB (A, V, 7) присваивает переменной R среднее взвешенное значение семи элементов массива V с весом a_i каждый, т. е.

$$r = \frac{a_1 v_1 + a_2 v_2 + \cdots + a_7 v_7}{a_1 + a_2 + \cdots a_7}.$$

Имена A фактического и формального массивов в данном случае совпадают.

В операторе присваивания X = ... есть ссылка на внешнюю функцию XCPB, причем в указателе функции использован фактический параметр K + 1 (соответствующий формальному параметру N). Значение K должно находиться в пределах 0—19 (чтобы значение K + 1 не выходило за границы объявленных размеров массивов). Обратим внимание, что в головном модуле X — это имя простой переменной, а во внешней функции — имя формального массива.

Следующие две ссылки на внешнюю функцию пояснений не требуют.

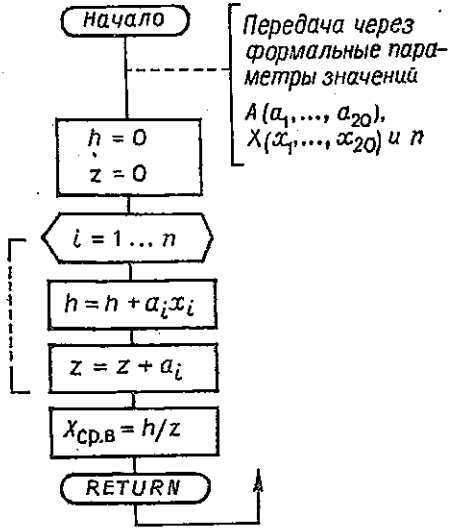


Рис. 4.25. Схема алгоритма для вычисления среднего взвешенного

Если в качестве фактического параметра используется элемент массива, то первому элементу формального массива ставится в соответствие элемент, указанный в списке фактических параметров. Между остальными элементами массивов соответствие устанавливается в порядке их следования. Так, оператор R = XCPB (A (4), V (4), 7) ссылающийся на внешнюю функцию, вызовет присваивание переменной R значения, вычисленного как

$$r = \frac{a_4 v_4 + a_5 v_5 + \cdots + a_{10} v_{10}}{a_4 + a_5 + \cdots a_{10}}.$$

4.14. Подпрограммы

Подпрограмма, как и внешняя функция, является модулем-процедурой, однако ее можно использовать для вычисления и одной, и нескольких величин, например корней уравнения, массивов и т. п.

Заголовком подпрограммы должно быть объявление вида

SUBROUTINE $s(a_1, a_2, \ldots, a_n)$

где s — имя подпрограммы; a_1, a_2, ..., a_n — формальные параметры.

В качестве формальных параметров можно использовать имена переменных, массивов и внешних процедур. К формальным относятся и те параметры, которым присваиваются вычисленные значения. Тип формальных параметров определяется неявно или указывается в объявлениях типа. Границы массивов в объявлениях указываются

целыми числами или переменными целого типа. Допустимы подпрограммы без параметров. В этом случае используются переменные из общих блоков памяти.

В теле подпрограммы должен быть хотя бы один оператор возврата RETURN, который передает управление в вызывающий программный модуль.

Для обращения к подпрограмме используют оператор вызова

$$\text{CALL } s\ (b_1,\ b_2,\ \ldots,\ b_n)$$

где s — имя вызываемой подпрограммы; $b_1, b_2, \ldots, b_n$ — фактические параметры, которые должны быть согласованы с формальными по количеству, порядку следования, типу и удовлетворять перечисленным ранее требованиям.

Оператор вызова устанавливает соответствие между фактическими и формальными параметрами, после чего передает управление первому оператору подпрограммы. Ее выполнение заканчивается оператором RETURN. При этом фактические параметры получают значения, присвоенные в подпрограмме соответствующим формальным параметрам, а управление передается оператору, стоящему за оператором CALL.

Пример 1. Составить подпрограмму для вычисления корней квадратного уравнения $ax^2 + bx + c = 0$. Из головного модуля обратиться к подпрограмме для нахождения корней уравнения $x^2 + px - q = 0$, после чего присвоить переменной A значение $a = x_1 + x_2$, если корни вещественные, или $a = 0$, если корни сопряженные комплексные. Затем вычислить корни уравнения $r^{3,7}y^2 + 16{,}5y + p - 2{,}3r = 0$ и присвоить переменной Z значение $z = y_1 + y_2$ при корнях вещественных или $z = \sqrt{U^2 + W^2}$ при корнях сопряженных комплексных ($y_{1,2} = u \pm iw$).

Необходимые операторы головного модуля и модуль-подпрограмма записываются следующим образом:

```
C ГОЛОВНОЙ МОДУЛЬ
      READ (5,1) P,Q,R
    1 FORMAT (3F10.3)
      ...
      A = 0
      CALL KKU (1.0,P,— Q,X1,X2,K)
      IF (K.EQ.0) A = X1 + X2
      ...
      CALL KKU (R**3.7,16.5,P — 2.3*R,Y1,Y2,K)
      Z = Y1 + Y2
       IF (K.E0.1) Z = SQRT (Y1*Y1 + Y2*Y2)
      ...
      END
C
C МОДУЛЬ — ПОДПРОГРАММА
      SUBROUTINE KKU (A,B,C,X1,X2,N)
      D = B*B — 4*A*C
      R = —B/2/A
      S = SQRT (ABS (D))/2/A
      IF (D) 1,2,2
    2 X1 = R + S
      X2 = R — S
C X1 И X2 — ВЕЩЕСТВЕННЫЕ КОРНИ
      N = 0
      RETURN
    1 X1 = R
      X2 = S
C X1 И X2 — ВЕЩЕСТВЕННАЯ И МНИМАЯ
```

```
C     СОСТАВЛЯЮЩИЕ СОПРЯЖЕННЫХ
C     КОМПЛЕКСНЫХ КОРНЕЙ
          N = 1
          RETURN
          END
```

Перед первым обращением к подпрограмме KKU переменной A присваивается значение «нуль». Затем оператор CALL вызывает подпрограмму KKU. При этом формальным параметрам передаются значения A ⇐ 1, B ⇐ P, C ⇐ —Q. Если вычисленные в подпрограмме корни окажутся вещественными (K ⇐ N = 0), в условном операторе переменной A будет присвоено значение X1 + X2, а если комплексными (K = 1), значение переменной A в головном модуле останется равным нулю.

При следующем вызове подпрограммы KKU формальным параметрам передаются значения A ⇐ R**3.7, B ⇐ 16.5, C ⇐ P — 2.3*R, а в головной модуль возвращаются значения Y1 ⇐ X1, Y2 ⇐ X2 и K ⇐ N. Величины Y1 и Y2 являются вещественными корнями, если K = 0, или вещественной и мнимой частями сопряженных комплексных корней, если K = 1. Следующий оператор, выполняемый после передачи управления из подпрограммы, присваивает переменной Z значение Y1 + + Y2. Оно или сохраняется, или заменяется (при K = 1) величиной Z = = SQRT (Y1**2 + Y2**2).

Пример 2. Составить подпрограмму для умножения матрицы на вектор. Произведением матрицы A на вектор B является вектор C, элементы которого вычисляются по формуле

$$c_i = \sum_{j=1}^{n} a_{ij} b_j, \quad i = 1, 2, \ldots, m, \tag{4.14}$$

где n — количество столбцов матрицы A и элементов вектора B; m — количество строк матрицы A и элементов вектора C.

Схема алгоритма показана на рис. 4.26, а ФОРТРАН-программа, состоящая из головного модуля и подпрограммы, записывается так:

```
C ГОЛОВНОЙ МОДУЛЬ
      DIMENSION P (8.12),Q (12),Z (8)
      READ (5,10) P,Q
      CALL MATPIX (P,Q,Z,8,12)
      WRITE (6,10) Z
   10 FORMAT (8F10.3)
      END
C
C МОДУЛЬ-ПОДПРОГРАММА
C УМНОЖЕНИЕ МАТРИЦЫ A НА ВЕКТОР B
      SUBROUTINE MATRIX (A,B,C,M,N)
      DIMENSION A (M,N),B (N),C (M)
      DO 2 I = 1,M
         S = 0.
         DO 1 J = 1,N
    1       S = S + A (I,J)*B (J)
    2    C (I) = S
      RETURN
      END
```

После заголовка подпрограммы MATRIX записано объявление формальных массивов, верхние границы которых заданы переменными целого типа. Внешний оператор цикла организует переход к вычислению очередного элемента вектора C. В области DO этого оператора находится второй (внутренний) оператор цикла, который производит вычисления по формуле (4.14) для i-го элемента вектора C.

В головном модуле объявлены матрица P, векторы Q и Z. После ввода исходных массивов вызывается подпрограмма MATRIX. В списке фактических параметров находятся массивы P, Q и Z, а также число строк (8) и столбцов (12) массива P. Подпрограмма MATRIX возвращает на место фактического параметра Z вектор-произведение P*Q.

Последний оператор задает вывод вектора Z на АЦПУ. Вектор печатается одной строкой — 8 элементов в формате F10.3.

4.15 Объявления внешних имен

Внешние функции и подпрограммы могут иметь в списке формальных параметров имена других внешних процедур. При обращениях к таким функциям и подпрограммам в списках фактических параметров указателей функций и операторов CALL записываются имена фактических внешних процедур. Поскольку они не отличаются от имен переменных и массивов, необходимо сообщить транслятору, какие из параметров являются именами фактических внешних процедур. Это осуществляется с помощью объявлений внешних имен

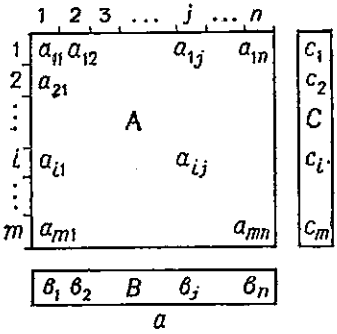
$$\text{EXTERNAL } v_1, v_2, \ldots, v_n$$

где $v_1, v_2, \ldots, v_n$ — имена внешних процедур, используемые в качестве фактических параметров при обращении к другой внешней процедуре.

Объявление EXTERNAL помещают перед разделом операторов.

Пример. Допустим, в программе необходимо присвоить переменной S значение $S = \frac{6{,}8\, I_1}{r}$, где интеграл

$$I_1 = \int_{T_1}^{T_2} f_1(t)\, dt = \int_{T_1}^{T_2} 7{,}3e^{-1{,}2t} \sin(5{,}4t + 0{,}8)\, dt, \qquad (4.15)$$



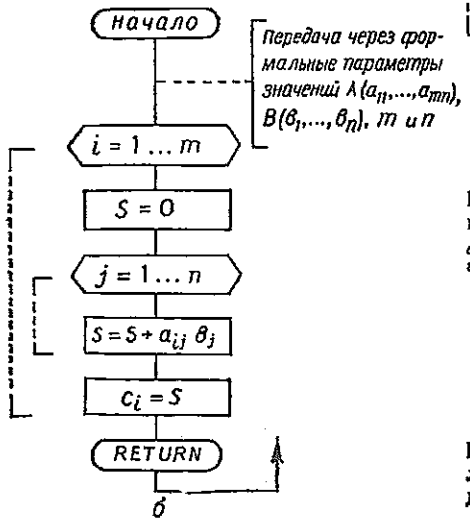


Рис. 4.27. Схема алгоритма для вычисления определенного интеграла методом трапеций

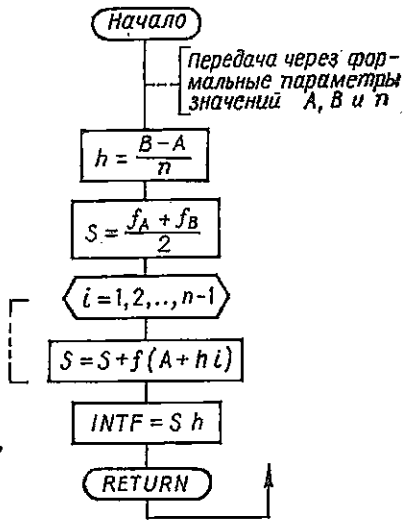


Рис. 4.26. Умножение матрицы A на вектор B:

а — матрица A, векторы B и C; *б* — схема алгоритма

а в другом месте той же программы переменной q должно быть присвоено значение $q = aI_2$, где I_2 также является определенным интегралом, но другой функции:

$$I_2 = \int_{x_1}^{x_2} f_2(x)\,dx = \int_{2}^{6{,}5} \frac{\ln x + x^2}{1 + 0{,}8x}\,dx. \qquad (4.16)$$

Поскольку в программе необходимо дважды выполнять процедуру интегрирования, целесообразно оформить ее в виде внешней функции. Для вычисления интеграла воспользуемся методом трапеций:

$$I = \int_{A}^{B} f(x)\,dx = h\left(\frac{f_0}{2} + f_1 + f_2 + \cdots + f_{n-1} + \frac{f_n}{2}\right),$$

где $h = (B - A)/n$ — длина интервала разбиения (шаг интегрирования); $f_0 = f_A = f(A)$; $f_n = f_B = f(B)$; $f_i = f(x_i)$; $x_i = A + h_i$, $(i = 1, 2, 3, \ldots, n - 1)$; n — число интервалов разбиения на отрезке AB.

Схема алгоритма показана на рис. 4.27. ФОРТРАН-программа, содержащая внешнюю функцию с именем INTF, записывается следующим образом:

```
      REAL INTF
      EXTERNAL F1,F2
      READ (5,5) R,T1,T2,N
    5 FORMAT (3F10.3,I5)
        ...
        S = INTF (F1,T1,T2,N)*6.8/R
        ...
      A = ...
      ...
      Q = A*INTF (F2,4.0,7.5,20)
      ...
      WRITE (6,6)...
    6 FORMAT (...
      ...
      END
      REAL FUNCTION INTF (F,A,B,N)
      H = (B - A)/N
      S = (F (A) + F (B))/2.
      N1 = N - 1
      DO 10 I = 1,N1
   10   S = S + F (A + H*I)
      INTF = S*H
      RETURN
      END
      FUNCTION F1 (T)
      F1 = 7.3*EXP (-1.2*T)*SIN (5.4*T + 0.8)
      RETURN
      END
      FUNCTION F2 (X)
      F2 = (ALOG (X) + X*X)/(1. + 0.8*X)
      RETURN
      END
```

Тип REAL в заголовке этой функции указывает, что результат вычисления (значение интеграла) — вещественное число. В списке формальных параметров кроме пределов интегрирования A, B и числа интервалов разбиения N помещено имя внешней функции F. Это необходимо, поскольку процедура INTF универсальная и, следовательно, в заголовке должно быть указано, какая именно функция подлежит интегрированию.

При обращении из головного модуля к процедуре INTF на место формального параметра F передается имя той внешней процедуры, в которой вычисляются зна-

чения подынтегральной функции. Внешние функции с именами F1 и F2 помещены после модуля-процедуры INTF.

После оператора ввода и других операторов (они не относятся к рассматриваемой теме и поэтому обозначены многоточием) записан оператор присваивания

S = INTF (F1, T1, T2, N) * 6.8/R

При его выполнении по имени INTF отыскивается соответствующая внешняя функция, и формальные параметры, указанные в ее заголовке, заменяются фактическими (F ⇐ F1, A ⇐ T1, B ⇐ T2, N ⇐ N).

После передачи фактических параметров выполняется модифицированный раздел операторов функции INTF:

```
      H = (T2 — T1)/N
      S = (F1 (T1) + F1 (T2))/2.
      N1 = N — 1
      DO   10 I = 1, N1
10        S = S + F1 (T1 + H * I)
      INTF = S * H
      RETURN
```

В операторном разделе функции INTF ссылка на внешнюю функцию F1 встречается трижды. При каждой такой ссылке выполняется функция с именем F1, причем вместо формального параметра T используются фактические: первый раз — значение T1, второй — значение T2, а затем N1 раз (в цикле) — значения T1 + H*I.

По завершении вычислений в модуле-процедуре INTF оператор RETURN передает значение интеграла $I_2 = \int_{T_1}^{T_2} f_1(t)\,dt$ на место указателя функции в головной модуль. Отсюда продолжаются вычисления: значение интеграла умножается на 6,8, делится на R, после чего результат присваивается переменной S.

Аналогично выполняется оператор присваивания

Q = A * INTF (F2, 4.0, 7.5, 20),

но со ссылками на внешнюю функцию F2 и вычислениями по формуле (4.16) значений F2 (4.0), F2 (7.5) и F2 (4.0 + H*I), I = 1, 2, 3, ..., 19.

Полученное значение интеграла передается в головной модуль, умножается на A и присваивается переменной Q. Переменная A, разумеется, к этому времени должна быть определена. Напомним, что эта переменная не имеет ничего общего с формальным параметром A в модуле INTF.

4.16. Объявления общих объектов

В языке ФОРТРАН допускается два способа обмена данными между программными модулями: передача их через формальные — фактические параметры (рассмотрен ранее) и объявление некоторых объектов (переменных, массивов) расположенными в *общих блоках* (областях памяти ЭВМ). Такие блоки становятся общими для тех программных модулей, в которых помещены объявления

COMMON /*имя*$_1$/ *сп*$_1$ /*имя*$_2$/ *сп*$_2$ ... /*имя*$_n$/ *сп*$_n$

где *имя*$_1$, *имя*$_2$, ..., *имя*$_n$ — имена общих блоков; *сп*$_1$, *сп*$_2$, ..., *сп*$_n$ — списки объектов в общих блоках.

Имена общих блоков образуют по тем же правилам, что и другие символические имена. Они не обязательно должны быть уникальными и могут совпадать с именами других объектов программы. В объявлениях COMMON один блок может быть непомеченным (неименованным). Если это первый блок, то обе дробные черты перед списком могут отсутствовать.

Каждый список состоит из разделенных запятыми элементов (в частности, одного). Ими могут быть имена переменных, массивов и

описания массивов, не являющиеся формальными параметрами. Общие объекты должны быть согласованы по типу и длине.

Примеры объявлений общих объектов:

```
COMMON B3, Q1/B2/X (10)/B3/A, ALFA, BETA
COMMON/B3/X, Y, Z//W,Q2
```

В первом из них три общих блока: непомеченный, после него с именем B2 и затем с именем B3. В списке непомеченного блока встречается также имя переменной B3, что вполне допустимо. Во втором объявлении сначала записан общий блок B3, а затем непомеченный.

Объекты, которые содержатся в списках одноименных блоков, расположены в общих областях (ячейках) памяти в той же последовательности, что и имена в списках общих блоков. Если приведенные объявления записаны в разных модулях ФОРТРАН-программы, то в одних и тех же областях памяти располагаются соответственно переменные B3 и W, Q1 и Q2, а также A и X, ALFA и Y, BETA и Z; первые две пары объявлены общими для непомеченного блока, а последние три — для блока B3.

Если элементом списка является описание массива, оно заменяет объявление массива (в других спецификациях этот массив уже не объявляют). Верхние границы индексов в таких описаниях могут быть указаны только целыми числами. Эквивалентны, например, следующие записи:

```
COMMON D,U (30)   и   DIMENSION U (30)
                      COMMON D,U
```

Пример 1. В головном модуле объявлен непомеченный общий блок

```
COMMON A, X1, S. Q (3)
```

а в подпрограмме

```
COMMON B, A, R (4)
```

В памяти ЭВМ переменным отводятся такие общие ячейки.

Номер ячейки	*Переменная в головном модуле*	*Переменная в подпрограмме*
l	A	B
$l+1$	X1	A
$l+2$	S	R (1)
$l+3$	Q (1)	R (2)
$l+4$	Q (2)	R (3)
$l+5$	Q (3)	R (4)

Если в подпрограмме переменной A присвоено некоторое значение, а затем управление передано в головной модуль, то это же значение будет иметь теперь переменная X1. Таким образом, принадлежность к одной и той же ячейке памяти определяется не тождественностью, а последовательностью записи имен в списке элементов общего блока. В частности, эти имена могут быть и одинаковыми.

Пример 2. Изменим рассмотренную в примере 1 подразд. 4.15 программу таким образом, чтобы интегралы I_1 и I_2 можно было вычислить по более общим формулам:

$$I_1 = \int_{T_1}^{T_2} f_1(t)\,dt = \int_{T_1}^{T_2} Ge^{-\alpha t}\sin(\omega t + \varphi)\,dt, \qquad (4.17)$$

$$I_2 = \int_{x_1}^{x_2} f_2(x)\,dx = \int_{x_1}^{x_2} \frac{\ln x + x^2}{1 + bx}\,dx. \qquad (4.18)$$

Значения G, α, ω, φ и b нельзя передавать через формальные параметры, поскольку количество фактических параметров при вычислении интегралов I_1 и I_2 не одинаково. Передать значения этих параметров можно с помощью общих блоков:

```
C     ГОЛОВНОЙ МОДУЛЬ
      REAL INTF
      COMMON G, AL, W, F1/B/B
      EXTERNAL F1, F2
      READ (5,15) . . .
   15 FORMAT (. . .)
      . . .
      S = INTF (F1, T1, T2, N) * 6.8/R
      . . .
      Q = A * INTF (F2, X1, X2, M)
      . . .
      WRITE (6,16) S, Q
   16 FORMAT (2E16.5)
      . . .
      END
C
C ВНЕШНЯЯ ФУНКЦИЯ ДЛЯ ВЫЧИСЛЕНИЯ ИНТЕГРАЛА
      REAL FUNCTION INTF (F, A, B, N)
      H = (B — A)/N
      S = (F (A) * F (B))/2.
      N1 = N — 1
      DO 10 I = 1, N1
   10    S = S + F (A + H * I)
      INTF = S * H
      RETURN
      END
C
C ВНЕШНИЕ ФУНКЦИЙ F1 И F2
      FUNCTION F1 (T)
      COMMON G, AL, W, F1
      F1 = G * EXP (— AL * T) * SIN (W * T + F1)
      RETURN
      END
C
      FUNCTION F2 (X)
      COMMON /B/B
      F2 = (ALOG (X) + X * X)/(1. + B * X)
      RETURN
      END
```

В головном модуле объявлены непомеченный общий блок со списком переменных G, AL, W, F1 и общий блок B с одной переменной B в списке. Эти блоки объявлены также во внешних функциях F1 (непомеченный) и F2 (с именем B). Одноименные общие блоки содержат в списках одинаковые имена переменных, что является частным случаем.

Оператор присваивания

```
S = INTF (F1, T1, T2, N) * 6.8/R
```

передает во внешнюю функцию INTF фактические параметры, в числе которых имя внешней функции F1 (оно указано в объявлении EXTERNAL). При вычислении интеграла встречаются ссылки на внешнюю функцию F1 (формальный параметр F заменен фактическим F1), в результате чего каждый раз вычисляется значение подынтегрального выражения:

```
F1 = G * EXP (— AL * T) * SIN (W * T + F1)
```

Формальный параметр T заменяется фактическим: первый раз это T1 (формальные параметры A и B в процедуре INTF заменены фактическими T1 и T2), затем T2, после чего многократно (в цикле с переменной I) вместо параметра T берется факти-

ческий T1 + H*I. Значения переменных G, AL, W и FI берутся из непомеченного общего блока, объявленного и в головном модуле, и в модуле с именем F1. Эти переменные должны быть определены в головном модуле до выполнения оператора

S = INTF (F1, T1, T2, N) * 6.8/R

Аналогично выполняется оператор

Q = A * INTF (F2, X1, X2, M)

но при вычислении F2 значение переменной B берется из общего блока B.

4.17. Библиотека научных подпрограмм

В состав математического обеспечения ЕС ЭВМ обычно включают библиотеку научных подпрограмм (БНП) [1]. Она содержит более 500 подпрограмм, которые широко применяются в инженерных и экономических расчетах, а также в научных исследованиях. С помощью подпрограмм, входящих в состав БНП, можно реализовать часто встречающиеся численные методы анализа, например размещение матриц в памяти, операции с матрицами, системы линейных алгебраических уравнений и родственные темы, операции с полиномами, корни полиномов и нелинейных уравнений, экстремумы функций, подстановки, суммы и пределы последовательностей, численное дифференцирование и интегрирование, интерполяция, сглаживание и аппроксимация функций, анализ Фурье, решение обыкновенных дифференциальных уравнений и их систем, корреляция и регрессия, планирование эксперимента, временные ряды, непараметрическая статистика, образование случайных переменных, элементарная статистика и др.

Описания численных методов и инструкции ко всем подпрограммам БНП содержатся в специальных выпусках «Математическое обеспечение ЕС ЭВМ».

Обращение к подпрограмме осуществляется оператором CALL, фактические параметры которого должны быть согласованы с формальными по количеству, порядку следования и типу. Если в списке фактических параметров оператора CALL встречается имя внешней функции (она составляется пользователем), то его необходимо указать в объявлении EXTERNAL. Процедура с этим именем должна включаться в состав ФОРТРАН-программы в виде отдельного модуля. При работе с массивами в подпрограмму следует передавать (через формальные — фактические параметры) те же их размеры, что и в головном модуле.

Пример. Составить ФОРТРАН-программу для решения системы линейных алгебраических уравнений с четырьмя неизвестными

$$\begin{aligned} 3{,}8x_1 + 5{,}4x_2 + 19{,}2x_3 - 0{,}3x_4 &= 5{,}06; \\ 15{,}4x_1 + 8{,}7x_2 - 2{,}6x_3 &= 191{,}4; \\ -2{,}1x_1 + 0{,}3x_2 + 4{,}0x_3 + 8{,}4x_4 &= 82{,}01; \\ -1{,}5x_2 + 41{,}2x_3 - 0{,}7x_4 &= -222{,}22. \end{aligned}$$

[1] Комплекс входящих в библиотеку подпрограмм разработан Институтом математики АН БССР.

Для решения системы воспользуемся подпрограммой SIMQ, имеющейся в БНП. ФОРТРАН-программа, из которой производится обращение к указанной подпрограмме, записывается так:

```
C РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ.
C ОПИСАНИЕ И ВВОД ИСХОДНЫХ МАССИВОВ
      DIMENSION A(4,4), B(4)
      READ(5, 10) A, B
   10 FORMAT (8F10.3)
C ВЫВОД ИСХОДНЫХ ДАННЫХ
      WRITE (6,20) ((A(I, J), J = 1,4), B(I), I = 1,4)
   20 FORMAT (5X, 5F10.4)
C ОБРАЩЕНИЕ К ПОДПРОГРАММЕ SIMQ
      CALL SIMQ(A, B, 4, K)
C ВЫВОД РЕШЕНИЯ
      WRITE (6,30) B, K
   30 FORMAT (/5X, 4F10.4,I10//)
      END
```

После объявления массивов A (матрица коэффициентов) и B (вектор свободных членов) оператор READ (1,10) осуществляет их ввод. При выбранном формате на первых двух картах должны быть по столбцам отперфорированы числа, образующие матрицу A, а на третьей карте — свободные члены (массив B):

Номер перфокарты	Номер колонки							
	1	11	21	31	41	51	61	71
1	3.8	15.4	—2.1	0.	5.4	8.7	0.3	—1.5
2	19.2	—2.6	4.	41,2	—0.3	0.	8.4	—0.7
3	5.06	191.4	82.01	—222.22				

Оператор WRITE (6,20) выводит четыре строки исходных данных (коэффициенты при неизвестных и свободные члены). Оператор

```
CALL SIMQ(A, B, 4,K)
```

вызывает для исполнения библиотечную подпрограмму SIMQ. В списке фактических параметров оператора вызова указаны имена массивов (A, B) и их размеры (4). Решение системы (x_1, ..., x_4) помещается программой SIMQ на место вектора B, который выводится на печать оператором

```
WRITE (3, 30) B, K
```

Одновременно печатается значение переменной K. Если K = 0, то система решена, а если K = 1, система решения не имеет.

Исходные данные и полученные значения неизвестных ($x_1 = 2{,}5$, $x_2 = 16{,}2$, $x_3 = -4{,}6$, $x_4 = 12$) распечатываются на АЦПУ в таком виде:

```
   3.8000     5.4000    19.2000   — 0.3000     5.0600
  15.4000     8.7000   — 2.6000     0.0      191.4000
— 2.1000      0.3000     4.0000     8.4000    82.0100
   0.0      — 1.5000    41.2000   — 0.7000   — 222.2200
   2.5000     16.2000   — 4.6000    12.0000     0
```

5. ОПЕРАЦИОННАЯ СИСТЕМА ЕС ЭВМ

5.1. Общие сведения

Как уже отмечалось, программу, записанную на алгоритмическом языке (например, на ФОРТРАНе), ЭВМ непосредственно исполнять не могут. Ее необходимо перевести на язык машинных команд. Такой перевод называется *трансляцией*.

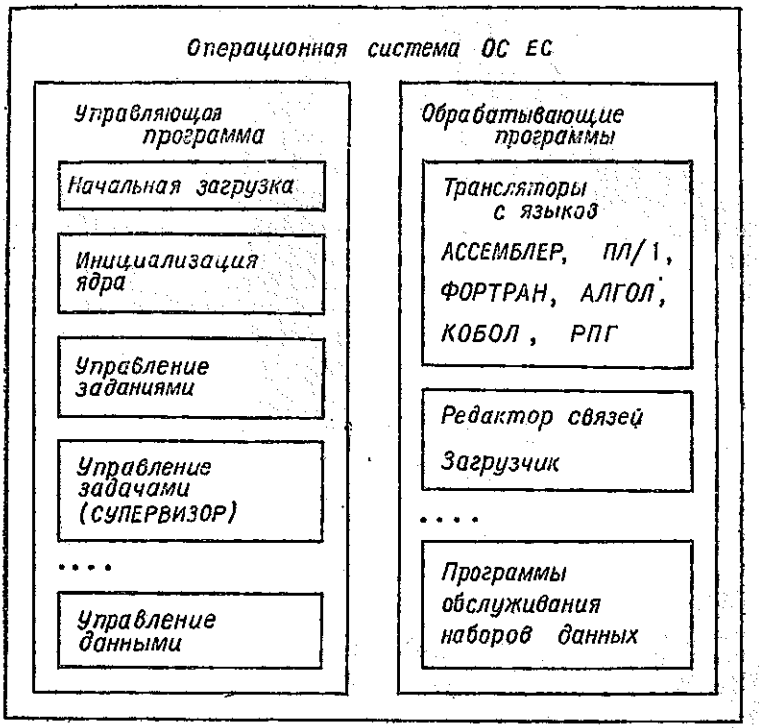


Рис. 5.1. Состав операционной системы ЕС ЭВМ

Программа пользователя может состоять из нескольких программных модулей, включать в себя модули-процедуры стандартных функций и подпрограммы БНП. Все эти программные модули следует скомпоновать (связать) в единую программу, т. е. *отредактировать*.

Прием заданий к исполнению, распределение между ними ресурсов[1], трансляцию исходных модулей, редактирование связей, управление ходом вычислительного процесса, обработку ошибок, связь оператора с ЭВМ и ряд других вспомогательных функций выполняют различные служебные программы, совокупность которых называется *операционной системой*.

В программном обеспечении машин серии ЕС ЭВМ имеются две операционные системы: ДОС ЕС и ОС ЕС. Для младших моделей ЕС ЭВМ разработана ДОС ЕС — дисковая операционная система, ориентированная на сравнительно небольшой объем оперативной памяти и небольшое количество внешних устройств. Более широкими возможностями обладает ОС ЕС, применение которой значительно повышает эффективность использования ЭВМ. В настоящее время ОС ЕС используется практически на всех машинах, имеющих достаточно мощную конфигурацию внешних устройств и объем оперативной памяти не менее 256К.

[1] К ресурсам вычислительной системы относятся центральный процессор, рабочие области основной памяти, запоминающие устройства прямого доступа (например, НМД), каналы и устройства ввода-вывода, а также некоторые программы и данные.

Основным для ОС ЕС является мультипрограммный режим работы с переменным числом выполняемых задач или режим MVT [1], в котором операционная система организует ввод заданий, записывает их на диски и ставит в очередь на выполнение. Очередность зависит от приоритета задания. После выбора очередного задания ему выделяются необходимые ресурсы и время центрального процессора. В режиме MVT одновременно решаются несколько задач [2], что обеспечивает высокую производительность и эффективное использование ресурсов вычислительной системы.

Программы ОС ЕС размещаются во внешней памяти ЭВМ на магнитных дисках, называемых резидентными. Основу операционной системы составляет управляющая программа (рис. 5.1), которая выполняет такие функции: подготовку операционной системы к работе, прием задания и подготовку его к выполнению, управление ходом выполнения всех задач в системе (комплекс программ СУПЕРВИЗОР) и др.

Кроме того, в состав ОС ЕС входят обрабатывающие программы, наиболее важные функции которых заключаются в следующем:

перевод программ с алгоритмического языка в коды команд ЭВМ, который выполняют программы-трансляторы;

редактирование связей, осуществляемое обрабатывающими программами: редактор связей, выборка, загрузчик;

обеспечение работы с наборами данных, в частности с библиотеками.

5.2. Этапы обработки программ операционной системой

Программа, написанная на одном из алгоритмических языков, должна быть введена в ЭВМ. Для этого ее записывают на какой-либо носитель информации, например на перфокарты. Подготовленную таким образом программу называют *исходным модулем* (рис. 5.2). Он может быть введен в ЭВМ с перфокарточного устройства ввода, НМЛ, НМД (если он был предварительно записан на магнитную ленту или диски) или с клавиатуры дисплея.

Первый этап обработки исходого модуля на ЭВМ — трансляция — производится специальной программой операционной системы. В ОС ЕС программа-транслятор с языка ФОРТРАН-IV имеет имя IEYFORT [3] и хранится в системной библиотеке. Программа может состоять из нескольких исходных модулей, например головного и модулей-процедур. В этом случае каждый модуль транслируется независимо от других. В процессе трансляции на печать выдается исходная программа, диагностические сообщения об ошибках и другая информация.

---

[1] Расшифровка сокращенных обозначений дана в приложении.

[2] Например, если при решении одной из задач осуществляется вывод результатов, то освободившийся центральный процессор передается другой задаче.

[3] Указанное имя имеет транслятор уровня G. Кроме того, в составе ОС ЕС существует транслятор уровня H — оптимизирующий.

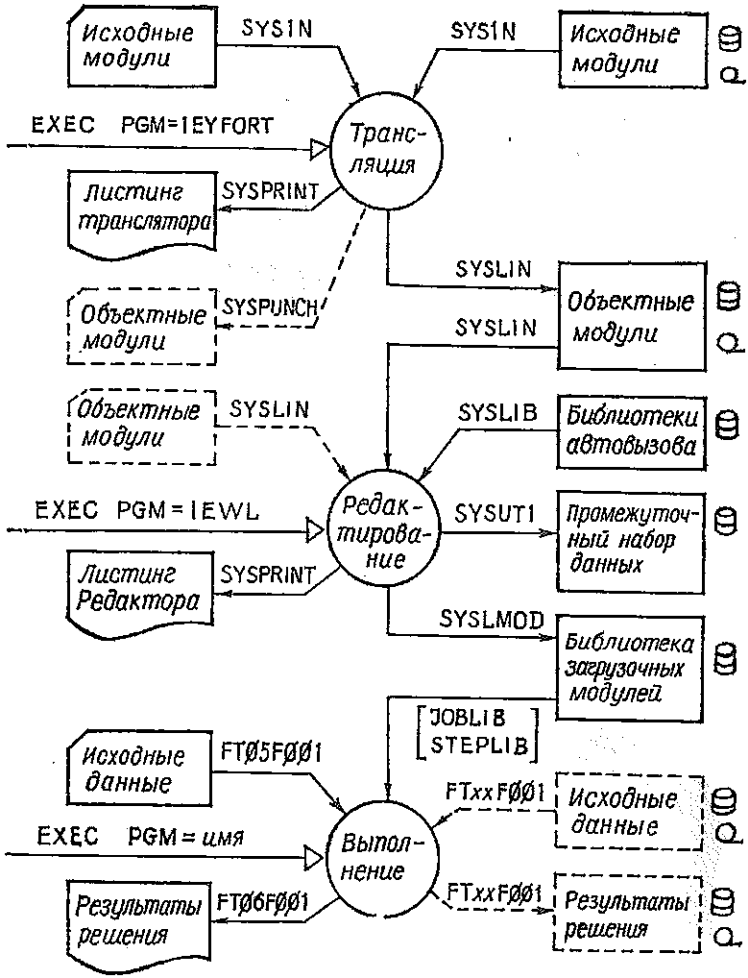


Рис. 5.2. Этапы обработки программы, наборы данных и dd-имена

Результатом работы транслятора является один или несколько *объектных модулей*. Каждый из них представляет собой программу в кодах машинных команд (программа на машинном языке), которая, однако, к исполнению не готова. К ней следует добавить другие объектные модули (стандартных функций, из БНП и специальные), а затем из них создать готовую к выполнению программу.

Функцию объединения объектных модулей в единый *загругочный модуль*, т. е. редактирование связей, выполняет программа-редактор связей (см. рис. 5.1), которая хранится в системной библиотеке и имеет имя IEWL. Подготовленный к дальнейшему использованию загрузочный модуль помещается во временную библиотеку на магнитных дисках. В процессе редактирования на печать выдается дополнительная диагностическая информация.

Настройку и загрузку программы на определенное место в основной памяти ЭВМ осуществляет специальная программа (выборка), входящая в состав управляющей программы.

При выполнении программы необходимые исходные данные вводятся с устройств, определяемых программистом в операторе READ, чаще всего с перфокарточного устройства ввода ($k = 5$ в операторе READ). Результаты решения выводятся на устройства, определяемые программистом в операторе WRITE, обычно на системное устройство печати ($k = 6$ в операторе WRITE), для которого назначено АЦПУ.

Этапы обработки наборов данных при решении задачи в ОС ЕС показаны на рис. 5.2.

5.3. Наборы данных. Библиотеки ОС ЕС

Наборы данных. Набором данных называют определенным образом организованную и поименованную совокупность данных, объединенных общим назначением. Они размещаются на внешних носителях информации: перфокартах, перфолентах, магнитных лентах или магнитных дисках. Примеры наборов данных: совокупность исходных данных, подготовленная для обработки на ЭВМ и хранящаяся на магнитной ленте; ФОРТРАН-программа на перфокартах; объектный модуль, полученный в результате трансляции и помещенный на магнитные диски; библиотека подпрограмм на магнитных дисках; результаты решения задачи, направляемые в выходной поток на печать.

Набор данных состоит из *блоков,* размеры которых определяются физическими свойствами носителей информации. Примерами блоков данных могут служить перфокарта, строка текста на АЦПУ, информация, расположенная между двумя промежутками на магнитной ленте или на магнитном диске. Каждый блок может содержать одну или несколько логических записей.

Наборы данных могут быть временными и постоянными. *Временные* наборы создаются при выполнении шагов задания и автоматически уничтожаются после его завершения, а *постоянные* сохраняются и существуют до тех пор, пока не будет дано специальное указание об их уничтожении.

Постоянные наборы данных могут быть *каталогизированы,* т. е. имя набора и некоторые другие сведения о нем записываются в системный каталог, расположенный на дисках. Каталогизированный набор данных отыскивается по его имени (без указания устройства и тома). Для доступа к некаталогизированному сохраняемому набору данных необходимо кроме имени указать место размещения.

Организация набора данных может быть прямой, последовательной, библиотечной и индексно-последовательной. *Прямая* организация позволяет обратиться к любой записи набора без чтения предыдущих, например, по ее действительному адресу. При *последовательной* организации обратиться к какой-либо записи набора можно только после прочтения всех предыдущих. *Библиотечная* организация набора данных заключается в создании разделов, каждый из которых, как и библиотека в целом, имеет имя. Оно указывается в оглав-

лении [1], размещенном в начале библиотечного набора. Библиотечные наборы данных можно располагать только на устройствах прямого доступа (на магнитных дисках).

Библиотеки. Библиотека представляет собой набор данных, состоящий из именованных разделов. Каждая из них имеет имя и может быть каталогизирована.

Имя библиотеки, как и имя раздела (программы), должно содержать 1—8 символов и начинаться с буквы. Библиотеки размещаются на магнитных дисках и обычно используются для хранения программ и подпрограмм. Программы и подпрограммы могут храниться в библиотеках в виде исходных, объектных и загрузочных модулей.

Библиотеки исходных модулей предназначены для хранения отлаженных и готовых к трансляции программ и подпрограмм, записанных на ФОРТРАНе или другом языке программирования. В *библиотеках объектных модулей* находятся оттранслированные подпрограммы, подключаемые к создаваемому модулю на стадии редактирования. Все программы, подлежащие выполнению в ОС ЕС, должны храниться в *библиотеках загрузочных модулей.*

Библиотеки могут быть системными и личными.

Системные библиотеки — это постоянные каталогизированные библиотечные наборы данных, создаваемые при генерации операционной системы [2]. В них находятся программы самой операционной системы. В общих системных библиотеках загрузочных модулей хранятся управляющие и обрабатывающие программы, в том числе трансляторы и редактор связей (SYS1.LINKLIB), программы математического обеспечения ФОРТРАНа (SYS1.FORTLIB), подпрограммы БНП (например, SYS1.SSPLIB) [3]. В системной библиотеке исходных модулей SYS1.PROCLIB находятся каталогизированные процедуры.

Личные библиотеки — это постоянные, часто каталогизируемые, библиотечные наборы данных, которые создают пользователи для хранения разработанных ими программ и подпрограмм.

Кроме того, существуют *временные библиотеки,* предназначенные для использования в пределах одного задания, после завершения которого они уничтожаются. Имя временной библиотеки должно начинаться символами && или &, являющимися признаком временного набора данных.

5.4. Язык управления заданиями

Структура задания. Для того чтобы программа, написанная на языке ФОРТРАН (или другом языке программирования), транслировалась и выполнялась под управлением ОС ЕС, необходимо составить задание и ввести его в ЭВМ.

---

[1] Следует различать оглавление библиотеки и оглавление тома. Библиотека является набором данных внутри тома. Один том (пакет дисков), как правило, содержит несколько библиотек.

[2] Генерация — это процесс формирования из универсального набора программ рабочего варианта операционной системы, учитывающего особенности конкретной ЭВМ и условия ее применения.

[3] Последние две библиотеки должны быть определены с помощью оператора DD.

Задание состоит из последовательности управляющих операторов, которые записываются на бланках в позициях 1—71. В первых двух позициях размещают две наклонные черты — признак управляющего оператора. Исключение составляет ограничительный оператор/ *.

Управляющий оператор может содержать до четырех полей, разделенных пробелами, и имеет следующую структуру:

//имя ␣ операция ␣ параметры ␣ комментарии

В *поле имени,* начиная с третьей позиции, помещают идентификатор, который содержит от 1 до 8 символов и начинается с буквы. В некоторых операторах имя может отсутствовать.

В *поле операции* записывают код операции: JOB, EXEC, DD или некоторые другие.

В *поле параметров* размещают позиционные и (или) ключевые параметры. П о з и ц и о н н ы е (если они есть) записываются первыми в строго установленной последовательности, а к л ю ч е в ы е — в произвольной. Параметры отделяются друг от друга запятой. Если какие-либо из параметров не указаны, то по умолчанию действуют стандартные или принятые на ВЦ их значения.

В *поле комментариев* (если они есть) включается пояснительная информация.

Управляющий оператор при необходимости может быть записан на нескольких перфокартах. Прерванная запись должна заканчиваться запятой, которая ставится после параметра или подпараметра. Продолжение оператора записывается на следующей перфокарте с позиций 4—16. В первых двух позициях размещаются две наклонные черты. Если переносится комментарий, то в колонке 72 необходимо поставить какой-либо символ.

Начинается задание оператором JOB, а заканчивается пустым оператором (// пусто) или оператором JOB следующего задания.

Задание состоит из одного или нескольких шагов (пунктов). *Шаг задания* — это совокупность сведений, необходимых для выполнения одной программы, например транслятора, редактора связей. Каждый шаг начинается оператором EXEC, указывающим на необходимость выполнить определенную программу. За ним следуют операторы DD, описывающие используемые наборы данных. При необходимости в шаге задания могут присутствовать наборы данных, вводимые во входном потоке.

В ЭВМ задания вводятся через перфокарточное устройство или с клавиатуры дисплея.

Оператор начала задания. Каждое задание начинается оператором

//имя ␣ JOB *␣ параметры ␣ комментарии*

Имя задания следует выбирать таким образом, чтобы оно по возможности не совпадало с именами других одновременно выполняемых заданий.

Код операции JOB отделяется от имени не менее чем одним пробелом.

В поле параметров первые два — позиционные (учетная информация и идентификация программиста), вид которых устанавливается

вычислительным центром. Из возможных ключевых параметров (MSGLEVEL, MSGCLASS, PRTY, REGION, TIME и др.) рассмотрим только один — MSGLEVEL. С его помощью устанавливают уровень сообщений системы. Этот параметр записывается следующим образом:

MSGLEVEL = (a, b)

Подпараметр a указывает состав выводимой на печать информации о задании. Если $a = 0$, печатается только оператор JOB, при $a = 2$ — все управляющие операторы из входного потока, а при $a = 1$ — все управляющие операторы задания, включая операторы каталогизированных процедур.

Подпараметр b принимает значение 0 или 1. Число 1 на месте b задает распечатку информации о распределении внешних устройств и состоянии наборов данных после завершения каждого шага задания. При $b = 0$ эти сообщения не печатаются.

Если параметр MSGLEVEL отсутствует, вступает в действие принятое на ВЦ значение, обычно (0, 1).

Рассмотрим примеры операторов начала задания. В операторе

//EXML51 JOB MSGLEVEL = (0,0) ИВАНОВ А.В.

позиционные параметры не записаны (учет на ВЦ не ведется); фамилия программиста размещена в поле комментариев.

В операторе

//EXML52 JOB

параметры и комментарии отсутствуют, значения параметров стандартные.

Оператор начала шага задания. Он записывается в виде

// имя ␣ EXEC ␣ *параметры* ␣ *комментарии*

Имя, выбираемое произвольно, может отсутствовать. Оно необходимо только при ссылке на конкретный оператор EXEC.

В операторе EXEC записывают один позиционный параметр. Он указывает имя программы или каталогизированной процедуры, которая будет выполняться на данном шаге задания. Для вызова программы из раздела библиотеки этот параметр имеет вид

PGM = *имя-программы*

или

PGM = **.имя-шага.* dd-*имя*

а для выполнения каталогизированной процедуры

PROC = *имя-процедуры*

или

имя-процедуры

Например,

```
//STEP3 EXEC PGM = RASTR2
//        EXEC FORTGCLG
```

Первый из этих управляющих операторов (его имя STEP3) вызывает для исполнения программу с именем RASTR2, а второй (безымянный) — каталогизированную процедуру FORTGCLG.

Из ключевых параметров рассмотрим наиболее распространенные: PARM, COND, REGION, TIME.

Параметр PARM передает обрабатывающей программе дополнительную информацию о режимах работы. Его записывают в таком формате:

$$\text{PARM} = (p_1, p_2, \ldots, p_n)$$

Если указан один из подпараметров p, то скобки не обязательны. При отсутствии параметра PARM для обрабатывающих программ (транслятор, редактор связей и др.) устанавливаются стандартные режимы.

Параметр COND определяет условия обхода шага задания при выявлении на предыдущих шагах ошибок, делающих выполнение данного шага бессмысленным. Условие, при котором шаг задания не будет осуществляться, записывается таким образом:

$$\text{COND} = ((y_1), (y_2), \ldots, (y_n))$$

Каждое условие пропуска данного шага задания $y_1, \ldots, y_n$ имеет вид

(*число, отношение, имя-шага*)

Отошение кодируется буквами LT, LE, EQ, NE, GE, GT. Имя-шага — это имя одного из предшествующих операторов EXEC, при выполнении которого вырабатывается проверяемый код завершения шага. Код сравнивается с числом, и если хотя бы одно условие удовлетворяется, данный шаг задания пропускается.

Параметр REGION осуществляет запрос на выделение памяти машины, когда стандартного ее размера (60K) недостаточно. Записывается он как

$$\text{REGION} = n\text{K}$$

где n — целое число, указывающее необходимый объем основной памяти.

Параметр TIME ограничивает время выполнения шага задания. В его записи

$$\text{TIME} = (m, s)$$

m означает минуты, а s — секунды. Когда секунды не указываются, отпадает необходимость в скобках. Стандартное время параметра TIME равно 30 мин.

Например, при записи оператора начала шага задания

```
//EXML53 EXEC PGM = IEWL,REGION = 96K,
//  PARM = (MAP,LET),COND = (4,LT,FORT)
```

будет выполняться редактирование объектного модуля, так как в поле параметра PGM указано имя IEWL. Для выполнения шага задания запрашивается раздел основной памяти объемом 96K, устанавливаются режимы работы редактора связей MAP и LET. Редактирование не будет выполняться, если при трансляции (имя шага FORT) код завершения (уровень ошибки) больше чем 4.

Оператор описания наборов данных. Все наборы данных, используемые в задании, должны быть определены. Это означает, что в уста-

новленном порядке необходимо указать их имена, размеры, структуру, место хранения и другие небходимые операционной системе данные. Такие сведения содержатся в операторах описания наборов данных DD, которые имеют следующий формат:

// имя ␣ DD ␣ *параметры* ␣ *комментарии*

Имя оператора DD (dd-имя) должно присутствовать во всех случаях. Иногда группа операторов DD может иметь одно dd-имя. Оно записывается в поле первого оператора, а в остальных это поле остается свободным. Данная группа операторов DD используется для описания сцепленных наборов данных, которые рассматриваются обрабатывающей программой как один набор. Естественно, что сцепленные наборы должны иметь одинаковую организацию, длину и формат записей.

Пять dd-имен зарезервированы в ОС ЕС для специальных целей, среди них имена наборов данных (JOBLIB и STEPLIB), описывающих личные библиотеки. Специальные имена должны иметь также операторы DD, которые описывают наборы данных, используемые транслятором и редактором связей. Эти имена следующие:

dd-*имя*	*Программа, использующая набор данных*	*Оператором* DD *описывается*
JOBLIB	Управляющая ОС ЕС	Личная библиотека, используемая в задании (оператор DD с таким именем записывается непосредственно после оператора JOB)
STEPLIB	То же	Личная библиотека, используемая в шаге задания
SYSIN	Транслятор ФОРТРАНа	Входной набор данных, содержащий исходный модуль
SYSLIN	То же	Выходной набор данных, содержащий объектный модуль
SYSPRINT	То же	Набор данных, выводимый на АЦПУ при трансляции
SYSLIN	Редактор связей	Входной набор данных, содержащий объектный модуль, используемый для создания загрузочного модуля
SYSLIB	То же	Библиотечный набор данных, используемый для автоматического вызова недостающих модулей
SYSLMOD	»	Выходной набор данных, содержащий загрузочный модуль
SYSUT1	»	Рабочий набор данных, используемый редактором связей для собственных нужд
SYSPRINT	»	Набор данных, выводимый на АЦПУ при редактировании

Для описания наборов данных, вводимых или выводимых операторами ФОРТРАНа READ или WRITE, используют операторы DD, имена которых должны иметь вид

FT*xx*F*00*1

Ссылочный номер *xx* (целое число от 1 до 99) связывает устройство, обозначенное числом *k* в операторе READ или WRITE, с устрой-

ством, указанным в операторе DD. Для наборов со ссылочными номерами 05, 06 и 07 в трансляторе принято следующее назначение внешних устройств: FT05F001 — ввода данных с перфокарт; FT06F001 — вывода данных на АЦПУ; FT07F001 — то же, на перфокарты.

Например, если в ФОРТРАН-программе встречается оператор

READ (5, 12) A, B, C

то в шаге задания, который организует выполнение загрузочного модуля, должен быть оператор DD, описывающий входной набор данных:

//FT05F001 DD *параметры*

Позиционные параметры DATA и * применяются при вводе в ЭВМ через входной поток набора данных, являющегося исходным для обрабатывающей программы (например, исходный или объектный модуль). Параметр DATA указывает, что непосредственно за оператором DD размещается набор данных.

Если вводимый набор данных не содержит перфокарт, начинающихся двумя дробными чертами, то вместо параметра DATA можно использовать параметр *. Например, для ввода исходного модуля во входном потоке помещают перфокарты

```
//SYSIN DD *
      (Исходный модуль ФОРТРАНа)
/*
```

а для ввода исходных данных по оператору READ (5, ...)

```
//FT05F001 DD *
      (Исходные данные для загрузочного модуля)
/*
```

Остановимся теперь на некоторых ключевых параметрах.

Параметр DSNAME (сокращенно DSN) определяет имя набора данных (ds-имя) и записывается следующим образом:

DSNAME = *имя-набора-данных*

Каждый постоянный набор данных обязательно должен иметь имя. В некоторых случаях при описании библиотечных наборов данных используют ds-имя с указанием имени раздела:

DSN = *имя-библиотечного-набора-данных* (*имя-раздела*)

Имя временного набора данных должно начинаться одним или двумя символами &. Отсутствие параметра DSN свидетельствует, что оператором DD описывается временный набор данных.

Рассмотрим пример. Запись

//EXML54 EXEC PGM = IEWL,*другие параметры*

//SYSLMOD DD DSN = BIBL05 (RASTR2),*другие параметры*

означает, что оператор EXEC с именем EXML54 вызывает для исполнения редактор связей (имя программы — IEWL). Сформированный им загрузочный модуль будет помещен в раздел RASTR2 (это имя программы) библиотечного набора с именем BIBL05.

Если понадобится выполнить расчеты по программе RASTR2, то обратиться к ней и ввести исходные данные можно с помощью операторов

```
//EXML541 EXEC PGM = RASTR2, другие параметры
//STEPLIB DD DSN = BIBL05, другие параметры
//FT05F001 DD *
  (Исходные данные для программы RASTR2)
/*
```

Параметр UNIT указывает устройство, предназначенное для записи нового набора данных, или устройство, с которого будет прочитан ранее созданный и хранящийся там набор данных. Формат данного параметра

UNIT = *шифр-устройства*

В качестве шифра чаще всего используется групповое имя (в частности, для НМД всех типов SYSDA, а для НМЛ и НМД всех типов SYSSQ) или типовой номер (например, 5061 для НМД объемом 29 мегабайт).

Параметр VOLUME (сокращенно VOL) используют для указания серийного (регистрационного) номера тома, содержащего набор данных. На одном устройстве, например НМД, можно устанавливать различные пакеты дисков, поэтому каждый том имеет свой уникальный серийный номер (имя). Серийный номер — это буквенно-цифровой код (до шести символов), который устанавливается вычислительным центром и сообщается пользователю.

Параметр VOLUME записывается в виде

VOLUME = SER = *серийный-номер*

Например, VOL=SER=MF2E

При записи набора данных номер тома обязательно указывают, если этот набор размещают на конкретном (например, личном) пакете дисков, а при чтении серийный номер необходим только в том случае, если набор данных сохраняемый некаталогизированный.

Параметр SPACE сообщает операционной системе, какой объем памяти следует зарезервировать на магнитном диске для размещения создаваемого набора данных. Параметр SPACE имеет следующий формат:

SPACE = (*a*, (*b*, *c*, *d*), *e*)

где *a* — единица измерения запрашиваемого объема памяти (CYL — цилиндры, TRK — дорожки, число — длина блока записи в байтах); *b* — количество единиц памяти, запрашиваемых для размещения набора данных; *c* — то же, выделяемых дополнительно (до 15 раз), если первоначального объема памяти недостаточно; *d* — количество блоков по 256 байт, выделяемых для оглавления библиотеки (только для библиотечных наборов); *e* — подпараметр RLSE, указывающий, что после завершения создания набора данных вся выделенная для него, но не использованная память освобождается.

Из перечисленных обязательны только подпараметры *a* и *b*. Если подпараметры *c* и *d* отсутствуют, то внутренние скобки не нужны.

Например, запись

SPACE = (CYL,(5,,2)),

означает, что для создаваемого библиотечного набора данных отводится пять цилиндров, дополнительная память не выделяется, для оглавления библиотеки отводится два блока по 256 байтов.

Параметр

SRACE = (1024,(20,10),RLSE)

расшифровывается так. Под набор данных первоначально отводится 20 блоков по 1024 байта. Если этого окажется недостаточно, дополнительно будет выделяться по 10 таких же блоков. Оставшаяся неиспользованной память освобождается для распределения под другие наборы данных.

Параметр DISP определяет состояние набора данных перед и после выполнения шага задания. Параметр DISP обязателен для всех входных и постоянных выходных наборов данных. Формат оператора следующий:

DISP = (*a*, *b*, *c*)

Подпараметр *a* устанавливает состояние набора данных перед выполнением шага задания. Он может иметь одно из следующих значений: NEW — набор данных создается на данном шаге задания; OLD — набор данных создан ранее: на предыдущих шагах задания, или в другом задании; MOD — к новому или старому набору данных добавляются записи; SHR — набор данных существует и может использоваться только для чтения разными заданиями в мультипрограммном режиме.

Подпараметр *b* определяет диспозицию, т. е. предписывает, что делать с набором данных после нормального завершения задания. Он имеет такие значения: DELETE — освободить память для других наборов данных; KEEP — сохранить после выполнения шага задания; PASS — передать для использования на последующих шагах задания; CATLG — поместить ds-имя постоянного набора данных в системный каталог.

Если подпараметр *b* отсутствует, система оставляет набор данных в том же состоянии, в каком он находился до выполнения шага задания: при NEW будет установлена диспозиция DELETE, а при OLD или SHR — диспозиция KEEP.

Подпараметр *c* — условная диспозиция — указывает, что делать с набором данных после аварийного завершения задания. При отсутствии подпараметра *c* в случае аварийного завершения задания временные наборы уничтожаются, а постоянные сохраняются.

Подпараметры *b* и *c* можно не указывать, тогда параметр DISP пишется без скобок.

Примеры записи параметра DISP:

DISP = (NEW,PASS)

— создается новый набор данных, который будет использован на следующих шагах задания;

DISP = SHR

— существующий набор данных может применяться как в данном, так и в других заданиях, выполняемых в режиме MVT.

Параметр DCB предназначен для описания таких характеристик, как формат и длина записи, максимальная длина блока и др. Параметр DCB обязателен только при описании создаваемого (нового) постоянного набора данных. Если в списке указывается только длина блоков, параметр DCB записывается следующим образом:

DCB = BLKSIZE = *длина-блока*

Например,

DCB = BLKSIZE = 1024

Параметр SYSOUT означает, что набор данных направляется через выходной поток. Формат этого параметра

SYSOUT= *выходной-класс*

Выходной класс кодируется буквой (или цифрой) и определяет класс выходной очереди. Обычно выходной класс A означает вывод на АЦПУ, а класс B — на карточный перфоратор. Так, оператор

//EXML55 DD SYSOUT = A

определяет для набора данных выходной класс A; при этом набор будет выведен на системное АЦПУ.

Параметр DDNAME используют, чтобы сослаться на другой оператор DD и скопировать его параметры. При ссылке на оператор с позиционным параметром * или DATA копируется не только этот параметр, но и следующий за ним входной набор данных. Формат оператора:

DDNAME = *dd-имя*

Например, если в шаге задания встречается оператор

//FT05F001 DD DDNAME = ABCDE

не содержащий никакого описания набора данных, и в этот же шаг включен оператор DD и входной набор

```
//ABCDE DD *
(Входной набор данных)
/*
```

то первый из этих операторов копирует все параметры второго (имя которого указано в параметре DDNAME) и преобразуется следующим образом:

```
//FT05F001 DD *
<Входной набор данных>
/*
```

Такая необходимость возникает, например, при составлении и использовании процедур, поскольку последние не могут содержать операторов DD с параметром * или DATA.

5.5. Составление заданий

Задание на трансляцию. Задание, содержащее только один шаг — трансляцию, выполняется для выявления синтаксических ошибок в составленной программе или при записи объектного модуля

в личную библиотеку. Это задание должно содержать операторы JOB, EXEC, а также операторы DD, описывающие используемые наборы данных.

В операторе EXEC указывают имя транслятора (IEYFORT) и требуемый объем памяти 100K. Если необходимо изменить установленные при генерации операционной системы стандартные режимы, то в операторе EXEC записывают параметр PARM. Наиболее употребляемые режимы (жирным шрифтом — по умолчанию) следующие:

SOURCE/NOSOURCE — печатать на АЦПУ исходную программу (сообщение об ошибках выдается в любом случае);

MAP/**NOMAP** — печатать имена переменных с их адресами;

LOAD/NOLOAD — поместить объектный модуль в набор данных редактора связей;

DECK/**NODECK** — перфорировать объектный модуль;

LIST/**NOLIST** — печатать объектный модуль.

Транслятор использует такие наборы данных (см. рис. 5.2): исходный модуль, вводимый с перфокарт, магнитной ленты или дисков (dd-имя SYSIN); объектный модуль, передаваемый редактору связей (dd-имя SYSLIN); набор данных, направляемый в режимах SOURCE, MAP, LIST в выходной поток для выдачи листинга (dd-имя SYSPRINT).

Пример 1. Составить задание на трансляцию исходного модуля и записать объектный модуль на диски с серийным номером тома PE55. Набору присвоить имя OBMOD11. Задание может быть записано так:

```
//EXML61   JOB MSGLEVEL = (0,0)
//         EXEC PGM = IEYFORT,REGION = 100K
//SYSIN    DD *
      (ИСХОДНЫЙ МОДУЛЬ)
/*
//SYSPRINT DD SYSOUT = A
//SYSLIN   DD DSN = OBMOD11,UNIT = SYSDA,
//            VOL = SER = PE55,SPACE = (80,(200,100),RLSE),
//            DISP = (NEW,KEEP)
//
```

Задание на редактирование. Объединение объектных модулей в единый загрузочный модуль осуществляет редактор связей — программа с именем IEWL, хранящаяся в системной библиотеке. Для редактирования требуется раздел основной памяти объемом 96K.

Режимы редактирования устанавливаются параметром PARM, подпараметры которого вызывают следующие действия:

LET — модулю приписывается атрибут «выполнимый» даже в том случае, если при редактировании были обнаружены ошибки;

NCAL — запрещается использовать аппарат автоматического вызова библиотек;

LIST — выводятся на печать управляющие предложения редактора связей;

MAP — печатается план загрузочного модуля;

XREF — печатается таблица перекрестных ссылок, включающая план загрузочного модуля.

Редактор связей использует: *набор данных основного ввода* (dd-имя SYSLIN), содержащий один или несколько объектных модулей, которые расположены на магнитных дисках, магнитных лентах или пер-

фокартах; *автоматически вызываемые библиотеки* (dd-имя SYSLIB), в которых хранятся модули стандартных функций ФОРТРАНа, подпрограммы БСП и др. (ds-имя библиотеки указывается в операторе DD, описывающем этот набор данных); *рабочий набор данных* (dd-имя SYSUT1), применяемый редактором связей для собственных нужд; *диагностический выходной набор данных* (dd-имя SYSPRINT); *выходной библиотечный набор данных* (dd-имя SYSLMOD), помещаемый во временную или постоянную библиотеку. В операторе DD, описывающем последний набор данных, кроме ds-имени указывается имя раздела, в котором будет размещен загрузочный модуль.

Пример 2. Составить задание на редактирование, включив в загрузочный модуль предварительно оттранслированные объектные модули с именами OBMOD11 и OBMOD14 (головной модуль и подпрограмма), которые содержит пакет дисков с регистрационным номером PE55. Модуль OBMOD14 каталогизирован. Загрузочный модуль поместить в раздел ZM27 существующей библиотеки с именем BIBL05, расположенной на этом же томе. В данном случае задание можно оформить так:

```
//EXML62   JOB MSGLEVEL = (2,0)
//         EXEC PGM = IEWL,REGION = 96K
//SYSLIN   DD DSN = OBMOD11,DISP = OLD,
//            UNIT = SYSDA,VOL = SER = PE55
//         DD DSN = OBMOD14,DISP = OLD
//SYSLIB   DD DSN = SYS1.FORTLIB,DISP = SHR
//SYSUT1   DD DSN = &UT1,UNIT = SYSDA,
//            SPACE = (1024,(100,10),RLSE)
//SYSPRINT DD SYSOUT = A
//SYSLMOD  DD DSN = BIBL05 (ZM27),DISP = (OLD,KEEP),
//            UNIT = SYSDA,VOL = SER = PE55
//
```

Задание на выполнение задачи. В ОС ЕС выполнению подлежат только те программы, которые находятся в библиотеках загрузочных модулей. Для выполнения программы необходимо указать ее имя (ds-имя) в параметре PGM оператора EXEC и описать используемый библиотечный набор данных в операторе DD. Если программа находится в личной библиотеке загрузочных модулей, то dd-имя должно быть JOBLIB или STEPLIB.

Пример 3. В задании на выполнение программы ZM27

```
//EXML63   JOB
//         EXEC PGM = ZM27
//STEPLIB  DD DSN = BIBL05,DISP = OLD,
//            UNIT = SYSDA,VOL = SER = PE55
//FT05F001 DD *
        (ИСХОДНЫЕ ДАННЫЕ К ПРОГРАММЕ)
/*
//FT06F001 DD SYSOUT = A
//
```

вызывается программа, хранящаяся в разделе ZM27 личной библиотеки BIBL05 (регистрационный номер тома PE55). Имя личной библиотеки и место расположения набора указаны в операторе DD с именем STEPLIB, описывающем библиотеку шага задания. Исходные данные помещаются после оператора DD с именем FT05F001. Эти данные будут использованы, когда в ФОРТРАН-программе встретится оператор ввода со ссылочным номером 5. Результаты выводятся на АЦПУ, поскольку указан выходной класс A в операторе с dd-именем FT06F001.

5.6. Каталогизированные процедуры

Составление задания — кропотливая и трудоемкая работа, требующая от программиста совершенного владения операционной системой, в частности знания характеристик исполняемых программ и параметров наборов данных. Чтобы упростить работу по составлению часто встречающихся заданий, операционная система предоставляет пользователю набор каталогизированных процедур, хранящихся в системной библиотеке SYS1.PROCLIB.

Каталогизированная процедура представляет собой последовательность управляющих операторов языка управления заданиями, организующих один или несколько шагов задания. Она содержит операторы EXEC, вызывающие к исполнению необходимые программы (например, транслятор), а также операторы DD, описывающие наборы данных, используемые этими программами.

Каталогизированная процедура может быть вызвана из библиотеки по имени, например

```
// EXEC FORTGCL
```

При выполнении задания оператор вызова каталогизированной процедуры будет заменен ее текстом, взятым из библиотеки. Очевидно, что в данном случае отпадает необходимость в записи операторов вызова программ и операторов описания наборов данных с их многочисленными параметрами и подпараметрами.

В ОС ЕС имеются каталогизированные процедуры трансляции с языка ФОРТРАН (имя процедуры FORTGC), трансляции и редактирования (FORTGCL), редактирования и исполнения (FORTGLG), трансляции, редактирования и исполнения (FORTGCLG) и др. Приведем текст[1] каталогизированной процедуры FORTGCLG:

```
//FORT     EXEC PGM = IEYFORT,REGION =100K
//SYSPRINT DD SYSOUT = A
//SYSPUNCH DD SYSOUT = B
//SYSLIN   DD DSNAME = &LOADSET,DISP = (MOD,PASS),
//            UNIT = SYSSQ,SPACE = (80, (200,100),RLSE),
//            DCB = BLKSIZE = 80
//LKED     EXEC PGM = IEWL,REGION = 96K,PARM = (XREF,LET,
//            LIST),COND = (4,LT,FORT)
//SYSLIB   DD DSNAME = SYS1.FORTLIB,DISP = SHR
//SYSLIN   DD DSNAME = &LOADSET,DISP = (OLD,DELETE)
//            DD DDNAME = SYSIN
//SYSUT1   DD UNIT = SYSDA,SPACE = (1024,(100,10),RLSE),
//            DCB = BLKSIZE = 1024,DSN = &SYSUT1
//SYSPRINT DD SYSOUT = A
//SYSLMOD  DD DSNAME = &GOSET (MAIN),DISP = (NEW,PASS),
//            UNIT = SYSDA,SPACE = (1024, (20,10,1),RLSE),
//            DCB = BLKSIZE = 1024
//GO       EXEC PGM =*.LKED.SYSLMOD,COND = ((4,LT,FORT),
//            (4,LT,LKED))
//FT05F001 DD DDNAME = SYSIN
//FT06F001 DD SYSOUT = A
//FT07F001 DD SYSOUT = B
```

[1] При генерации операционной системы некоторые операторы и их последовательность в шаге каталогизированной процедуры могут быть изменены.

Другие из перечисленных выше процедур содержат только шаг трансляции, или трансляции и редактирования, или редактирования и выполнения.

Пример 1. При выполнении задания

```
//EXML71  JOB
//        EXEC FORTGC
//SYSIN   DD *
    (ИСХОДНЫЙ МОДУЛЬ НА ФОРТРАНЕ
    (ОДИН ИЛИ НЕСКОЛЬКО))
/*
//
```

оператор // EXEC FORTGC заменяется текстом каталогизированной процедуры FORTGC. В результате задание на трансляцию принимает вид

```
//EXML71   JOB
//FORT     EXEC PGM = IEYFORT,REGION = 100K
//SYSPRINT DD SYSOUT = A
//SYSPUNCH DD SYSOUT = B
//SYSLIN   DD DSN = &LOADSET,DISP = (MOD,PASS),
//            UNIT = SYSSQ,SPACE = (80,(200,100),RLSE),
//            DCB = BLKSIZE = 80
//SYSIN    DD *
    (ИСХОДНЫЙ МОДУЛЬ НА ФОРТРАНЕ)
/*
//
```

5.7. Модификация процедур

При необходимости каталогизированную процедуру можно модифицировать, т. е. изменять, добавлять или исключать параметры в операторах EXEC и DD, а также включать в любой из ее шагов новые операторы DD. Для модификации операторов EXEC каталогизированной процедуры в операторе вызова после ее имени записывают модифицированные значения параметров:

$$p.\ s = z$$

где p — имя модифицируемого параметра; s — имя оператора EXEC каталогизированной процедуры, параметр которого модифицируется; z — новое значение параметра либо пусто (если параметр должен быть исключен).

Модифицирующие записи должны располагаться в той же последовательности, что и операторы EXEC в каталогизированной процедуре, т. е. записываются модифицированные параметры, относящиеся к первому шагу процедуры, затем ко второму шагу и т. д.

Для модификации или включения новых операторов DD следует после оператора вызова процедуры поместить такую запись:

//s.n DD *параметры*

Здесь s — имя модифицируемого шага задания каталогизированной процедуры (при изменении первого шага имя и точку можно опустить); n — dd-имя модифицируемого или включаемого оператора DD.

При включении новых операторов DD записываются все требуемые параметры, а при модификации — только те, которые необходимо добавить, изменить или исключить.

5.8. Примеры составления заданий

Пример 1. При выполнении задания на трансляцию, редактирование связей и выполнение загрузочного модуля

```
//EXML91      JOB MSGLEVEL = (1,1)
//           EXEC FORTGCLG,TIME.GO = 5
//FORT.SYSIN DD *
    ⟨ИСХОДНЫЙ МОДУЛЬ НА ФОРТРАНЕ
    (ОДИН ИЛИ НЕСКОЛЬКО)⟩
/*
//GO.SYSIN    DD *
    ⟨ИСХОДНЫЕ ДАННЫЕ РАБОЧЕЙ ПРОГРАММЫ⟩
/*
//
```

оператор EXEC FORTGCLG заменяется каталогизированной процедурой. К параметрам оператора EXEC в шаге выполнения (имя шага GO) добавляется параметр TIME=5, а в шагах трансляции (FORT) и выполнения (GO) — операторы DD с именем SYSIN и следующими за ними наборами данных: исходным модулем на ФОРТРАНе и исходными данными рабочей программы. Преобразованное задание выполняется следующим образом.

Первый шаг — трансляция. Исходный модуль находится во входном потоке после оператора с dd-именем SYSIN. Полученный в результате трансляции объектный модуль размещается как временный выходной набор данных на магнитных дисках или лентах под именем &LOADSET, на что указывает оператор DD с именем SYSLIN каталогизированной процедуры.

Второй шаг — редактирование. Входными для редактора связей являются наборы данных, описанные в операторах DD с именами SYSLIN и SYSLIB, т. е. объектный модуль с ds-именем &LOADSET и объектные модули из библиотеки автовызова SYS1.FORTLIB. После разрешения всех ссылок полученный загрузочный модуль записывается на магнитные диски в раздел MAIN библиотечного набора данных с ds-именем &GOSET.

Третий шаг — выполнение программы, имя которой указано неявно. Запись

```
// GO EXEC PGM = *.LKED. SYSLMOD
```

означает, что имя выполняемой программы указано в операторе с dd-именем SYSLMOD, который описывает набор данных в шаге с именем LKED, т. е. это программа MAIN из временной библиотеки &GOSET (отредактированный на предыдущем шаге загрузочный модуль).

Если при выполнении рабочей программы встречаются команды, реализующие оператор ввода READ (5,...)..., то исходные данные читаются из набора, описанного оператором DD:

```
// FT05F001 DD DDNAME = SYSIN
```

Запись DDNAME = SYSIN указывает, что параметры данного оператора DD должны быть скопированы с параметров оператора с dd-именем SYSIN, находящегося в этом же шаге задания. Оператор с таким именем включен в шаг GO, и, следовательно, предыдущая запись эквивалентна такой:

```
// FT05F001 DD *
⟨Исходные данные рабочей программы⟩
/*
```

Результаты работы программы выводятся на АЦПУ, что предусмотрено оператором

```
// FT06F001 DD SYSOUT = A
```

Пример 2. Составить задание на трансляцию исходного модуля и редактирование связей, используя каталогизированную процедуру FORTGCL. В ФОРТРАН-программе имеется обращение к библиотеке научных подпрограмм. Загрузочный модуль следует поместить в раздел ZM28 личной библиотеки ZHUK02, которая создается в этом же задании и размещается на пакете магнитных дисков с регистрационным номером PE55.

При составлении задания каталогизированную процедуру FORTGCL необходимо модифицировать следующим образом:

1. В шаг трансляции включить оператор DD с именем SYSIN, описывающий входной набор данных.

2. В шаг редактирования добавить оператор DD, описывающий библиотеку научных подпрограмм SYS1.SSPLIB.

3. В шаге редактирования оператор DD с именем SYSLMOD изменить таким образом, чтобы загрузочный модуль был помещен не во временную библиотеку &GOSET, а в раздел ZM28 личной библиотеки ZHUK02.

4. В этом шаге создать библиотеку ZUK02, запросить необходимую память.

Это задание оформляется так:

```
//EXML92          JOB
//                  EXEC FORTGCL
//FORT.SYSIN        DD *
      (ИСХОДНЫЙ МОДУЛЬ НА ФОРТРАНЕ)
/*
//LKED.SYSLIB       DD
//                  DD  DSN = SYS1.SSPLIB, DISP = SHR
//LKED.SYSLMOD      DD  DSN = ZHUK02(ZM28),DISP = (NEW,CATLG),
//                  UNIT = SYSDA,VOL = SER = PE55,
//                  SPACE = (CYL, (8,, 10)),DCB = BLKSIZE = 1024
//
```

Операторы DD, описывающие библиотеки SYS1.SSPLIB и SYS1.FORTLIB, сцеплены, поскольку должны иметь одно имя SYSLIB. Параметры первого из сцепленных операторов DD остаются без изменений, поэтому повторять их не обязательно. Для создаваемой библиотеки на дисках выделяется восемь цилиндров, а под оглавление отводится 10 блоков по 256 байтов. Библиотека будет каталогизирована.

Обратиться к программе ZM28 можно таким же образом, как в примере 3 подразд. 5.5.

Пример 3. Составить задание на трансляцию отлаженного исходного модуля (внешней функции или подпрограммы ФОРТРАНа) и редактирование связей. Загрузочный модуль поместить в раздел INTF2 существующей каталогизированной библиотеки с именем ZHUK02 для последующего редактирования с другими модулями.

Задание с использованием каталогизированной процедуры FORTGCL можно оформить так:

```
//EXML93          JOB MSGLEVEL = (0,0)
//                EXEC FORTGCL, PARM.LKED = NCAL
//FORT.SYSIN      DD *
      (ИСХОДНЫЙ МОДУЛЬ НА ФОРТРАНЕ)
/*
//LKED.SYSLMOD DD DSN = ZHUK02(INTF2),DISP = (OLD,KEEP)
//
```

В шаге редактирования (имя шага LKED) параметр PARM модифицируется, чтобы установился режим редактирования NCAL, в результате чего в загрузочный модуль не включаются библиотечные подпрограммы. Это способствует существенной экономии памяти на дисках. Следует отметить, что библиотеку прикладных программ целесообразнее составлять не из объектных, а из загрузочных модулей, которые занимают меньше места на дисках.

Пример 4. Составить задание на трансляцию исходного модуля, редактирование связей и решение. В загрузочный модуль включить также модули INTF2 и F1 (к этим процедурам ФОРТРАНа есть обращение из головного модуля), которые были предварительно оттранслированы, отредактированы и находятся в каталогизированной библиотеке ZHUK02 (см. пример 3).

Задание должно иметь вид:

```
//EXML94          JOB MSGLEVEL = (2,0)
//                EXEC PROC = FORTGCLG,TIME.GO = 5
//FORT.SYSIN      DD *
     (ИСХОДНЫЙ МОДУЛЬ)
```

```
/*
//LKED.SYSLIB DD
//           DD DSN = ZHUK20, DISP = OLD
//GO.SYSIN    DD *
  (ИСХОДНЫЕ  ДАННЫЕ)
/*
//
```

Если загрузочный модуль помещается в ту же библиотеку для последующего использования (например, под именем PROG21), то в задание необходимо добавить запись, модифицирующую оператор с dd-именем SYSLMOD:

```
//LKED.SYSLMOD DD DSN = ZHUK82 (PROG21), DISP = OLD
```

6. АЛГОРИТМЫ ЧИСЛЕННЫХ МЕТОДОВ

6.1. Решение математических задач методом последовательных приближений

Вычисление значения функции. Пусть необходимо вычислить значение функции $y = f(x)$. Ее можно записать в неявном виде $F(x, y) = 0$. Предположим, что $F(x, y)$ — непрерывная функция, имеющая непрерывную частную производную $F'_y(x, y) \neq 0$.

Обозначим y_n приближенное значение функции y. Согласно теореме Лагранжа

$$F(x, y_n) - F(x, y) = (y_n - y) F'_y(x, \bar{y}), \tag{6.1}$$

где $y < \bar{y} < y_n$, если $y_n > y$; $y > \bar{y} > y_n$, если $y_n < y$.

Учитывая, что $F(x, y) = 0$, из уравнения (6.1) получаем:

$$y = y_n - \frac{F(x, y_n)}{F'_y(x, \bar{y})}. \tag{6.2}$$

Значение y неизвестно, поэтому, заменив $\bar{y}$ на y_n, найдем приближенное значение y_{n+1}:

$$y_{n+1} = y_n - \frac{F(x, y_n)}{F'_y(x, y_n)}. \tag{6.3}$$

Формула (6.3) называется итерационной формулой Ньютона. Вычисления по ней выполняются при $n = 0, 1, 2, \ldots$ до тех пор, пока разность $|y_{n+1} - y_n|$ не станет меньше некоторой наперед заданной положительной величины ε. Сходимость последовательности $y_0, y_1, y_2, \ldots, y_n, y_{n+1}, \ldots$ к искомому значению y будет обеспечена, если производные $F'_y(x, y)$ и $F''_{yy}(x, y)$ в окрестности y сохраняют постоянные знаки. Начальное значение y_0 выбирается произвольно. При имеющейся возможности оценить интервал $[a; b]$, внутри которого находится y, значение y_0 следует брать внутри него.

Применение итерационной формулы Ньютона проиллюстрируем на примере вычисления функции $y = \sqrt[m]{x}$. Записываем функцию в неявном виде:

$$F(x, y) = y^m - x = 0;$$

$$F'_y(x, y) = my^{m-1};$$

$$y_{n+1} = y_n - \frac{y_n^m - x}{my_n^{m-1}} = \frac{1}{m}\left[(m-1)\,y_n + \frac{x}{y_n^{m-1}}\right]. \qquad (6.4)$$

Пусть $x = 22$, $m = 3$, т. е. необходимо вычислить функцию $y = \sqrt[3]{22}$. Выполним вычисления по итерационной формуле Ньютона с точностью до $\varepsilon = 0{,}01$. При $m = 3$ формула (6.4) перепишется так:

$$y_{n+1} = \frac{1}{3}\left(2y_n + \frac{x}{y_n^2}\right).$$

Примем $y_0 = 3$. Тогда

$$y_1 = \frac{1}{3}\left(2 \cdot 3 + \frac{22}{3}\right) = 2{,}814; \quad |y_1 - y_0| = 0{,}186 > 0{,}01;$$

$$y_2 = \frac{1}{3}\left(2 \cdot 2{,}814 + \frac{22}{2{,}814}\right) = 2{,}8021; \quad |y_2 - y_1| = 0{,}0119 > 0{,}01;$$

$$y_3 = \frac{1}{3}\left(2 \cdot 2{,}8021 + \frac{22}{2{,}8021}\right) = 2{,}802; \quad |y_3 - y_2| = 0{,}0001 < 0{,}01.$$

Следовательно, после выполнения трех итераций значение y определенно с заданной точностью.

Чтобы составить программу вычисления функции $y = f(x)$ методом последовательных приближений, представим формулу (6.3) в виде подпрограммы-функции FUNK (X, Y). Схема алгоритма вычисления указанной функции показана на рис. 6.1. Данный алгоритм реализуется программой

```
C ПРОГРАММА ВЫЧИСЛЕНИЯ ЗНАЧЕНИЯ ФУНКЦИИ Y = F (X)
C         ПРИМЕЧАНИЕ
C     В ПРОГРАММЕ ИСПОЛЬЗУЕТСЯ
C     ПОДПРОГРАММА-ФУНКЦИЯ FUNK (X,Y)
      READ 1, X, Y0, EPS
    1 FORMAT (2F8.3, F6.4)
    2 Y = FUNK (X,Y0)
      IF (ABS (Y — Y0). LE. EPS) GO TO 5
      Y0 = Y
      GO TO 2
    5 PRINT 10,Y
   10 FORMAT (' Y =', F8.3)
      STOP
      END
```

Решение нелинейных алгебраических и трансцендентных уравнений. При решении инженерных задач часто возникает необходимость вычислять корни нелинейных алгебраических и трансцендентных уравнений, точные значения которых определить невозможно. В этих случаях применяют численные методы, позволяющие определить приближенные значения корней с любой наперед заданной точностью.

Вычисляя корни уравнения, приходится решать две задачи.

1. На оси аргумента выделить такие отрезки, на каждом из которых находится только один корень. Эта задача называется *задачей отделения корней.*

2. На каждом выделенном отрезке найти точку, которая отстоит от точного корня на расстоянии, не превышающем наперед заданной

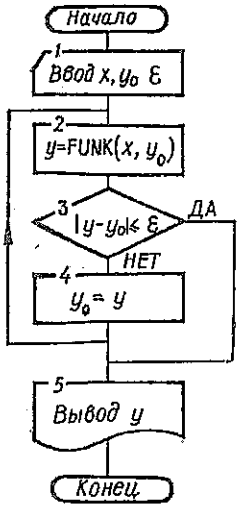
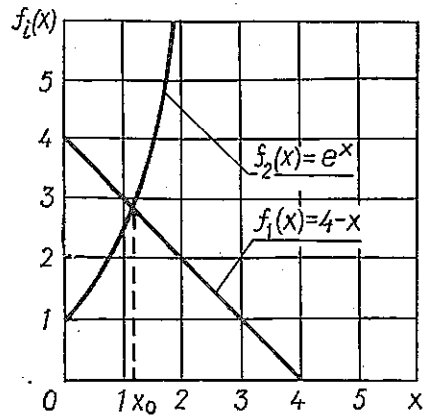


Рис. 6.1. Схема алгоритма вычисления значения функции методом последовательных приближений

Рис. 6.2. Определение интервалов отделения корней



точности. Координата данной точки принимается в качестве приближенного значения корня. Такая задача называется *задачей уточнения корня.*

Для отделения корней уравнения $f(x) = 0$ применяют графический и аналитический методы. Рассмотрим их на примерах.

Пример 1. Отделить корни трансцендентного уравнения $4 - x - e^x = 0$ графическим методом.

Представим уравнение $f(x) = 0$ следующим образом: $f_1(x) = f_2(x)$, $f_1(x) = 4 - x$, $f_2(x) = e^x$. Для функций $f_1(x)$ и $f_2(x)$ построим графики (рис. 6.2). На рисунке видно, что кривые функций $f_1(x)$ и $f_2(x)$ пересекаются в одной точке: x_0, $1 < x_0 < 2$. Это значит, что уравнение $f(x) = 4 - x - e^x = 0$ имеет один корень, который находится в интервале $[1; 2]$.

Пример 2. Отделить корни нелинейного алгебраического уравнения $x^3 + 3x^2 - 3 = 0$ аналитическим методом.

Аналитический метод отделения корней основан на следующей теореме: если функция $f(x)$ непрерывна на отрезке $[a; b]$ и принимает на его концах значения разных знаков, а производная $f'(x)$ сохраняет внутри этого отрезка постоянный знак, то внутри отрезка существует корень уравнения $f(x) = 0$, причем единственный.

Значения аргумента (точки) x, при которых функция $f(x)$ сохраняет непрерывность, а первая производная $f'(x)$ равна нулю или обращается в бесконечность, называются *критическими.* Из теоремы следует, что если в двух смежных критических точках или в критической точке и на границе определения функция принимает значения разных знаков, то между ними существует единственный корень уравнения $f(x) = 0$.

Следовательно, чтобы определить интервалы отделения корней, необходимо: найти первую производную функции и вычислить корни уравнения $f'(x) = 0$; вычислить знаки функции $f(x)$ в критических точках, а также на границах определения функции; определить интервалы, на концах которых знаки функции противоположны.

В данном примере получаем $f'(x) = 3x^2 + 6x = 0$; $x(x + 2) = 0$; $x_1 = 0$; $x_2 = -2$. Поскольку функция $f(x) = x^3 + 3x^2 - 3$ и ее первая производная определены на всей числовой оси, можно утверждать, что производная $f'(x)$ имеет постоянный знак на отрезках $(-\infty; -2)$, $[-2; 0]$, $(0; +\infty)$. Для границ отрезков найдем знаки функции

x	$-\infty$	-2	0	$+\infty$
Знак	$-$	$+$	$-$	$+$

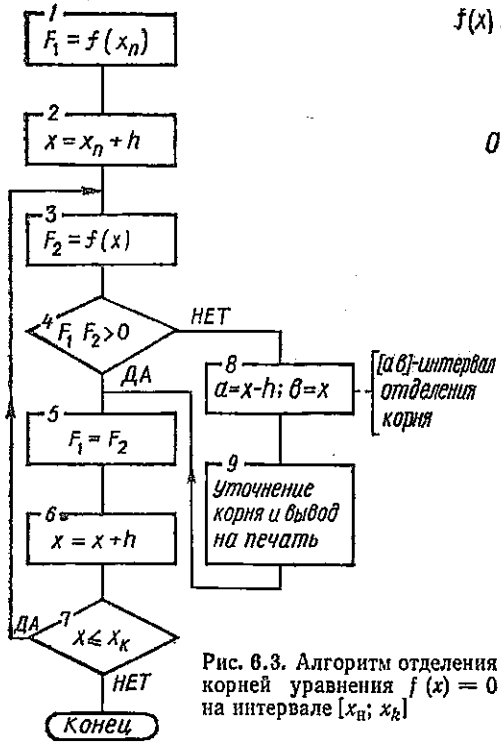


Рис. 6.3. Алгоритм отделения корней уравнения $f(x) = 0$ на интервале $[x_н; x_к]$

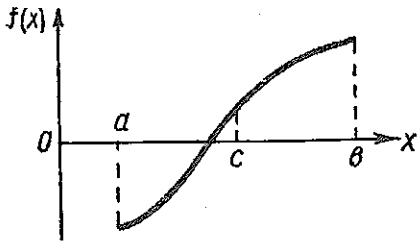


Рис. 6.4. Определение корня методом половинного деления

Так как на границах выделенных отрезков функция имеет разные знаки, внутри каждого из отрезков функция должна иметь корень. Чтобы решить задачи уточнения корня, необходимо получить конечные отрезки отделения корней. Для этого следует определить знаки функции в конечных точках слева от $x = -2$ и справа от $x = 0$. Взяв для пробы $x = -3$ и $x = 1$, получим:

x	-3	-2	0	1
Знак	$-$	$+$	$-$	$+$

Следовательно, уравнение $x^3 + 3x^2 - 3 = 0$ имеет три корня, которые находятся в интервалах $[-3; -2)$, $[-2; 0]$, $(0; 1]$.

Если возникает необходимость в отыскании корней на некотором конечном интервале $[x_н; x_к]$, последний разбивают на отрезки длиной h и вычисляют значение функции $f(x)$ в точках $x_н$, $(x_н + h)$, $(x_н + 2h)$, ..., $x_к$. Сравнивая знаки функции в двух последовательных точках, находят те пары точек, в которых знаки функции противоположны. Выделенные отрезки $[x_н + (k-1)h;\ x_н + kh]$, на границах которых функция имеет значения с противоположными знаками, являются интервалами отделения корней. Укрупненная схема алгоритма отделения корней уравнения $f(x) = 0$ на интервале $[x_н; x_к]$ показана на рис. 6.3.

Для решения задачи уточнения корней применяют методы последовательных приближений: половинного деления, хорд, метод итерации и некоторые другие. Рассмотрим алгоритмы наиболее распространенных методов.

Сущность метода *половинного деления* заключается в следующем. Пусть установлено, что на отрезке $[a; b]$ уравнение $f(x)$ содержит единственный корень. Очевидно, что на концах отрезка функция имеет значения противоположных знаков, т. е. $f(a) f(b) < 0$ (рис. 6.4). В качестве первого приближения значения корня возьмем точку c, делящую отрезок пополам: $c = (a + b)/2$.

Абсолютная погрешность Δ приближенного значения корня c не превышает $(b - a)/2$. Если она больше заданной, то в качесте нового

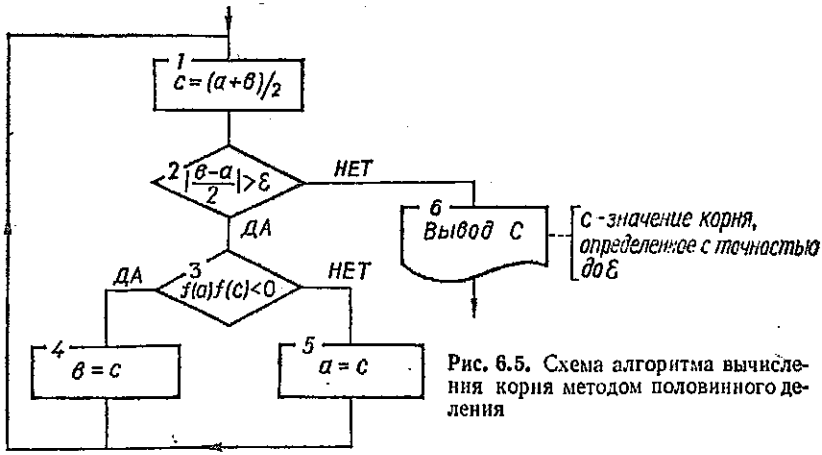


Рис. 6.5. Схема алгоритма вычисления корня методом половинного деления

выберем тот отрезок, на концах которого функция имеет значения противоположных знаков. Так, при $f(a)\,f(c) < 0$ принимаем $a' = a$, $b' = c$, а при $f(a)\,f(c) > 0 - a' = c$, $b' = b$. Далее делим пополам отрезок $[a'; b']$ и вычисляем следующее приближение корня $c' = (a' + b')/2$. Нетрудно заметить, что при этом абсолютная погрешность Δ уменьшится в два раза.

Рассмотренный вычислительный процесс повторяем до тех пор, пока абсолютная погрешность Δ не станет меньше наперед заданной положительной величины ε. Схема алгоритма метода половинного деления изображена на рис. 6.5.

Метод хорд состоит в том, что на интервале отделения корня $[a; b]$ дуга кривой $y = f(x)$ заменяется стягивающей ее хордой (рис. 6.6). Хорда пересекает ось x в точке

$$c = a - f(a)\,\frac{b-a}{f(b)-f(a)}\,.$$

Затем сравниваются знаки значений функции в точке c и на границах интервала. При $f(a)\,f(c) < 0$ в качестве нового интервала отделения корней принимаем $[a'; b']$, где $a' = a$, $b' = c$. При $f(a)\,f(c) > 0$ границами нового интервала будут $a' = c$, $b' = b$. Описанный вычислительный процесс повторяется с новым интервалом, и так до тех пор, пока функция в точке c не примет достаточно малое значение: $|f(c)| < \varepsilon$. Полученная при этом величина c соответствует приближенному значению корня.

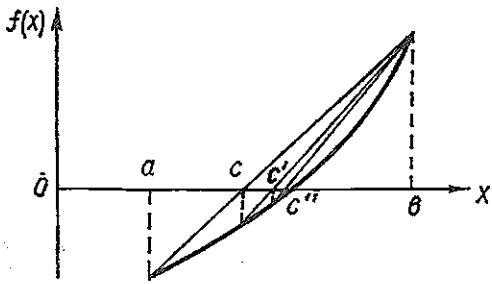


Рис. 6.6. Последовательность вычисления методом хорд

Другим критерием, определяющим прекращение вычислительного процесса, может служить сравнение модуля разности значений корня на данной и предыдущей ите-

рациях с заданной точностью. Схема алгоритма вычисления корня методом хорд показана на рис. 6.7.

Рассмотрим *метод итерации*. Пусть требуется определить вещественный корень уравнения $f(x) = 0$, заключенный в отрезке $[a; b]$. Уравнение заменим равносильным:

$$x = kf(x) + x = \varphi(x), \tag{6.5}$$

где k — постоянный коэффициент.

Для отыскания корня методом итерации необходимо, чтобы функция $\varphi(x)$ на отрезке $[a; b]$ удовлетворяла условию

$$\left|\frac{d\varphi(x)}{dx}\right| < 1. \tag{6.6}$$

Условие (6.6) эквивалентно неравенству

$$-1 < kf'(x) + 1 < 1). \tag{6.7}$$

Если производная $f'(x)$ на отрезке $[a; b]$ ограничена, всегда можно подобрать такое значение коэффициента k, чтобы выполнялось неравенство (6.7).

Например, дано уравнение $x^3 - 3x + 1 = 0$. При проверке устанавливаем, что вещественный корень этого уравнения находится на отрезке [0; 1]. Согласно уравнению (6.5) запишем:

$$\varphi(x) = k(x^3 - 3x + 1)x,$$

$$\varphi'(x) = k(3x^2 - 3) + 1.$$

На отрезке [0; 1] производная $f'(x) = 3x^2 - 3$ непрерывна и принимает значение от

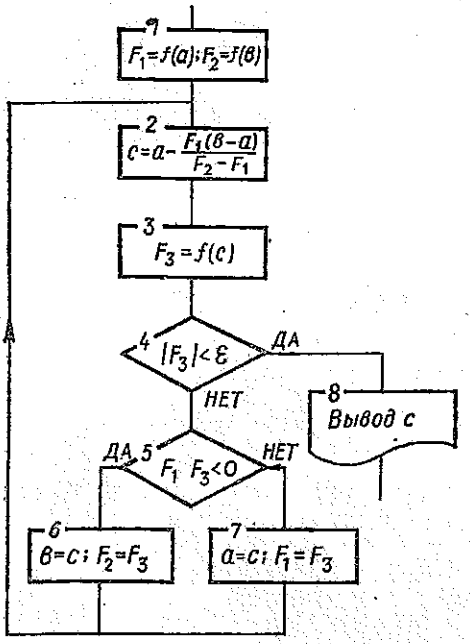


Рис. 6.7. Схема алгоритма вычисления корня методом хорд

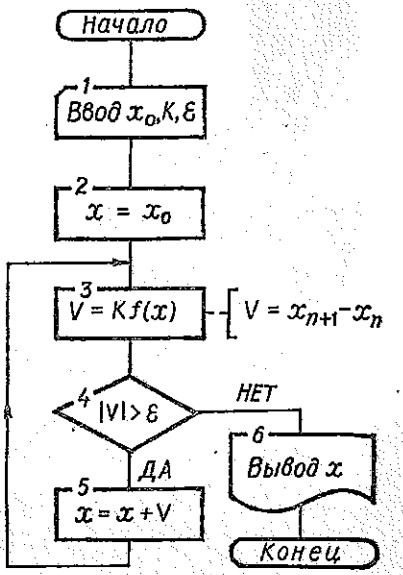


Рис. 6.8. Схема алгоритма вычисления корня методом итераций

—3 до —2. Для обеспечения условия (6.7) примем $k = 0{,}2$, в результате получим $\varphi(x) = 0{,}2x^3 + 0{,}4x + 0{,}2$.

На отрезке $[a; b]$ возьмем некоторую точку x_0 и будем считать ее приближенным значением корня уравнения $f(x) = 0$. Подставив значение x_0 в правую часть уравнения (6.5), найдем $x_1 = \varphi(x_0)$. Выполнив аналогичную подстановку значения x_1 в уравнение (6.5), получим: $x_2 = \varphi(x_1)$. Повторив этот процесс со значениями $x_2, x_3, \ldots$, приходим к рекуррентной формуле:

$$x_{n+1} = \varphi(x_n). \tag{6.8}$$

Предел последовательности $x_0, x_1, x_2, \ldots, x_n, \ldots$ является корнем уравнения $f(x) = 0$. Если задана точность ε, с которой надо определить значение корня, то вычисления по формуле (6.8) следует продолжать до тех пор, пока не будет выполнено условие $|x_{n+1} - x_n| \leqslant \varepsilon$.

Схема алгоритма вычисления корня методом итерации показана на рис. 6.8. При составлении данной схемы было учтено, что $|x_{n+1} - x_n| = |kf(x_n)|$.

Решение системы линейных уравнений. Рассмотрим систему n уравнений с n неизвестными:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1;\\ &a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2;\\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\ &a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n. \end{aligned} \tag{6.9}$$

Если система уравнений является совместной и определенной, ее всегда можно записать в таком виде, что элементы $a_{ii} \neq 0$ $(i = 1, 2, \ldots, n)$. Выделим x_1 из первого уравнения системы (6.9), x_2 — из второго и т. д. В результате получим:

$$\begin{aligned} &x_1 = \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3 - \cdots - \frac{a_{1n}}{a_{11}}x_n;\\ &x_2 = \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}}x_1 - \frac{a_{23}}{a_{22}}x_3 - \cdots - \frac{a_{2n}}{a_{22}}x_n;\\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\ &x_n = \frac{b_n}{a_{nn}} - \frac{a_{n1}}{a_{nn}}x_1 - \frac{a_{n2}}{a_{nn}}x_2 - \frac{a_{n3}}{a_{nn}}x_3 - \cdots \end{aligned} \tag{6.10}$$

Обозначим $\beta_i = b_i/a_{ii}$, $\alpha_{ij} = -a_{ij}/a_{ii}$ $(i, j = 1, 2, \ldots, n)$. Тогда систему уравнений (6.10) можно записать следующим образом:

$$\begin{aligned} &x_1 = \beta_1 + \alpha_{12}x_2 + \alpha_{13}x_3 + \cdots + \alpha_{1n}x_n\\ &x_2 = \beta_2 + \alpha_{21}x_1 + \alpha_{23}x_3 + \cdots + \alpha_{2n}x_n;\\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\ &x_n = \beta_n + \alpha_{n1}x_1 + \alpha_{n2}x_2 + \alpha_{n3}x_3 + \cdots \end{aligned} \tag{6.11}$$

Система уравнений (6.11) называется приведенной к нормальному виду. Ее можно записать в матричной форме

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \beta_n \end{pmatrix} + \begin{pmatrix} \alpha_{11}\alpha_{12} \dots \alpha_{1n} \\ \alpha_{21}\alpha_{22} \dots \alpha_{2n} \\ \dots\dots\dots \\ \alpha_{n1}\alpha_{n2} \dots \alpha_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} \tag{6.12}$$

или сокращенно

$$X = \beta + \alpha X. \tag{6.13}$$

В качестве нулевого приближения решения системы (6.11) примем столбец свободных членов, т. е.

$$\begin{Vmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \dots \\ x_n^{(0)} \end{Vmatrix} = \begin{Vmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \beta_n \end{Vmatrix} \quad \text{или} \quad X^{(0)} = \beta.$$

Подставив значение $X^{(0)}$ в формулу (6.13), получим

$$X^{(1)} = \beta + \alpha X^{(0)}, \quad X^{(2)} = \beta + \alpha X^{(1)}, \dots, X^{(k+1)} = \beta + \alpha X^{(k)}, \dots$$

Если последовательность приближений $X^{(0)}$, $X^{(1)}$, $X^{(2)}$, ... имеет предел X^*, он и будет решением системы (6.9). Следовательно, систему уравнений можно решить по рекуррентной формуле

$$X^{(k+1)} = \beta + \alpha X^{(k)} \quad (k = 0, 1, 2, \dots),$$

когда для всех элементов векторов $X^{(k)}$ и $X^{(k+1)}$ будет выполняться условие

$$|X_i^{(k+1)} - X_i^{(k)}| \leqslant \varepsilon \quad (i = 1, 2, \dots, n),$$

где ε — заданная точность.

Для сходимости итерационного процесса решения системы линейных уравнений должно удовлетворяться одно из условий:

$$\max_i \sum_{j=1}^{n} |\alpha_{ij}| < 1; \tag{6.14}$$

$$\max_j \sum_{i=1}^{n} |\alpha_{ij}| < 1; \tag{6.15}$$

$$\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |\alpha_{ij}|^2} < 1. \tag{6.16}$$

В том случае, когда определитель системы уравнений (6.9) отличается от нуля, путем линейных преобразований систему можно привести к виду, обеспечивающему сходимость итерационного процесса.

Алгоритм решения системы уравнений методом последовательных приближений сводится к выполнению следующих действий:

1. Ввести исходные данные.

2. Привести систему уравнений к нормальному виду, т. е. вычислить матрицу α и вектор β.

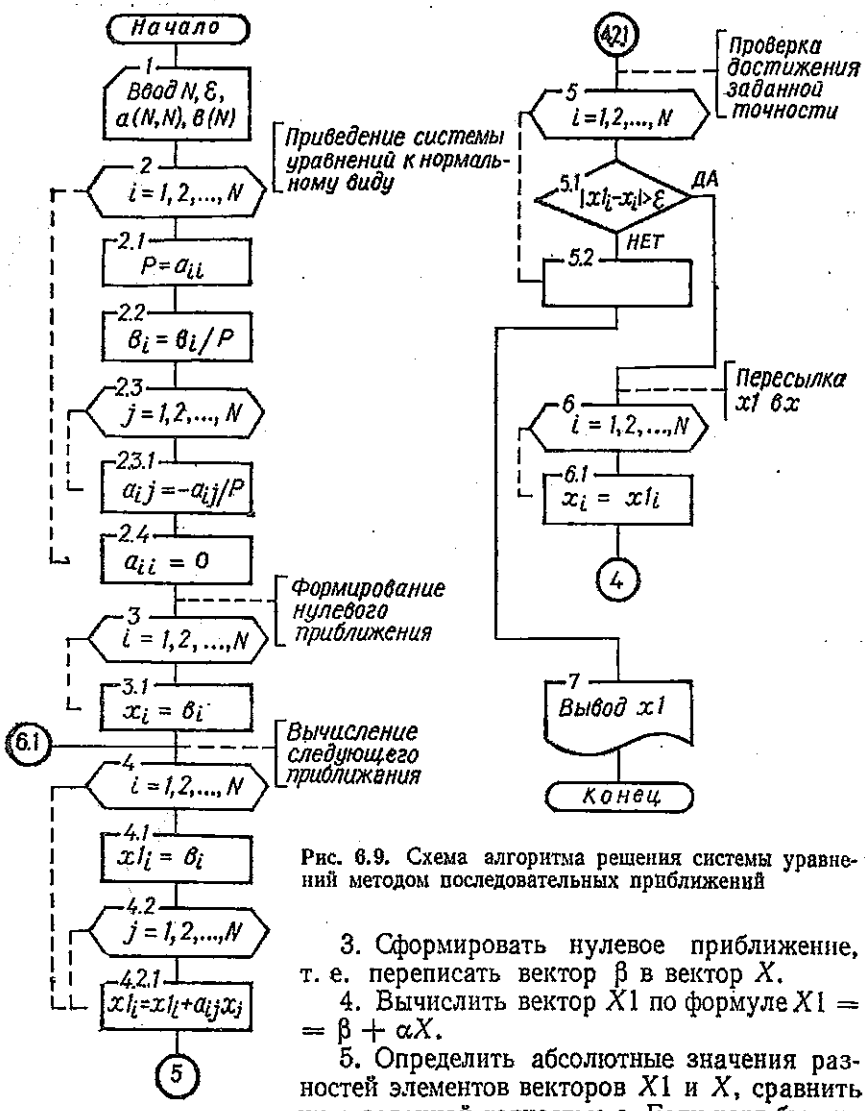


Рис. 6.9. Схема алгоритма решения системы уравнений методом последовательных приближений

3. Сформировать нулевое приближение, т. е. переписать вектор β в вектор X.

4. Вычислить вектор $X1$ по формуле $X1 = \beta + \alpha X$.

5. Определить абсолютные значения разностей элементов векторов $X1$ и X, сравнить их с заданной точностью ε. Если хотя бы для одной разности выполняется неравенство $|X1_i - X_i| > \varepsilon$, перейти к п. 6, в противном случае — к п. 7.

6. Переписать вектор $X1$ в вектор X; перейти к. п. 4.

7. Отпечатать вектор $X1$.

8. Конец.

Схема приведенного алгоритма показана на рис. 6.9. В схеме предусмотрено размещение элементов матрицы α и вектора β в ячейках памяти ЭВМ, отведенных для матрицы a и вектора b.

ФОРТРАН-программа решения системы уравнений методом последовательных приближений записывается так:

```
C ПРОГРАММА РЕШЕНИЯ СИСТЕМЫ УРАВНЕНИЙ МЕТОДОМ
C ПОСЛЕДОВАТЕЛЬНЫХ ПРИБЛИЖЕНИЙ
C
      DIMENSION A (20,20),B (20),X (20),X1 (20)
      READ 100,N,EPS
  100 FORMAT (I3,E10.3)
      PRINT 101,N,EPS
  101 FORMAT (' N =',I3,' EPS =',E10.3)
      READ 102,((A (I,J),J = 1,N),I = 1,N),(B (I),I = 1,N)
  102 FORMAT (8F10.4)
      DO 1 I = 1,N
    1 PRINT 103,(A (I,J),J = 1,N),B (I)
  103 FORMAT (10F10.4)
C
C ПРИВЕДЕНИЕ СИСТЕМЫ УРАВНЕНИЙ К НОРМАЛЬНОМУ
C ВИДУ
C
      DO 20 I = 1,N
      P = A (I,I)
      B (I) = B (I)/P
      DO 23 J = 1,N
   23 A (I,J) = —A (I,J)/P
   20 A (I,I) = 0.
C
C ФОРМИРОВАНИЕ НУЛЕВОГО ПРИБЛИЖЕНИЯ
C
      DO 30 I = 1,N
   30 X (I) = B (I)
C
C ВЫЧИСЛЕНИЕ СЛЕДУЮЩЕГО ПРИБЛИЖЕНИЯ
C
   61 DO 40 I = 1,N
      X1 (I) = B (I)
      DO 40 J = 1,N
   40 X1 (I) = X1 (I) + A (I,J)*X (J)
C
C ПРОВЕРКА ДОСТИЖЕНИЯ ЗАДАННОЙ ТОЧНОСТИ
C
      DO 50 I = 1,N
      IF (ABS (X1 (I) — X (I)).GT.EPS) GO TO 62
   50 CONTINUE
      GO TO 70
C
C ПЕРЕСЫЛКА X1 В X
C
   62 DO 60 I = 1,N
   60 X (I) = X1 (I)
      GO TO 61
C
C ВЫВОД РЕЗУЛЬТАТОВ РЕШЕНИЯ ЗАДАЧИ
C
   70 PRINT 104,(X1 (I),I = 1,N)
  104 FORMAT (' РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ'/
     *(10F10.4))
      STOP
      END
```

Диагональные элементы исходной матрицы a не равны нулю. Максимальная размерность системы $N = 20$.

Решение системы линейных уравнений методом Зейделя. В отличие от метода последовательных приближений в его модификации —

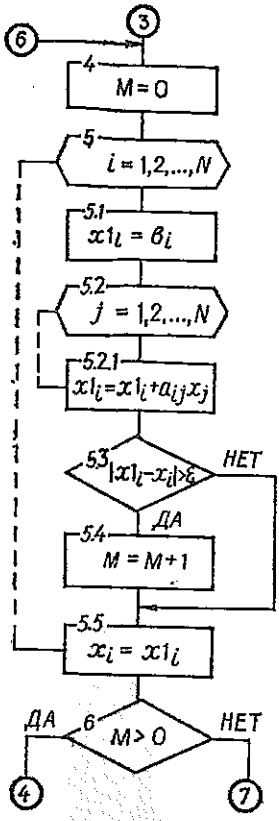


Рис. 6.10. Фрагмент схемы алгоритма решения системы уравнений, учитывающий особенности метода Зейделя

методе Зейделя — при вычислении $(k+1)$-го приближения неизвестных x_i учитываются найденные ранее $(k+1)$-е приближения неизвестных $x_1, \ldots, x_{i-1}$:

$$x_i^{(k+1)} = \beta_i + \sum_{j=1}^{i-1} \alpha_{ij} x_j^{(k+1)} + \sum_{j=1}^{n} \alpha_{ij} x_j^{(k)}. \quad (6.17)$$

Согласно этому методу блоки *5.1* и *6.1* схемы алгоритма, показанной на рис. 6.9, следует включить в состав блока *4*, поскольку передачу значения $X1_i$ в ячейку памяти ЭВМ X_i и сравнение абсолютного значения их разности с заданной точностью необходимо выполнять сразу же после вычисления $X1_i$. С этой целью вводится также счетчик M, подсчитывающий количество неизвестных, для которых заданная точность еще не достигнута. Если после очередной итерации получаем $M = 0$, значит для всех неизвестных достигнута заданная точность и вычислять следующее приближение нет необходимости.

На рис. 6.10 показан фрагмент схемы алгоритма, в котором учтены особенности метода Зейделя. Этот фрагмент заменяет блоки *4*, *5* и *6* схемы алгоритма, изображенной на рис. 6.9.

Алгоритм Зейделя, как и алгоритм метода последовательных приближений, является сходящимся при выполнении хотя бы одного из условий (6.14), (6.15), (6.16). ФОРТРАН-программа решения системы уравнений методом Зейделя записывается следующим образом:

```
C ПРОГРАММА РЕШЕНИЯ СИСТЕМЫ УРАВНЕНИЙ
C МЕТОДОМ ЗЕЙДЕЛЯ
C
      DIMENSION A (20,20),B (20),X (20),X1 (20)
      READ 100,N,EPS
  100 FORMAT (I3,E10.3)
      PRINT 101,N,EPS
  101 FORMAT (' N =',I3,' EPS =',E10.3)
      READ 102,((A (I,J),J = 1,N),I = 1,N),(B (I),I = 1,N)
  102 FORMAT (8F10.4)
      DO 1 I = 1,N
    1 PRINT 103,(A (I,J),J = 1,N),B (I)
  103 FORMAT (10F10.4)
C
C ПРИВЕДЕНИЕ СИСТЕМЫ УРАВНЕНИЙ К НОРМАЛЬНОМУ
C ВИДУ
C
      DO 20 I = 1,N
      P = A (I,I)
```

```
      B (I) = B (I)/P
      DO 23 J = 1,N
   23 A (I,J) = —A (I,J)/P
   20 A (I,I) = 0.
C
C ФОРМИРОВАНИЕ НУЛЕВОГО ПРИБЛИЖЕНИЯ
C
      DO 30 I = 1,N
   30 X (I) = B (I)
C
C ВЫЧИСЛЕНИЕ СЛЕДУЮЩЕГО ПРИБЛИЖЕНИЯ
C
   40 M = 0
      DO 50 I = 1,N
      X1 (I) = B (I)
      DO 52 J = 1,N
   52 X1 (I) = X1 (I) + A (I,J)*X (J)
C
C ПОДСЧЕТ КОЛИЧЕСТВА НЕИЗВЕСТНЫХ, ДЛЯ КОТОРЫХ
C НЕ ДОСТИГНУТА ЗАДАННАЯ ТОЧНОСТЬ EPS
C
      IF (ABS (X1 (I) — X (I)).GT.EPS) M = M + 1
C
C ПЕРЕСЫЛКА X1 В X
C
      50 X (I) = X1 (I)
C
C ПРОВЕРКА ДОСТИЖЕНИЯ ЗАДАННОЙ ТОЧНОСТИ
C ДЛЯ ВСЕХ НЕИЗВЕСТНЫХ
C
      IF (M.GT.0) GO TO 40
C
C ВЫВОД РЕЗУЛЬТАТОВ РЕШЕНИЯ ЗАДАЧИ
C
      PRINT 104,(X1 (I),I = 1,N)
  104 FORMAT (' РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ'/
     *(10F10.4))
      STOP
      END
```

С помощью данной программы была решена с точностью $\varepsilon = 10^{-3}$ система уравнений

$$6x_1 + 5x_2 = 16;$$
$$x_1 - 2x_2 = -3.$$

В результате получено:

```
N =  2  EPS = 0.100E — 02
     6.0000    5.0000   16.0000
     1.0000   —2.0000   —3.0000
РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ
     0.9998    1.9999
```

Точное решение системы уравнений: $x_1 = 1$, $x_2 = 2$.

6.2. Решение задач интерполирования функций

Постановка задачи. Задача интерполирования — одна из наиболее распространенных в горно-технических расчетах, поскольку зачастую на практике не удается записать функциональ-

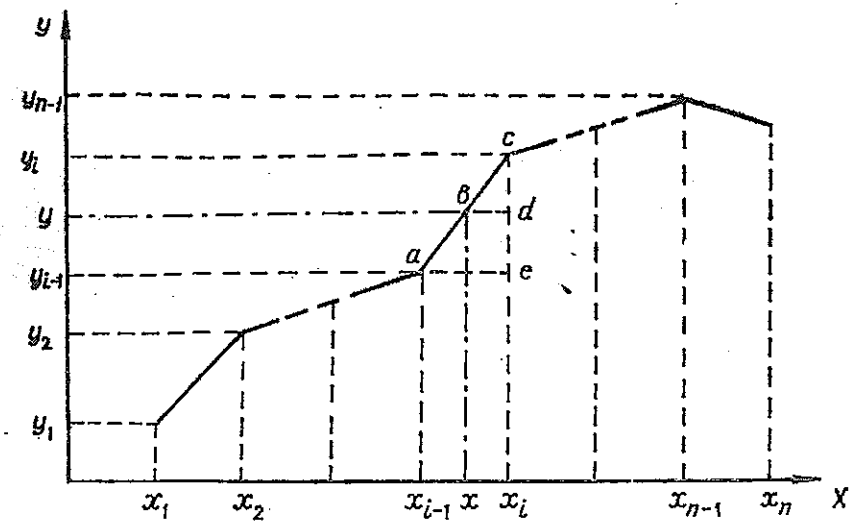


Рис. 6.11. Линейная интерполяция функции $y = f(x)$

ные зависимости аналитически и тогда они задаются узловыми точками:

Аргумент	x_1	x_2	...	x_n
Функция	y_1	y_2	...	y_n

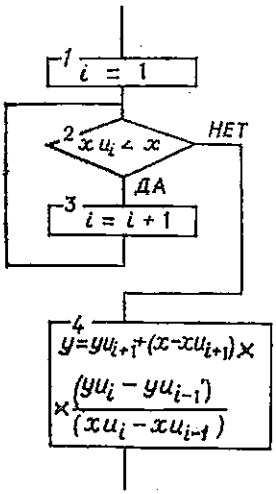


Рис. 6.12. Фрагмент схемы алгоритма решения задачи линейной интерполяции (XU — массив аргументов узловых точек; YU — массив значений узловых точек)

В данном случае $y_1 = f(x_1)$, $y_2 = f(x_2)$, ..., $y_n = f(x_n)$, $x_i < x_{i+1}$ $(i = 1, 2, ..., n - 1)$.

Областью определения функции $f(x)$ является некоторый непрерывный интервал $[x_н; x_в]$, причем $[x_1; x_n] \subseteq [x_н; x_в]$. Возникает задача определения значения $y = f(x)$ в точках $x \in [x_н; x_в]$, $x \neq x_i$ $(i = 1, 2, ..., n)$. Если $x_1 < x < x_n$, решается задача *интерполирования*, а если $x < x_1$ или $x > x_n$ — задача *экстраполирования*. Точки $x_1, x_2, ..., x_n$ называются у з л а м и интерполяции.

Задачи интерполирования решаются при горно-геометрическом анализе карьерного или шахтного поля, разведанного сеткой скважин, при выполнении расчетов по определению содержания полезного компонента и других технологических характеристик полезного ископаемого по результатам скважинного опробования на участках месторождения. Они получили широкое распространение в геодезии: рисовка рельефа при нивелировании по квадратам, прогнозы осадок реперов, интерполяция отметок в тахометрической съемке и др.

В простейших расчетах обычно ограничиваются линейной интерполяцией (рис. 6.11), которая состоит в том, что узловые точки (x_1y_1), (x_2y_2), ..., (x_ny_n) соединяют прямолинейными отрезками. При этом функция $f(x)$ аппроксимируется ломаной с прямолинейными звеньями. При линейной интерполяции для нахождения функции $f(x)$ требуются только два узла интерполяции: x_{i-1} и x_i, $x_{i-1} < x < x_i$. Из подобия треугольников *ace* и *bcd* (см. рис. 6.11) получаем

$$f(x) \approx y_{i-1} + \frac{y_i - y_{i-1}}{(x_i - x_{i-1})}(x - x_{i-1}). \qquad (6.18)$$

Знак приближенного равенства в этой формуле подчеркивает тот факт, что линейная интерполяция позволяет получить самое грубое приближение функции $f(x)$.

Фрагмент схемы алгоритма, реализующего решение задачи линейной интерполяции, показан на рис. 6.12. В данной схеме можно выделить две подзадачи: определение узлов интерполяции, между которыми находится точка x; вычисление функции $f(x)$ по формуле линейной интерполяции.

Соответствующая данному алгоритму программа записывается как модуль-подпрограмма:

```
C МОДУЛЬ-ПОДПРОГРАММА ЛИНЕЙНОЙ ИНТЕРПОЛЯЦИИ
      SUBROUTINE LIN (N,XU,YU,X,Y)
      DIMENSION XU (N),YU (N)
C
C ОПРЕДЕЛЕНИЕ УЗЛОВ ИНТЕРПОЛЯЦИИ, МЕЖДУ
C КОТОРЫМИ НАХОДИТСЯ ТОЧКА X
C
      I = 1
    2 IF (XU (I).GT.X) GO TO 4
      I = I + 1
      GO TO 2
C
C ВЫЧИСЛЕНИЕ Y ПО ФОРМУЛЕ ЛИНЕЙНОЙ ИНТЕРПОЛЯЦИИ
C
    4 Y = YU (I - 1) + (X - XU (I - 1))*(YU (I) - YU (I - 1))/
     *(XU (I) - XU (I - 1))
      RETURN
      END
```

Формальные параметры подпрограммы: N — количество узловых точек; X — аргумент интерполируемой функции; Y — искомое значение интерполируемой функции.

Очевидно, что линейная интерполяция далеко на всегда обеспечивает требуемую точность. Задачу интерполирования можно решить точнее, если в качестве интерполирующей функции $\varphi(x)$ взять полином (многочлен) степени m, $m > 1$. Такая интерполяция называется *параболической*. Отметим, что линейная интерполяция является частным случаем параболической при $m = 1$.

Параболическая интерполяция. В данном случае полином строится по $(m + 1)$ узловым точкам. Поскольку их количество $n \gg m$, получаем несколько вариантов выбора узлов интерполяции x_{i-m}, x_{i-m+1}, ..., x_{i-1}, x_i, охватывающих аргумент интерполируемой функции x:

$$x_{i-m} < x < x_i. \qquad (6.19)$$

Наилучшим из всех возможных будет вариант, при котором выполняется условие

$$\sum_{j=i-m}^{i} |x_j - x| \to \min. \qquad (6.20)$$

Иными словами, формула (6.20) определяет выбор такого индекса i, для которого сумма расстояний от задаваемых этим индексом аргументов узловых точек до аргумента интерполируемой функции будет минимальной при выполнении ограничения (6.19).

Задачу (6.19) — (6.20) можно решить следующим способом. Вычислив $\Delta_j = |x_j - x|$ $(j = \overline{1, n})$, из массива разностей $\Delta_1, \Delta_2, \ldots, \Delta_n$ выбрать $(m+1)$ минимальных разностей и из их индексов найти максимальный (i). Если $x_i < x$, индекс i увеличивать на единицу, а если $x < x_{i-m}$, уменьшить на единицу.

Схема алгоритма решения задачи выбора узлов интерполяции показа на на рис. 6.13. При поиске минимальных разностей значение $\Delta_{\min}$ поочередно сравнивается со всеми разностями. Если встречается $\Delta_j < \Delta_{\min}$, то последнее принимает значение Δ_j. Перед началом сравнения величина $\Delta_{\min}$ должна быть больше любой из разностей, поэтому ей присваивается значение $x_n - x_1$, заведомо большее максимальной разности.

Рассмотрим методику построения полинома m-й степени. Как уже отмечалось, степень полинома $P_m(x)$ определяется количеством узловых точек, по которым он строится.

Полином первой степени строится по двум узловым точкам $(x_{i-1}; y_{i-1})$, $(x_i; y_i)$:

$$P_1(x) = a_0 + a_1(x - x_{i-1}). \qquad (6.21)$$

В точке $x = x_{i-1}$ $P(x) = y_{i-1}$, откуда $a_0 = y_{i-1}$. В точке $x = x_i$ $P_1(x) = y_i$. Тогда $a_1 = (y_i - y_{i-1})/(x_i - x_{i-1})$. Таким образом, полином первой степени можно представить следующим образом:

$$P_1(x) = y_{i-1} + \frac{y_i - y_{i-1}}{(x_i - x_{i-1})}(x - x_{i-1}). \qquad (6.22)$$

Нетрудно заметить, что формула (6.22) совпадает с формулой линейной интерполяции (6.18).

По трем узловым точкам $(x_{i-2}; y_{i-2})$, $(x_{i-1}; y_{i-1})$, $(x_i; y_i)$ можно построить полином второй степени:

$$P_2(x) = a_0 + a_1(x - x_{i-2}) + a_2(x - x_{i-1})(x - x_{i-2}). \qquad (6.23)$$

В точке $x = x_{i-2}$ $P_2(x) = y_{i-2}$, значит, $a_0 = y_{i-2}$. В точке $x = x_{i-1}$ $P_2(x) = y_{i-1}$, тогда $a_1 = (y_{i-1} - y_{i-2})/(x_{i-1} - x_{i-2})$. В точке $x = x_i$ $P_2(x) = y_i$, откуда

$$a_2 = [(y_i - y_{i-2})(x_{i-1} - x_{i-2}) - (y_{i-1} - y_{i-2})(x_i - x_{i-2})]/[(x_i - x_{i-1})(x_i - x_{i-2})(x_{i-1} - x_{i-2})].$$

Произведя несложные преобразования выражений для a_1 и a_2, получаем:

$$a_1 = \frac{y_{i-1}}{x_{i-1} - x_{i-2}} + \frac{y_{i-2}}{x_{i-2} - x_{i-1}};$$

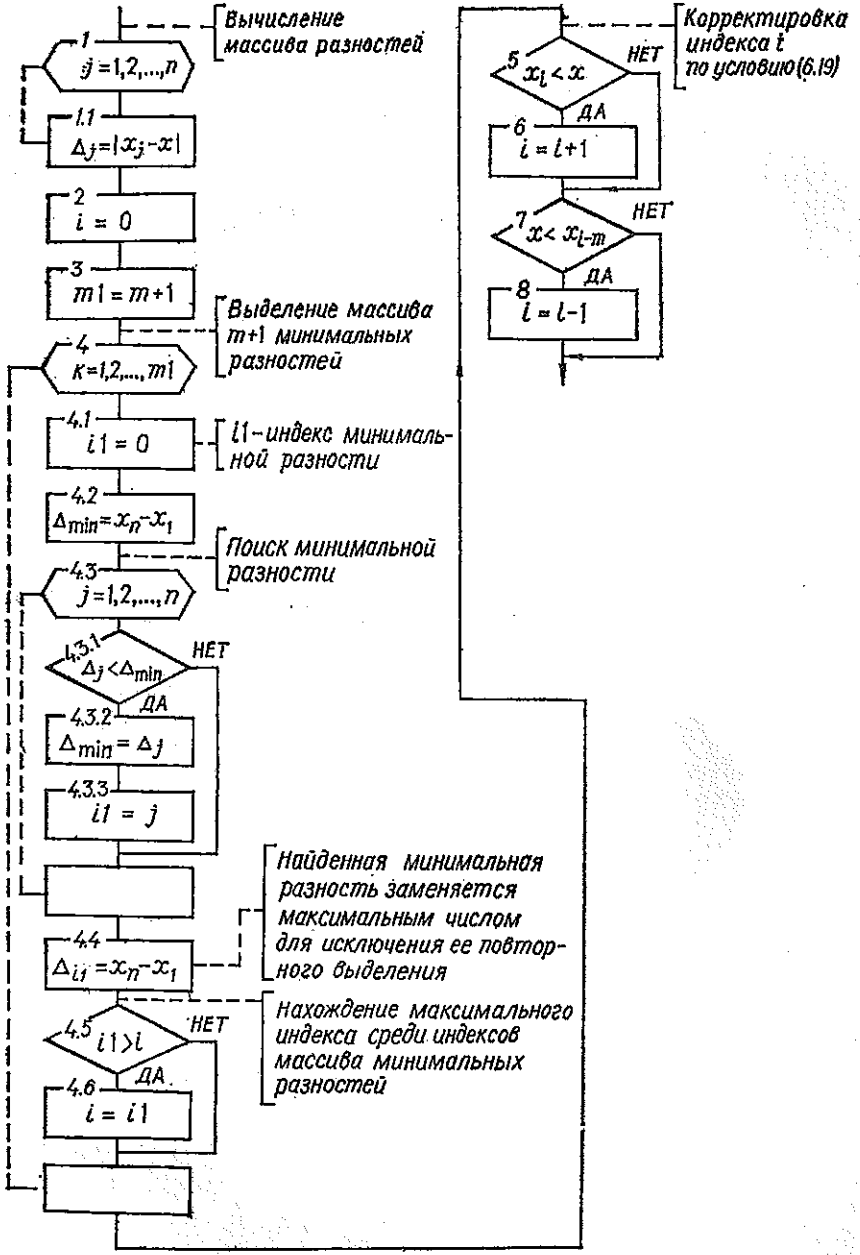


Рис. 6.13. Схема алгоритма выбора узлов интерполяции

$$a_2 = \frac{y_i}{(x_i - x_{i-1})(x_i - x_{i-2})} +$$

$$+ \frac{y_{i-1}}{(x_{i-1} - x_i)(x_{i-1} - x_{i-2})} + \frac{y_{i-2}}{(x_{i-2} - x_i)(x_{i-2} - x_{i-1})}.$$

Рассмотренная для полиномов первой и второй степени методика определения коэффициентов a_0, a_1, a_2 справедлива и для полиномов высших степеней. Так, полином m-й степени $P_m(x)$ можно представить в виде

$$P_m(x) = a_0 + a_1(x - x_{i-m}) + a_2(x - x_{i-m+1})(x - x_{i-m}) +$$
$$+ a_3(x - x_{i-m+2})(x - x_{i-m+1}) \times$$
$$\times (x - x_{i-m}) + \ldots + a_m(x - x_i) \ldots (x - x_{i-m}), \qquad (6.24)$$

где $a_0 = y_{i-m}$;

$$a_1 = \frac{y_{i-m+1}}{(x_{i-m+1} - x_{i-m})} + \frac{y_{i-m}}{(x_{i-m} - x_{i-m+1})};$$

. . . . . .

$$a_m = \frac{y_i}{(x_i - x_{i-m}) \ldots (x_i - x_{i-1})} + \frac{y_{i-1}}{(x_{i-1} - x_{i-m}) \ldots (x_{i-1} - x_i)} +$$

$$+ \ldots + \frac{y_{i-m}}{(x_{i-m} - x_{i-m+1}) \ldots (x_{i-m} - x_i)}.$$

Интервалы между узлами могут принимать существенно различные значения. Полином (6.24) называют *интерполяционным полиномом Ньютона для неравных интервалов*. Если в полиноме (6.24) сгруппировать члены с одинаковыми значениями y_k ($k = i - m, i - m + 1, \ldots, i$), получим *интерполяционный полином Лагранжа* $L_m(x)$ для последовательности узловых точек $x_{i-m}, x_{i-m+1}, \ldots, x_i$:

$$L_m(x) = y_{i-m} \frac{(x - x_{i-m+1})(x - x_{i-m+2}) \ldots (x - x_i)}{(x_{i-m} - x_{i-m+1})(x_{i-m} - x_{i-m+2}) \ldots (x_{i-m} - x_i)} +$$

$$+ y_{i-m+1} \frac{(x - x_{i-m})(x - x_{i-m+2}) \ldots (x - x_i)}{(x_{i-m+1} - x_{i-m})(x_{i-m+1} - x_{i-m+2}) \ldots (x_{i-m+1} - x_i)} + \ldots +$$

$$+ y_i \frac{(x - x_{i-m})(x - x_{i-m+1}) \ldots (x - x_{i-1})}{(x_i - x_{i-m})(x_i - x_{i-m+1}) \ldots (x_i - x_{i-1})}. \qquad (6.25)$$

Выражение (6.25) можно представить в компактной форме:

$$L_m(x) = \sum_{k=i-m}^{i} y_k \frac{\prod\limits_{j=i-m, j \neq k}^{i} (x - x_j)}{\prod\limits_{j=i-m, j \neq k}^{i} (x_k - x_j)}. \qquad (6.26)$$

При интерполяции с равными интервалами можно упростить и формулу (6.26). Обозначим

$$h = x_i - x_{i-1}, \quad q = (x - x_{i-m})/h.$$

Тогда

$$x - x_{i-m+1} = qh - h = (q-1)h,$$
$$x - x_{i-m+2} = (q-2)h,$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$x - x_i = (q-m)h.$$

Учитывая, что $x_k - x_j = (k-j)h$, получаем:

$$L_m(x) = \sum_{k=i-m}^{i} y_k \prod_{\substack{j=i-m \\ j\neq k}}^{i} \frac{(q-j+i-m)}{(k-j)}. \qquad (6.27)$$

Формулы (6.26), (6.27) позволяют выполнять параболическую интерполяцию функции полиномом m-й степени соответственно для неравных и равных интервалов.

Модуль-подпрограмма параболической интерполяции для неравных интервалов записывается так:

```
C МОДУЛЬ-ПОДПРОГРАММА ПАРАБОЛИЧЕСКОЙ ИНТЕРПОЛЯЦИИ
C ФУНКЦИИ ПОЛИНОМОМ ЛАГРАНЖА М-Й СТЕПЕНИ
C ДЛЯ НЕРАВНЫХ ИНТЕРВАЛОВ
      SUBROUTINE PARINN (N,M,I,XU,YU,X,Y)
      DIMENSION XU (N),YU (N)
C
C ВЫЧИСЛЕНИЕ НИЖНЕГО ЗНАЧЕНИЯ ИНДЕКСА К
C
      IM = I — M
      Y = 0.
C
C ВЫЧИСЛЕНИЕ СУММЫ ПО ФОРМУЛЕ (6.26)
C ЦИКЛОМ С МЕТКОЙ 10
C
      DO 10 K = IM,I
      V = XU (K)
      P = 1.
C
C ВЫЧИСЛЕНИЕ ПРОИЗВЕДЕНИЯ ПО ФОРМУЛЕ (6.26)
C ЦИКЛОМ С МЕТКОЙ 20
C
      DO 20 J = IM,I
C В ПРОИЗВЕДЕНИЕ ВКЛЮЧАЕТСЯ СОМНОЖИТЕЛЬ,
C ЕСЛИ J НЕ РАВНО К
   20 IF (J.NE.K) P = P* (X — XU (J))/(V — XU (J))
   10 Y = Y + P*YU (K)
      RETURN
      END
```

Перед обращением к подпрограмме устанавливаются узлы интерполяции $x_{i-m}, x_{i-m+1}, \ldots, x_i$. Их индексы, определяемые величинами i и m, передаются в подпрограмму через список параметров.

Формальные параметры подпрограммы: N — количество узловых точек; M — степень полинома Лагранжа; I — верхнее значение индекса суммы; XU — массив аргументов узловых точек; YU — массив функций узловых точек; X — аргумент интерполируемой функции; Y — искомое значение интерполируемой функции.

Аналогичную структуру имеет программа параболической интерполяции и при равных интервалах между узлами интерполяции.

Однако в данном случае отпадает необходимость в передаче через список формальных параметров массива аргументов узловых точек. Вместо аргумента интерполируемой функции x должна быть передана переменная q, значение которой вычисляется в вызывающем программном модуле. Операторная часть программы отличается тем, что вспомогательной переменной V не требуется и изменяется арифметическое выражение в предложении, помеченном меткой 20.

Интерполирование функции двух переменных. При расчете запасов и качественных характеристик полезных ископаемых по результатам скважинной разведки месторождения возникает необходимость в решении задачи интерполяции функции двух переменных.

Если координаты узлов интерполяции размещены в регулярной сетке и могут быть представлены матрицей

$$\begin{aligned} &(x_1y_1),\ (x_1y_2),\ \ldots,\ (x_1y_n),\\ &(x_2y_1),\ (x_2y_2),\ \ldots,\ (x_2y_n),\\ &\ldots\ldots\ldots\ldots\ldots\\ &(x_ky_1),\ (x_ky_2),\ \ldots,\ (x_ky_n), \end{aligned} \tag{6.28}$$

то алгоритм выбора узлов интерполяции аналогичен предыдущему и выполняется отдельно по каждой координате. При произвольном расположении узловых точек выбор узлов интерполяции сводится к нахождению тех m узловых точек, расстояние от которых до двумерного аргумента интерполируемой функции минимально.

Пусть функция двух переменных $V = f(x, y)$ задана значениями $V(x_l, y_l)$ на регулярной сетке узловых точек матрицы (6.28). Степень полинома по переменной x равна m_x, по переменной y — m_y, а верхние значения индексов — i_x и i_y соответственно.

Проведем интерполирование функции $V(x, y)$ по y при фиксированном значении $x = x_j$. Согласно формуле (6.26) полином Лагранжа степени m_y

$$L_{m_y}(x_j, y) = \sum_{l=i_y-m_y}^{i_y} V(x_j, y_l) \prod_{\substack{r=i_y-m_y \\ r\neq l}}^{i_y} \frac{(y-y_r)}{(y_l-y_r)}. \tag{6.29}$$

Теперь построим полином Лагранжа степени m_x для интерполирования по x, принимая в качестве значений функции в точках $x_{i_x-m_x}$, $x_{i_x-m_x+1}$, ..., x_{i_x} соответствующие значения полинома $L_{m_y}(x_j, y)$, $j = i_x - m_x, i_x - m_x+1, \ldots, i_x$, которые получены по формуле (6.29):

$$L_{m_xm_y}(x, y) = \sum_{j=i_x-m_x}^{i_x} L_{m_y}(x_j, y) \times$$

$$\times \prod_{\substack{r=i_x-m_x \\ r\neq j}}^{i_x} \frac{(x-x_r)}{(x_j-x_r)}. \tag{6.30}$$

Модуль-подпрограмма интерполирования функции двух переменных, реализующая алгоритм, заданный формулами (6.29), (6.30):

```
C МОДУЛЬ-ПОДПРОГРАММА ИНТЕРПОЛИРОВАНИЯ ФУНКЦИИ
C ДВУХ ПЕРЕМЕННЫХ ПОЛИНОМОМ ЛАГРАНЖА
C ДЛЯ СЛУЧАЯ РЕГУЛЯРНОЙ СЕТКИ УЗЛОВЫХ ТОЧЕК
C
      SUBROUTINE IN2R (N,K,IY,IX,MY,MX,X,Y,XU,YU,VU,V)
      DIMENSION XU (K),YU (N),VU (K,N)
      INTEGER R
      V = 0.
      IMX = IX — MX
      IMY = IY — MY
C
C ВЫЧИСЛЕНИЕ СУММЫ ПО ФОРМУЛЕ (6.30)
      DO 10 J = IMX,IX
C
C ВЫЧИСЛЕНИЕ ПРОИЗВЕДЕНИЯ ПО ФОРМУЛЕ (6.30)
      P1 = 1.0
      DO 20 R = IMX,IX
   20 IF (R.NE.J) P1 = P1* (X — XU (R))/(XU (J) — XU (R))
C ПРОИЗВЕДЕНИЕ ПО ФОРМУЛЕ (6.30) ВЫЧИСЛЕНО
C
C ВЫЧИСЛЕНИЕ СУММЫ ПО ФОРМУЛЕ (6.29)
      S = 0.
      DO 30 L = IMY,IY
      P2 = VU (J,L)
C
C ВЫЧИСЛЕНИЕ ПРОИЗВЕДЕНИЯ ПО ФОРМУЛЕ (6.29)
      DO 40 R = IMY,IY
   40 IF (R.NE.L) P2 = P2* (Y — YU (R))/(YU (L) — YU (R))
C ПРОИЗВЕДЕНИЕ ПО ФОРМУЛЕ (6.29) ВЫЧИСЛЕНО
   30 S = S + P2
C СУММА ПО ФОРМУЛЕ (6.29) ВЫЧИСЛЕНА
   10 V = V + S*P1
C СУММА ПО ФОРМУЛЕ (6.30) ВЫЧИСЛЕНА
      RETURN
      END
```

При обращении подпрограмме передается весь массив узлов интерполяции, а также значения степеней полинома и верхних индексов по переменным x и y, с помощью которых из матрицы (6.28) выбираются узлы, используемые в расчетных формулах.

Формальные параметры подпрограммы: N, K — количество значений соответственно аргументов y и x для узловых точек; IY, IX — верхние значения индексов по переменным y и x; MY, MX — степень полинома по переменным y и x; X, Y — значения аргументов x, y интерполируемой функции; XU, YU — массивы координат x и y узловых точек; VU — двумерный массив значений функции в узловых точках; V — искомое значение интерполируемой функции.

Рассмотрим теперь случай, когда узловые точки (x_i, y_i) имеют произвольное расположение. На рис. 6.14 показан фрагмент карты, на которую произвольно нанесены пронумерованные узловые точки. Для построения интерполяционного полинома по четырем узловым точкам выбираем узлы *51*, *75*, *93*, *76* и присваиваем им новые номера *1*, *2*, *3*, *4* соответственно. Таким образом, перед решением задачи интерполяции должен быть сформирован двумерный массив, содержащий m

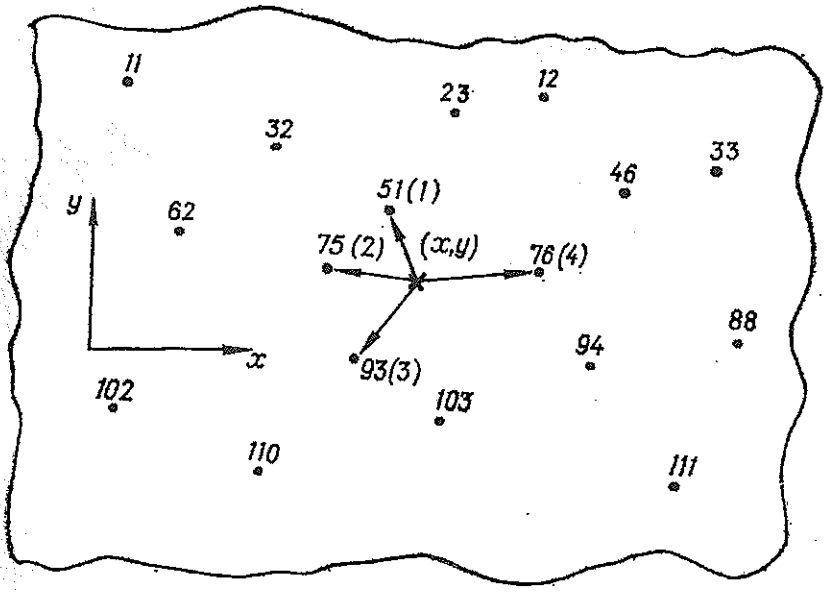


Рис. 6.14. Выбор узлов интерполяции при произвольном расположении узловых точек ($m = 4$)

узловых точек (x_1, y_1), (x_2, y_2), ..., (x_m, y_m), а также массив соответствующих им значений функции $V_1, V_2, \ldots, V_m$.

Обозначим $R_i = ((x, y), (x_i, y_i))$ вектор, направленный из точки (x_i, y_i) в точку (x, y), а $R_{ij} = ((x_i, y_i), (x_j, y_j))$ — вектор, направленный из точки (x_j, y_j) в точку (x_i, y_i).

Скалярное произведение векторов определим по формуле

$$C(R_{ij}, R_{kl}) = (x_i - x_j)(x_k - x_l) + (y_i - y_j)(y_k - y_l). \qquad (6.31)$$

С учетом принятых обозначений интерполяционный полином можно представить так:

$$V(x, y) = \sum_{i=1}^{m} V(x_i, y_i) \prod_{\substack{j=1 \\ j \neq i}}^{m} \frac{C(R_j, R_{ij})}{C(R_{ij}, R_{ij})}. \qquad (6.32)$$

Модуль-подпрограмма интерполирования функции двух переменных при произвольно расположенных узловых точках:

```
C МОДУЛЬ-ПОДПРОГРАММА ИНТЕРПОЛИРОВАНИЯ ФУНКЦИИ
C ДВУХ ПЕРЕМЕННЫХ ДЛЯ СЛУЧАЯ ПРОИЗВОЛЬНО
C РАСПОЛОЖЕННЫХ УЗЛОВ ИНТЕРПОЛЯЦИИ
      SUBROUTINE IN2P (M,XYU,VU,X,Y,V)
      DIMENSION XYU (2,M),VU (M)
      V = 0.
C
C ВЫЧИСЛЕНИЕ СУММЫ ПО ФОРМУЛЕ (6.32)
      DO 10 I = 1,M
C
C ВЫЧИСЛЕНИЕ ПРОИЗВЕДЕНИЯ ПО ФОРМУЛЕ (6.32)
      P = VU (I)
```

```
      DO 28 J = 1,M
      IF (J.EQ.I) GO TO 28
      P = P*
C ВЫЧИСЛЕНИЕ C (R (J),R (I,J))
     1 ((X — XYU (1,J))*(XYU (1,I) — XYU (1,J)) +
     1 (Y — XYU (2,J))*XYU (2,I) — XYU (2,J)))/
C ВЫЧИСЛЕНИЕ C (R (I,J),R (I,J))
     2 ((XYU (1,I) — XYU (1,J))**2 + (XYU (2,I) — XYU (2,J))**2)
   28 CONTINUE
C ПРОИЗВЕДЕНИЕ ПО ФОРМУЛЕ (6.32) ВЫЧИСЛЕНО
   18 V = V + P
C СУММА ПО ФОРМУЛЕ (6.32) ВЫЧИСЛЕНА
      RETURN
      END
```

При обращении подпрограмме передается двумерный массив координат выбранных узловых точек и соответствующих им значений функции.

Формальные параметры подпрограммы: XYU — двумерный массив координат узлов интерполяции; VU — массив зачений функции в узлах интерполяции; M — количество узлов интерполяции; X, Y — значения аргументов x, y интерполируемой функции; V — искомое значение интерполируемой функции.

6.3. Решение задач линейной алгебры методом Жордана—Гаусса

Преобразование Жордана — Гаусса. Рассмотрим систему n линейных алгебраических уравнений с n неизвестными:

$$\begin{aligned}
&a_{11}x_1 + \cdots + a_{1j}x_j + \cdots + a_{1n}x_n = b_1,\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
&a_{i1}x_1 + \cdots + a_{ij}x_j + \cdots + a_{in}x_n = b_i,\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
&a_{n1}x_1 + \cdots + a_{nj}x_j + \cdots + a_{nn}x_n = b_n.
\end{aligned} \qquad (6.33)$$

Преобразуем систему (6.33) так, чтобы переменная x_j была исключена из всех уравнений, кроме i-го. Для этого вначале разделим i-е уравнение на a_{ij}:

$$\frac{a_{i1}}{a_{ij}}x_1 + \frac{a_{i2}}{a_{ij}}x_2 + \cdots + 1\cdot x_j + \cdots + \frac{a_{in}}{a_{ij}}x_n = \frac{b_i}{a_{ij}}.$$

Теперь преобразованное i-е уравнение умножим поочередно на a_{1j}, a_{2j}, ..., $a_{(i-1)j}$, $a_{(i+1)j}$, a_{nj} и вычтем из 1, 2, ..., n-го уравнений соответственно:

$$\left(a_{11} - \frac{a_{i1}}{a_{ij}}a_{1j}\right)x_1 + \left(a_{12} - \frac{a_{i2}}{a_{ij}}a_{1j}\right)x_2 + \cdots +$$

$$+ 0\cdot x_j + \cdots - \left(a_{1n} + \frac{a_{in}}{a_{ij}}a_{1j}\right)x_n = b_1 - \frac{b_i}{a_{ij}}a_{1j},$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$\frac{a_{i1}}{a_{ij}}x_1 + \frac{a_{i2}}{a_{ij}}x_2 + \cdots + 1\cdot x_j + \cdots + \frac{a_{in}}{a_{ij}}x_n = \frac{b_j}{a_{ij}},$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$\left(a_{n1}-\frac{a_{i1}}{a_{ij}}a_{nj}\right)x_1+\left(a_{n2}-\frac{a_{i2}}{a_{ij}}a_{nj}\right)x_2+\cdots+0\cdot x_j+$$

$$+\cdots+\left(a_{nn}-\frac{a_{in}}{a_{ij}}a_{nj}\right)x_n=b_n-\frac{b_i}{a_{ij}}a_{nj}.$$

Коэффициент a_{ij}, стоящий при выделяемой неизвестной x_j в i-м уравнении, а также строку и столбец, на пересечении которых он находится, будем называть ведущими. В формулах преобразования коэффициентов, определяющих новые их значения во всех строках, кроме ведущей, видно, что преобразуемый и ведущий коэффициенты образуют вершины прямоугольника, лежащие на одной диагонали; две другие вершины этого прямоугольника образуют остальные коэффициенты. Например, при преобразовании коэффициента a_{kl} ($k \neq i$) в формуле участвуют коэффициенты

$$\begin{array}{c} \ldots\, a_{kl} \ldots a_{kj} \ldots \\ \cdots\cdots\cdots\cdots \\ \ldots\, a_{il} \ldots a_{ij} \ldots \end{array}$$

Преобразованный коэффициент определяется по выражению

$$a'_{kl}=a_{kl}-\frac{a_{il}a_{kj}}{a_{ij}}. \qquad (6.34)$$

Как видим, чтобы вычислить значение a'_{kl}, необходимо из коэффициента a_{kl} вычесть произведение коэффициентов, лежащих на противоположной диагонали прямоугольника, деленное на ведущий коэффициент. Это правило вычисления преобразованного коэффициента называется *правилом прямоугольника.*

Алгоритм преобразования системы уравнений (6.33), позволяющий исключить некоторую переменную x_j из всех уравнений, кроме i-го, можно сформулировать следующим образом: коэффициент ведущей строки разделить на ведущий коэффициент; коэффициенты остальных строк системы уравнений преобразовать по правилу прямоугольника.

Отметим, что для ведущего столбца получаем вырожденный прямоугольник ($l = j$). Применив к нему правило (6.34), получим нулевые значения коэффициентов.

Рассмотренное преобразование системы уравнений называют *преобразованием Жордана — Гаусса.* Его алгоритм используют при решении некоторых задач линейной алгебры и линейного программирования.

Решение систем линейных уравнений. Систему n линейных уравнений с n неизвестными (6.33) можно задать матрицей коэффициентов при неизвестных

$$A=\begin{Vmatrix} a_{11} \ldots a_{1l} \ldots a_{1n} \\ \cdots\cdots\cdots\cdots \\ a_{i1} \ldots a_{il} \ldots a_{in} \\ \cdots\cdots\cdots\cdots \\ a_{n1} \ldots a_{nj} \ldots a_{nn} \end{Vmatrix}$$

и столбцовыми матрицами свободных членов и неизвестных

$$B = \begin{Vmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{Vmatrix}, \quad X = \begin{Vmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{Vmatrix}$$

или записать в матричной форме:

$$AX = B. \tag{6.35}$$

Система уравнений (6.33) будет совместной и определенной, если соответствующий матрице A определитель n-го порядка не равен нулю.

Из матриц A и B можно составить расширенную матрицу системы уравнений C, которая образуется добавлением $(n+1)$-го столбца свободных членов к матрице A:

$$C = \begin{Vmatrix} a_{11} \ldots a_{1i} \ldots a_{1n} b_1 \\ \cdots\cdots\cdots\cdots \\ a_{i1} \ldots a_{ii} \ldots a_{in} b_i \\ \cdots\cdots\cdots\cdots \\ a_{n1} \ldots a_{ni} \ldots a_{nn} b_n \end{Vmatrix}.$$

Переход от матрицы C к системе уравнений производится разделением ее на матрицы A и B, подставляемые в уравнение (6.35).

Если произвести преобразование по методу Жордана — Гаусса над расширенной матрицей C, а затем перейти к системе уравнений (6.35), то результаты будут такими же, как и при преобразовании системы уравнений.

Решение системы уравнений (6.33) методом Жордана — Гаусса сводится к выполнению n преобразований Жордана — Гаусса матрицы C. При первом преобразовании ведущим будет коэффициент a_{11}, при k-м — $a_{kk}^{(k-1)}$, а при n-м — $a_{nn}^{(n-1)}$ (верхний индекс в скобках указывает на количество преобразований, выполняемых над этим коэффициентом до использования его в качестве ведущего), в результате чего получим матрицу

$$C^{(n)} = \begin{Vmatrix} 1 \ldots 0 \ldots 0 \; b_1^{(n)} \\ \cdots\cdots\cdots \\ 0 \ldots 1 \ldots 0 \; b_i^{(n)} \\ \cdots\cdots\cdots \\ 0 \ldots 0 \ldots 1 \; b_n^{(n)} \end{Vmatrix}.$$

Отсюда следует, что решение системы уравнений

$$X = B^{(n)} = \begin{pmatrix} b_1^{(n)} \\ b_i^{(n)} \\ b_n^{(n)} \end{pmatrix}. \tag{6.36}$$

Перед k-м преобразованием матрица C имеет вид

$$C^{(k-1)} = \begin{Vmatrix} 1 \ldots a_{1k}^{(k-1)} a_{1k+1}^{(k-1)} \ldots a_{1n}^{(k-1)} b_1^{(k-1)} \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ 0 \ldots a_{kk}^{(k-1)} a_{kk+1}^{(k-1)} \ldots a_{kn}^{(k-1)} b_k^{(k-1)} \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ 0 \ldots a_{nk}^{(k-1)} a_{nk+1}^{(k-1)} \ldots a_{nn}^{(k-1)} b_n^{(k-1)} \end{Vmatrix},$$

а после его выполнения с ведущим коэффициентом $a_{kk}^{(k-1)}$

$$C^{(k)} = \begin{Vmatrix} 1 \ldots 0 \; a_{1k+1}^{(k)} \ldots a_{1n}^{(k)} b_1^{(k)} \\ \cdots\cdots\cdots\cdots\cdots \\ 0 \ldots 1 \; a_{kk+1}^{(k)} \ldots a_{kn}^{(k)} b_k^{(k)} \\ \cdots\cdots\cdots\cdots\cdots \\ 0 \ldots 0 \; a_{nk+1}^{(k)} \ldots a_{nn}^{(k)} b_n^{(k)} \end{Vmatrix}.$$

Здесь

$$a_{kj}^{(k)} = a_{kj}^{(k-1)} / a_{kk}^{(k-1)}; \tag{6.37}$$

$$b_k^{(k)} = b_k^{(k-1)} / a_{kk}^{(k-1)}; \tag{6.38}$$

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k)}; \tag{6.39}$$

$$b_i^{(k)} = b_i^{(k-1)} - a_{ik}^{(k-1)} b_k^{(k)}; \tag{6.40}$$

$$i = 1, \ldots, k-1, k+1, \ldots, n; \quad j = k, k+1, \ldots, n.$$

Формулы (6.37), (6.38) соответствуют первому шагу алгоритма преобразования Жордана — Гаусса, а (6.39), (6.40) — второму. Отметим, что при выполнении k-го преобразования матрицы C первые $(k-1)$ столбцов не рассматриваются ($j \geqslant k$), поскольку применение указанного алгоритма не изменяет их (в соответствующем прямоугольнике в одной строке с ведущим коэффициентом находится коэффициент, равный нулю). Таким образом, решение системы уравнений (6.33) методом Жордана — Гаусса сводится к вычислению расширенной матрицы C по формулам (6.37) — (6.40) с $k = 1, 2, \ldots, n$.

При выполнении $(k+1)$-го и последующих преобразований Жордана — Гаусса k-й столбец матрицы C не участвует в вычислениях, поэтому для сокращения последних целесообразно на k-м этапе начинать преобразования с $(k+1)$-го столбца ($j = (k+1), (k+2), \ldots, n$).

Системы линейных уравнений во многих представляющих практический интерес случаях могут иметь *разреженные* матрицы, т. е. матрицы, в которых большинство коэффициентов равно нулю. Очевидно, не исключен вариант, что один из них может быть выбран в качестве ведущего. Поскольку деление на нуль невозможно, в k-м ведущем столбце отыщем ненулевой элемент в строках $(k+1), (k+2), \ldots, n$. Если $a_{lk} \neq 0$, поменяем местами k-ю и l-ю строки, после чего получим $a_{kk} \neq 0$. Так как от перестановки уравнений системы значения неизвестных не изменяются, выполненные действия не повлияют на результаты вычислений. Когда анализ ведущего столбца не позволяет выявить ненулевой элемент, это свидетельствует о несовместности или неопределенности данной системы уравнений.

Схема алгоритма решения системы n линейных алгебраических

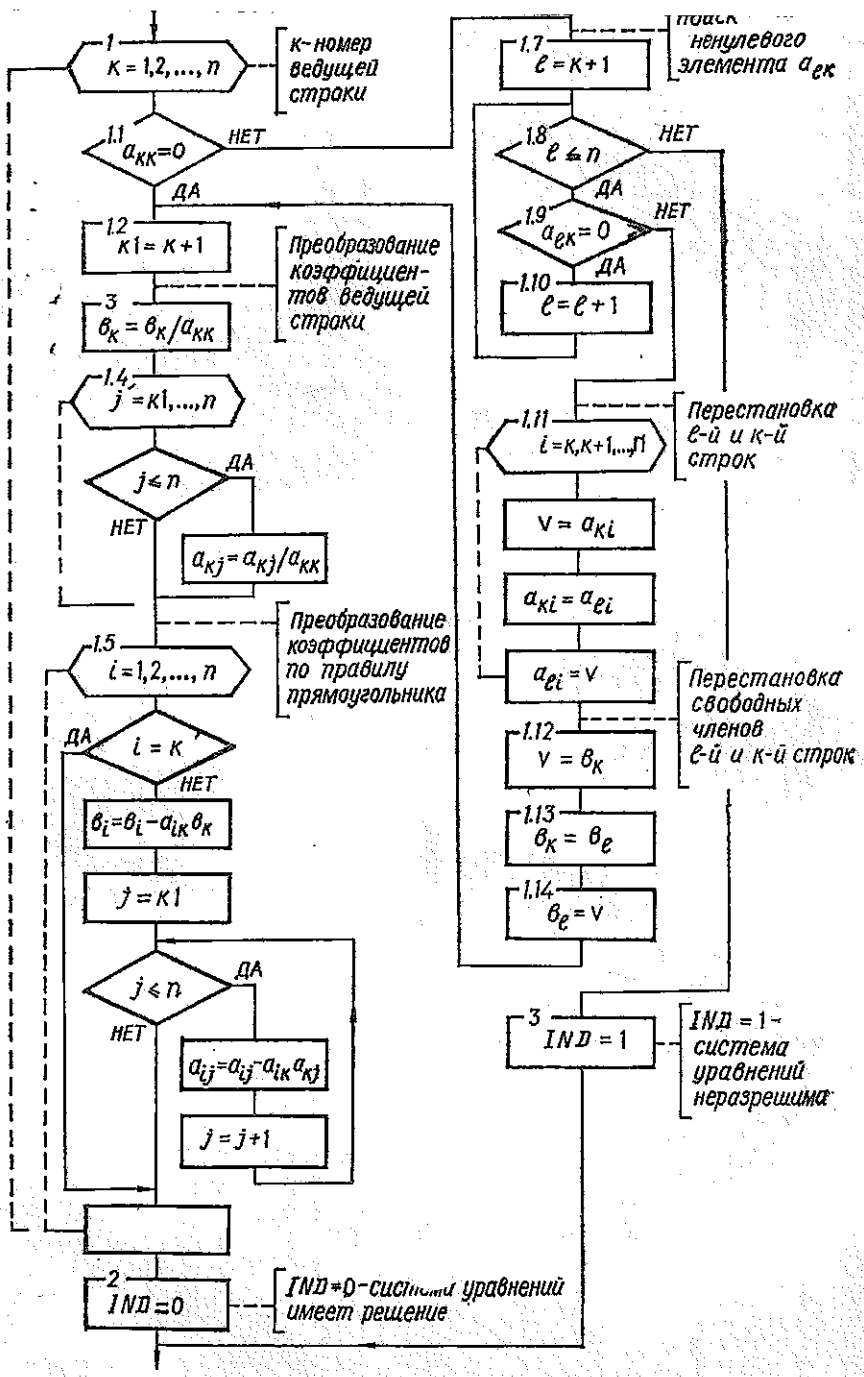


Рис. 6.15. Схема алгоритма решения системы линейных уравнений методом Жордана — Гаусса

уравнений с n неизвестными показана на рис. 6.15. В схеме алгоритма можно выделить две части: блоки *1.2—1.5* выполняют непосредственно преобразование Жордана — Гаусса по формулам (6.37) — (6.40); блоки *1.7—1.14* осуществляют алгоритмы поиска ненулевого элемента в ведущем столбце и перестановку соответствующих строк матрицы C. После получения решения системы уравнений, которое образуется в столбцовой матрице B, переменной IND, идентифицирующей признак полученного результата, присваивается нулевое значение. Если система уравнений не имеет единственного решения (в ведущем столбце не найден ненулевой элемент), принимается IND = 1. Таким образом, по значению переменной IND можно судить о том, какой результат получен.

Модуль-подпрограмма решения системы линейных алгебраических уравнений методом Жордана — Гаусса, составленная в соответствии с приведенной схемой алгоритма, записывается следующим образом:

```
C МОДУЛЬ-ПОДПРОГРАММА РЕШЕНИЯ СИСТЕМЫ ЛИНЕЙНЫХ
C АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ МЕТОДОМ ЖОРДАНА — ГАУССА
      SUBROUTINE RSUMJG (N,A,B,IND)
      DIMENSION A (N,N),B (N)
      DO 1 K = 1,N
C K — НОМЕР ВЕДУЩЕЙ СТРОКИ
      IF (A (K,K).NE.0.0) GO TO 12
C ПОИСК НЕНУЛЕВОГО ЭЛЕМЕНТА В ВЕДУЩЕМ СТОЛБЦЕ
      L = K + 1
   18 IF (L.GT.N) GO TO 3
      IF (A (L,K).NE.0.0) GO TO 111
      L = L + 1
      GOTO 18
C ПЕРЕСТАНОВКА L-Й И K-Й СТРОК
  111 DO 1113 I = K,N
      V = A (K,I)
      A (K,I) = A (L,I)
 1113 A (L,I) = V
C ПЕРЕСТАНОВКА СВОБОДНЫХ ЧЛЕНОВ L-Й И K-Й СТРОК
      V = B (K)
      B (K) = B (L)
      B (L) = V
   12 K1 = K + 1
C ПРЕОБРАЗОВАНИЕ КОЭФФИЦИЕНТОВ ВЕДУЩЕЙ СТРОКИ
      B (K) = B (K)/A (K,K)
      DO 14 J = K1,N
   14 IF (J.LE.N) A (K,J) = A (K,J)/A (K,K)
C ПРЕОБРАЗОВАНИЕ КОЭФФИЦИЕНТОВ ПО ПРАВИЛУ
C ПРЯМОУГОЛЬНИКА
      DO 1 I = 1,N
      IF (I.EQ.K) GO TO 1
      B (I) = B (I) — A (I,K)*B (K)
      J = K1
   15 IF (J.GT.N) GO TO 1
      A (I,J) = A (I,J) — A (I,K)*A (K,J)
      J = J + 1
      GO TO 15
    1 CONTINUE
      IND = 0
      GO TO 4
    3 IND = 1
    4 RETURN
      END
```

Формальные параметры подпрограммы: N — размерность системы уравнений; A — матрица коэффициентов при неизвестных; B — столбцовая матрица свободных членов. Следует иметь в виду, что после выполнения подпрограммы исходные матрицы A и B не сохраняются.

6.4. Метод наименьших квадратов

Основные понятия. Аппроксимация экспериментальных данных функциями, зависящими от исследуемых аргументов, — одна из наиболее распространенных в практике горно-технических расчетов. Математическим аппаратом, обеспечивающим решение этой задачи, является метод наименьших квадратов.

Предположим, необходимо установить функциональную зависимость продолжительности рейса автосамосвала от расстояния откатки. Результаты хронометражных наблюдений приведены в табл. 6.1.

Экспериментальные точки были нанесены на координатную сетку (t, r) (рис. 6.16). На рисунке видно, что продолжительность рейса t прямо пропорциональна расстоянию откатки r. Следовательно, искомую функциональную зависимость можно представить как $t = b_1 + b_2 r$.

Тот факт, что экспериментальные точки не ложатся на прямую линию, объясняется влиянием на продолжительность рейсов неучтенных случайных факторов.

Если принять прямую линию на рис. 6.16 за искомую, то коэффициенты b_1 и b_2 определить нетрудно. Однако проведенная таким образом линия может не отражать фактическую зависимость величин t и r. Действительно, с тем же успехом на этом рисунке можно про-

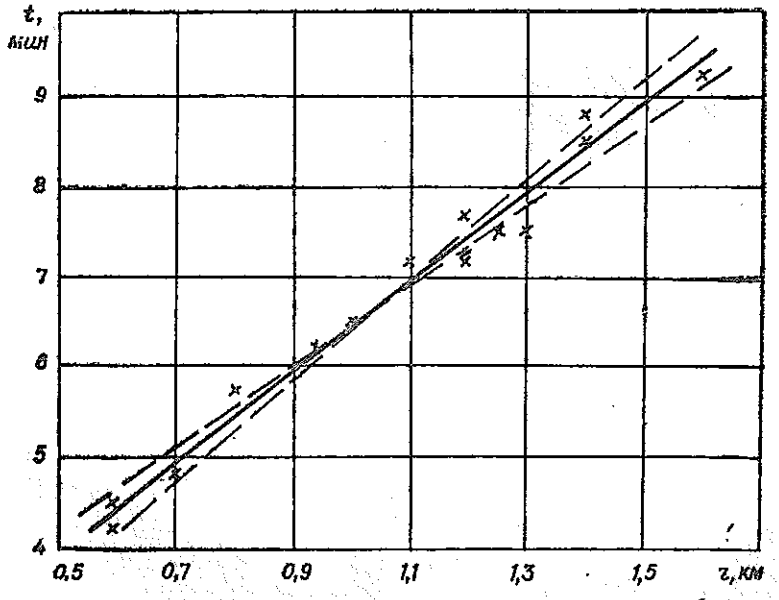


Рис. 6.16. Экспериментальная зависимость продолжительности рейса автосамосвала от расстояния откатки

Таблица 6.1. **Результаты хронометражных наблюдений за продолжительностью рейсов автосамосвалов**

r, км	t, мин	r, км	t, мин
0,60	4,5	1,20	7,2
0,60	4,2	1,20	7,6
0,70	4,8	1,25	7,5
0,80	5,7	1,30	7,8
0,95	6,1	1,40	8,5
1,00	6,5	1,40	8,8
1,10	7,1	1,60	9,3

вести прямую линию выше или ниже, под бо́льшим или меньшим углом (см. штриховые линии). Следовательно, для получения искомой зависимости необходим критерий, применение которого позволит найти единственное решение.

В методе наименьших квадратов таким критерием является следующий: сумма квадратов расстояний от эмпирических точек до соответствующих им точек на линии, изображающей искомую функциональную зависимость, должна быть минимальной. Для рассматриваемого примера его можно записать так:

$$D = \sum_{i=1}^{14} (b_1 + b_2 r_i - t_i)^2 \to \min.$$

Очевидно, что $D = f(b_1, b_2)$. Чтобы найти минимум D, решим систему уравнений:

$$\frac{\partial D}{\partial b_1} = 2 \sum_{i=1}^{14} (b_1 + b_2 r_i - t_i) = 0,$$

$$\frac{\partial D}{\partial b_2} = 2 \sum_{i=1}^{14} r_i (b_1 + b_2 r_i - t_i) = 0.$$

Выполнив несложные преобразования, получим:

$$b_1 + b_2 \frac{\sum_{i=1}^{14} r_i}{14} = \frac{\sum_{i=1}^{14} t_i}{14},$$

$$b_1 \frac{\sum_{i=1}^{14} r_i}{14} + b_2 \frac{\sum_{i=1}^{14} r_i^2}{14} = \frac{\sum_{i=1}^{14} t_i r_i}{14}.$$

Это система двух линейных алгебраических уравнений относительно двух неизвестных b_1 и b_2. Обозначив

$$S_1 = \sum_{i=1}^{14} r_i/14, \; S_2 = \sum_{i=1}^{14} t_i/14,$$

$$S_3 = \sum_{i=1}^{14} r_i^2/14, \; S_4 = \sum_{i=1}^{14} r_i t_i/14,$$

для искомых коэффициентов b_1 и b_2 запишем: $b_1 = (S_2 S_3 - S_1 S_4)/(S_3 - S_1^2)$, $b_2 = ((S_4 - S_1 S_2)/(S_3 - S_1^2)$. Подставив данные примера $S_1 = 1{,}0785$, $S_2 = 6{,}8286$, $S_3 = 1{,}25536$, $S_4 = 7{,}83$, получим: $b_1 = 1{,}385$, $b_2 = 5{,}04$.

Таким образом, окончательно зависимость продолжительности рейса автосамосвала от расстояния откатки имеет вид $t = 1{,}385 + 5{,}04r$.

Алгоритм аппроксимации экспериментальных данных функциями методом наименьших квадратов. Рассмотренный метод построения функциональной зависимости от одного аргумента по эмпирическим данным можно распространить на случай с несколькими аргументами. Сформулируем метод в общем виде.

Пусть экспериментальные данные $y_l = y(x_{1l}, \ldots, x_{ml})$ $(l = 1, 2, \ldots, n)$, имеющие случайный характер, необходимо аппроксимировать линейной функцией

$$y = b_1 + b_2x_1 + b_3x_2 + \cdots + b_{m+1}x_m. \tag{6.41}$$

Значения параметров уравнения $b_1, b_2, \ldots, b_{m+1}$ определим методом наименьших квадратов из условия минимальности величины

$$D = \sum_{l=1}^{n} (b_1 + b_2x_{1l} + b_3x_{2l} + \cdots + b_{m+1}x_{ml} - y_l)^2. \tag{6.42}$$

Вычислив частные производные от критерия D по параметрам b_1, b_2, ..., b_{m+1} и приравняв их нулю, получим систему уравнений:

$$\begin{gathered}
b_1 + b_2\frac{\sum_{l=1}^{n} x_{1l}}{n} + b_3\frac{\sum_{l=1}^{n} x_{2l}}{n} + \cdots + b_{m+1}\frac{\sum_{l=1}^{n} x_{ml}}{n} = \frac{\sum_{l=1}^{n} y_l}{n};\\
b_1\frac{\sum_{l=1}^{n} x_{1l}}{n} + b_2\frac{\sum_{l=1}^{n} x_{1l}^2}{n} + b_3\frac{\sum_{l=1}^{n} x_{1l}x_{2l}}{n} + \cdots + b_{m+1}\frac{\sum_{l=1}^{n} x_{1l}x_{ml}}{n} =\\
= \frac{\sum_{l=1}^{n} x_{1l}y_l}{n};\\
b_1\frac{\sum_{l=1}^{n} x_{2l}}{n} + b_2\frac{\sum_{l=1}^{n} x_{1l}x_{2l}}{n} + b_3\frac{\sum_{l=1}^{n} x_{2l}^2}{n} + \cdots +\\
+ b_{m+1}\frac{\sum_{l=1}^{n} x_{2l}x_{ml}}{n} = \frac{\sum_{l=1}^{n} x_{2l}y_l}{n};\\
\cdots\cdots\cdots\cdots\cdots\cdots\\
b_1\frac{\sum_{l=1}^{n} x_{ml}}{n} + b_2\frac{\sum_{l=1}^{n} x_{1l}x_{ml}}{n} + b_3\frac{\sum_{l=1}^{n} x_{2l}x_{ml}}{n} + \cdots +\\
+ b_{m+1}\frac{\sum_{l=1}^{n} x_{ml}^2}{n} = \frac{\sum_{l=1}^{n} x_{ml}y_l}{n}.
\end{gathered} \tag{6.43}$$

Система линейных уравнений (6.43) содержит $(m + 1)$ уравнений относительно $(m + 1)$ неизвестных — искомых параметров b_1, b_2, ..., b_{m+1}. По структуре матрицы коэффициентов при неизвестных

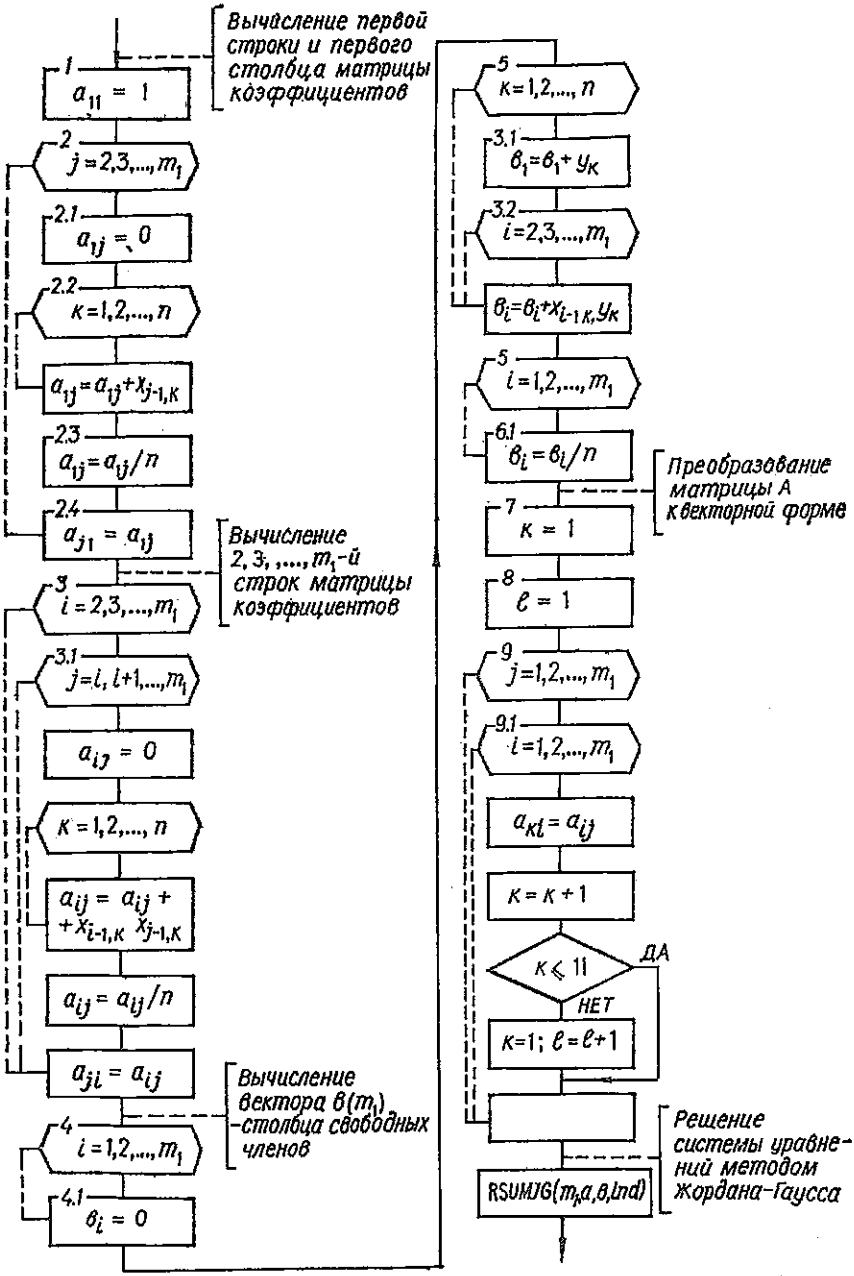


Рис. 6.17. Схема алгоритма вычисления параметров линейной функции, аппроксимирующей экспериментальные данные

видно, что диагональные элементы матрицы равны средним значениям квадратов аргументов, остальные элементы симметричны относительно диагонали.

Чтобы получить алгоритм построения и решения системы уравнений (6.43), запишем исходные данные в матричной форме. Обозначим

$$X = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{pmatrix}, \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{m+1} \end{pmatrix}.$$

Матрицу A коэффициентов при неизвестных системы (6.43) можно определить по формуле

$$A = \frac{1}{n} XX^{\mathrm{T}},$$

где X^{T} — транспонированная матрица X.

Столбец свободных членов равен $\frac{1}{n} XY$.

Система уравнений в матричной форме имеет вид

$$AB = \frac{1}{n} XY. \qquad (6.44)$$

Решить эту систему уравнений можно с использованием подпрограммы вычисления обратной матрицы ($B = A^{-1} \frac{1}{n} XY$) или методом Жордана — Гаусса с помощью подпрограммы RSUMJG.

На рис. 6.17 показана схема алгоритма вычисления параметров линейной функции, аппроксимирующей экспериментальные данные. При формировании матрицы коэффициентов учтена ее симметричность относительно главной диагонали. Поскольку в подпрограмме RSUMJG матрица коэффициентов при неизвестных имеет регулируемые размеры, перед обращением к ней матрица A преобразуется к векторной форме. При этом предполагается, что максимальное количество аргументов, которыми можно оперировать, не должно превышать десяти.

Модуль-подпрограмма вычисления параметров линейного эмпирического уравнения методом наименьших квадратов записывается так:

```
C МОДУЛЬ-ПОДПРОГРАММА ВЫЧИСЛЕНИЯ ПАРАМЕТРОВ
C ЛИНЕЙНОГО ЭМПИРИЧЕСКОГО УРАВНЕНИЯ
C МЕТОДОМ НАИМЕНЬШИХ КВАДРАТОВ
C              ПРИМЕЧАНИЕ
C       ДЛЯ ВЫПОЛНЕНИЯ ДАННОЙ ПОДПРОГРАММЫ
C       ТРЕБУЕТСЯ МОДУЛЬ-ПОДПРОГРАММА RSUMJG
C
      SUBROUTINE MNK (N,M,M1,X,Y,B,IND)
C
C N — КОЛИЧЕСТВО ЭКСПЕРИМЕНТАЛЬНЫХ ТОЧЕК
C M — КОЛИЧЕСТВО АРГУМЕНТОВ
C M1 — РАЗМЕРНОСТЬ СИСТЕМЫ УРАВНЕНИЙ (M1 = M + 1)
C X — МАТРИЦА ЗНАЧЕНИЙ АРГУМЕНТОВ
C Y — ВЕКТОР ЗНАЧЕНИЙ ФУНКЦИИ
C B — ВЕКТОР ПАРАМЕТРОВ ЭМПИРИЧЕСКОГО УРАВНЕНИЯ
C     (ИСПОЛЬЗУЕТСЯ ДЛЯ ХРАНЕНИЯ СТОЛБЦА СВОБОДНЫХ ЧЛЕНОВ)
C IND — ИНДИКАТОР РЕШЕНИЯ (IND = 0 — РЕШЕНИЕ ПОЛУЧЕНО,
```

```
C      IND = 1 — ЗАДАЧА НЕРАЗРЕШИМА)
C
       DIMENSION X (M,N),Y (N),A (11,11),B (M1)
C
C ВЫЧИСЛЕНИЕ 1-Й СТРОКИ И 1-ГО СТОЛБЦА
C МАТРИЦЫ КОЭФФИЦИЕНТОВ
C
       IF (M.LE.10) GO TO 1
       IND = 1
       GO TO 100
     1 A (1,1) = 1.0
       DO 2 J = 2,M1
         A (1,J) = 0.
         DO 22 K = 1,N
    22 A (1,J) = A (1,J) + X (J — 1,K)
         A (1,J) = A (1,J)/N
     2 A (J,1) = A (1,J)
C
C ВЫЧИСЛЕНИЕ 2-Й, 3-Й, ..., M1-Й СТРОК МАТРИЦЫ
C      КОЭФФИЦИЕНТОВ
C
       DO 3 I = 2,M1
       DO 3 J = 1,M1
       A (I,J) = 0.
         DO 32 K = 1,N
    32 A (I,J) = A (I,J) + X (I — 1,K)*X (J — 1,K)
       A (I,J) = A (I,J)/N
     3 A (J,I) = A (I,J)
C
C ВЫЧИСЛЕНИЕ ВЕКТОРА B — СТОЛБЦА СВОБОДНЫХ ЧЛЕНОВ
C
       DO 4 I = 1,M1
     4 B (I) = 0.
       DO 5 K = 1,N
       B (I) = B (I) + Y (K)
       DO 5 I = 2,M1
     5 B (I) = B (I) + X (I — 1,K) *Y (K)
       DO 6 I = 1,M1
     6 B (I) = B (I)/N
C
C ПРЕОБРАЗОВАНИЕ МАТРИЦЫ A В ВЕКТОРНУЮ ФОРМУ
C
       K = 1
       L = 1
       DO 10 J = 1,M1
       DO 10 I = 1,M1
       A (K,L) = A (I,J)
       K = K + 1
       IF (K.LE.11) GO TO 10
       K = 1
       L = L + 1
    10 CONTINUE
C
C РЕШЕНИЕ СИСТЕМЫ УРАВНЕНИЙ МЕТОДОМ ЖОРДАНА — ГАУССА
C
       CALL RSUMJG (M1,A,B,IND)
   100 RETURN
       END
```

Количество параметров эмпирического уравнения не должно превышать 11 (количество аргументов 10); это ограничение определяется размерами матрицы A.

Формальные параметры подпрограммы: N, M — количество соответственно экспериментальных точек и аргументов; M1 — количество параметров эмпирического уравнения (M1 = M + 1); X (M, N) — матрица значений аргументов; Y (N) — вектор значений функции; B (M1) — вектор параметров эмпирического уравнения.

При обращении к подпрограмме матрица X должна быть преобразована к векторной форме, поскольку она имеет регулируемые размеры.

Рассмотренный алгоритм применим и для нелинейной аппроксимации. Действительно, заменой переменных многие нелинейные зависимости можно привести к линейному виду. Например, приняв для уравнения $y = a + bx + cx^2$ обозначения $x = x_1$, $x^2 = x_2$, после подстановки получим: $y = a + bx_1 + cx_2$.

Таким образом, алгоритмы аппроксимации эмпирических данных различными функциональными зависимостями отличаются друг от друга только способами формирования матрицы X. Например, для параболической зависимости $y = b_1 + b_2x + b_3x^2$ матрица X запишется следующим образом:

$$X = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ x_1 & x_2 & x_3 & \dots & x_n \\ x_1^2 & x_2^2 & x_3^2 & \dots & x_n^2 \end{pmatrix}.$$

6.5. Метод статистических испытаний

Метод статистических испытаний, носящий также название метода Монте-Карло, нашел применение для решения различных вычислительных задач и моделирования процессов. Он используется в тех случаях, когда аналитическое описание и решение задач затруднительно или даже невозможно. В основе метода статистических испытаний лежит получение случайных чисел, с помощью которых моделируется статистический эксперимент и регистрируются его числовые характеристики, являющиеся числовыми оценками решения задачи.

Для пояснения сущности метода статистических испытаний рассмотрим пример. Пусть необходимо вычислить интеграл $S = \int_0^1 f(x)\,dx$, где $f(x) \leqslant 1$. Численное значение интеграла равно площади геометрической фигуры с основанием 1, ограниченной сверху графиком подынтегральной функции $f(x)$ (рис. 6.18).

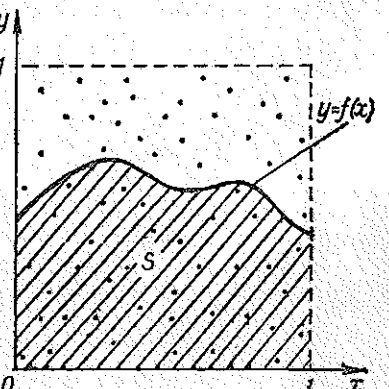


Рис. 6.18. К вычислению интеграла методом статистических испытаний

Организуем вычислительный процесс. Каждой паре случайных чисел φ и ξ, равномерно распределенных в интервале [0; 1], поставим в соответствие точку в квадрате $0 \leqslant x \leqslant 1$, $0 \leqslant y \leqslant 1$, причем координаты i-й точки $x_i = \varphi_i$ и $y_i = \xi_i$. Если описанным

способом получить достаточно большое количество случайных точек, они будут равномерно распределены по площади квадрата (см. рис. 6.18).

Вероятность того, что случайно размещенная в квадрате точка попала в область s под кривой $y = f(x)$, численно равна площади области s, т. е. значению интеграла. Точка с координатами $(\varphi; \xi)$ попадает в область s, если случайные числа φ и ξ удовлетворяют неравенству

$$\xi \leqslant f(\varphi). \tag{6.45}$$

Пусть в квадрате $0 \leqslant x \leqslant 1$, $0 \leqslant y \leqslant 1$ получено N случайных точек, из них N_1 точек удовлетворяют неравенству (6.45). Тогда вероятность попадания точки в область s равна N_1/N, следовательно, интеграл $S = N_1/N$.

Значение интеграла можно вычислить и другим способом метода статистических испытаний. Получим N случайных чисел $\varphi_1, \varphi_2, \ldots, \varphi_N$, равномерно распределенных в интервале $[0; 1]$, и образуем случайные числа $\xi_i = f(\varphi_i)$, $i = 1, 2, \ldots, N$. Можно показать, что математическое ожидание случайной величины ξ численно равно значению интеграла: $M(\xi) = \int_0^1 f(x)\,dx$. Для приближенного вычисления математического ожидания определим среднее арифметическое случайных чисел ξ $(i = 1, 2, \ldots, N)$:

$$S = M(\xi) \approx \frac{\sum_{i=1}^{N} \xi_i}{N} \approx \frac{\sum_{i=1}^{N} f(\varphi_i)}{N}.$$

Метод статистических испытаний позволяет также производить имитационное моделирование процессов по известным законам распределения их характеристик. Имитационное моделирование дает возможность анализировать процессы без проведения натурного эксперимента, поскольку имитирует его в цифровой форме.

Датчики случайных чисел. Для реализации метода статистических испытаний на ЭВМ используются датчики случайных чисел — программы, воспроизводящие случайные числа.

Случайные числа в современных ЭВМ можно получить с помощью физических датчиков, например генераторов шума, основанных на дробовом эффекте в электронных приборах. Получаемые таким образом числа называют *чисто случайными,* поскольку воспроизвести повторно их последовательность невозможно.

При отсутствии в ЭВМ физических датчиков применяют программные, которые позволяют получать последовательности случайных чисел, воспроизводимые при последующих исполнениях программы. Эти числа называют *псевдослучайными.*

Один из первых таких датчиков был предложен Дж. фон Нейманом. Алгоритм датчика заключается в следующем. Берем некоторое произвольное случайное число, например 0,857941. Возводим его в квадрат: $0{,}857941^2 = 0{,}739154587081$. Отбросив в этом числе три старших и три младших разряда, получаем второе случайное число 0,154587. Повторив с ним ту же процедуру, находим третье случайное число и т. д.

С помощью программного датчика можно получить через некоторый период такое случайное число, которое ранее уже было, причем все последующие числа тоже повторяются:

$$\xi_1, \xi_2, \ldots, \xi_p, \xi_{p+1}, \ldots, \xi_n, \xi_{n+1} = \xi_{p+1}, \xi_{n+2} = \xi_{p+2}, \ldots, \xi_{2n-p} = \xi_n, \ldots$$

Первые n чисел последовательности различны и обладают статистическими свойствами, близкими к статистическим свойствам чисто случайных чисел. Они образуют участок апериодичности длиной n. Количество повторяющихся чисел $n - p$ называют *длиной периода*. При решении задач методом статистических испытаний общее количество используемых случайных чисел не должно превышать длины апериодичности. Существуют программные датчики, позволяющие получать последовательности случайных чисел с длиной апериодичности 10^6—10^{12}.

В практике статистического моделирования нашел применение датчик Дэвиса, который дает возможность получить последовательность равномерно распределенных случайных чисел в интервале [0; 1]. Датчик Дэвиса реализует следующая модуль-подпрограмма:

```
C МОДУЛЬ-ПОДПРОГРАММА ДАТЧИКА РАВНОМЕРНО РАСПРЕДЕЛЕННЫХ
C     СЛУЧАЙНЫХ ЧИСЕЛ
      SUBROUTINE DRR (U1,U2,R)
      T = U1 + U2
      U1 = U2
      IF (T.GE.4.) T = T — 4.
      U2 = T
      R = T/4.
      RETURN
      END
```

Формальные параметры подпрограммы: U1, U2 — вспомогательные переменные; R — переменная, которой соответствует фактический параметр — генерируемое случайное счисло. В модуле, из которого производится обращение к подпрограмме, вспомогательным переменным должны быть присвоены начальные значения: U1 = = 3,14159265, U2 = 0,542101887.

При многократном обращении к подпрограмме DRR находим последовательность случайных чисел R_i ($i = 1, 2, 3, \ldots$), равномерно распределенных в интервале [0; 1] с математическим ожиданием $M(R) = 0{,}5$. С помощью датчика Дэвиса можно получать последовательность случайных чисел, подчиненных любому закону распределения.

Для последовательности нормально распределенных случайных чисел G_j ($j = 1, 2, 3, \ldots$) с математическим ожиданием $M(G) = 0$ и средним квадратическим отклонением $\sigma(G) = 1$ числа R_i группируются по m в порядке их получения. Из каждой группы, содержащей m R-чисел, образуется новое случайное число:

$$G = \sqrt{\frac{12}{m}}\left(\sum_{i=1}^{m} R_i - \frac{m}{2}\right). \qquad (6.46)$$

Согласно центральной предельной теореме теории вероятностей при достаточно большом значении m числа G_j подчиняются нормальному закону распределения. Практически можно принять $m = 10$—15.

Чтобы получить нормально распределенные случайные числа с заданным математическим ожиданием $M(G) \neq 0$ и средним квадратическим отклонением $\sigma(G) \neq 1$, величины G пересчитывают по формуле

$$G' = M(G) + \sigma(G)\, G. \qquad (6.47)$$

Датчик нормально распределенных случайных чисел реализует следующая подпрограмма:

```
C МОДУЛЬ-ПОДПРОГРАММА ДАТЧИКА НОРМАЛЬНО
C       РАСПРЕДЕЛЕННЫХ СЛУЧАЙНЫХ ЧИСЕЛ
        SUBROUTINE DNR (U1, U2, Gϑ, S, G)
        G = ϑ.
        DO 1 I = 1,12
        T = U1 + U2
        U1 = U2
        IF = (T. GE. 4.) T = T — 4.
        U2 = T
      1 G = G + T
        G = Gϑ + S * (G/4,s 6.)
        RETURN
        END
```

В подпрограмме используется алгоритм Дэвиса. Для упрощения вычислений по формуле (6.46) принято $m = 12$. Формальные параметры подпрограммы U1, U2 — вспомогательные переменные; Gϑ — математическое ожидание; S — среднее квадратическое отклонение; G — генерируемое случайное число.

Вычисленные по формулам (6.46), (6.47) нормально распределенные случайные числа являются некоррелированными. Вместе с тем для моделирования, например, распределения качества полезного ископаемого в добычных забоях необходимо получить коррелированную случайную последовательность с коэффициентом корреляции двух смежных случайных значений, равным r. Коррелированные центрированные случайные числа H_j, подчиненные нормальному закону распределения с $M(H) = 0$ и любым наперед заданным значением $\sigma(H)$, вычисляют по формуле

$$H_j = rH_{j-1} + \sigma(H)\sqrt{1 - r^2}\, G_j, \qquad (6.48)$$

где $H_0 = 0$ $(j = 1, 2, 3, \ldots)$.

Для получения чисел H_j' с математическим ожиданием $M(H) \neq 0$ величины H_j пересчитывают по формуле

$$H_j' = M(H) + H_j. \qquad (6.49)$$

Чтобы имитировать на ЭВМ процессы загрузки и разгрузки транспортных емкостей, продолжительность наработки на отказ и восстановления технологического оборудования, необходимо генерировать случайные числа, подчиненные экспоненциальному закону распределения. Эти числа можно вычислить с помощью датчика Дэвиса путем преобразования их по формуле

$$E_i = -\ln(R_i)/\lambda, \qquad (6.50)$$

где E_i — экспоненциально распределенные случайные числа; R_i — равномерно распределенные случайные числа, полученные с помощью подпрограммы DRR; λ — интенсивность процесса (число событий в единицу времени).

Для некоторых технологических процессов используются законы распределения, которые не поддаются точному аналитическому описанию. В таких ситуациях для имитации случайных процессов можно воспользоваться эмпирической кумулятивной кривой, по которой с помощью датчика равномерно распределенных случайных чисел формируется эмпирическое распределение.

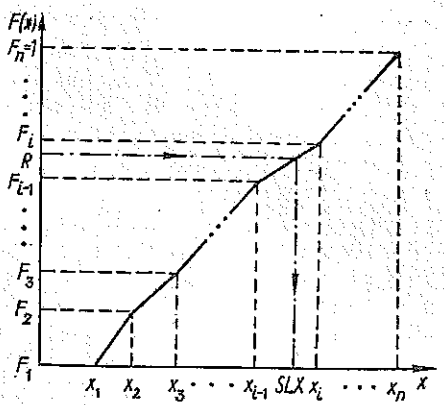


Рис. 6.19. Определение случайного числа по кумуляте эмпирической функции распределения

Пусть эмпирически установленный закон распределения задан кумулятивной кривой (рис. 6.19). Приравняем значение эмпирической функции распределения $F(x)$ случайному числу R и найдем по кумулятивной кривой соответствующее ему значение SLX на оси x. Выполняя эту процедуру многократно с последовательностью равномерно распределенных чисел R, получим последовательность случайных чисел SLX, подчиненных распределению, определяемому эмпирически установленной кумулятивной кривой.

Алгоритм нахождения случайного числа SLX по кумулятивной кривой и случайному числу R представлен модулем-подпрограммой SL:

```
C МОДУЛЬ-ПОДПРОГРАММА ДАТЧИКА СЛУЧАЙНЫХ ЧИСЕЛ,
C     ПОДЧИНЕННЫХ ЭМПИРИЧЕСКОЙ ФУНКЦИИ РАСПРЕДЕЛЕНИЯ
      SUBROUTINE SL (U1,U2,N,F,X,SLX)
      DIMENSION F (N),X (N)
      CALL DRR (U1,U2,R)
      I = 2
    1 IF (F (I).GE.R) GO TO 2
      I = I + 1
      GO TO 1
    2 SLX = (X (I) — X (I — 1)) * (R — F (I — 1))/(F (I) — F (I — 1)) + X (I —
     * 1)
      RETURN
      END
```

Для получения числа R предусмотрено обращение к подпрограмме DRR. Формальные параметры подпрограммы: U1, U2 — вспомогательные переменные; N — количество узловых точек кумулятивной кривой; X (M), F (N) — массивы координат узловых точек кумуляты; SLX — генерируемое случайное число.

Подпрограмму SL целесообразно использовать при имитации на ЭВМ процессов разгрузки транспортных средств, взвешивания их с грузом, распределения гранулометрического состава взорванной горной массы и некоторых других распределений.

Моделирование на ЭВМ процессов массового обслуживания. Объектом изучения теории массового обслуживания является обслуживающая система, в которую в случайные моменты времени поступают

заявки. Продолжительность их обработки также величина случайная. Зная законы распределения, которым подчиняются поток заявок и продолжительность обслуживания, можно рассчитать параметры системы.

Типичными моделями указанных систем являются модели, описывающие процессы погрузки и разгрузки транспорта, профилактики и ремонта технологического оборудования, поступления горной массы в аккумулирующие емкости и др. Для наиболее простых случаев (последовательное обслуживание, пуассоновский поток заявок и др.) разработаны аналитические методы решения задач; во всех остальных случаях используется метод статистических испытаний. На ЭВМ имитируется работа обслуживающей системы по тем же вероятностным законам, что и в натуре, регистрируются параметры процесса, статистическая обработка которых дает решение задачи.

Остановимся на общих принципах моделирования.

Пусть система массового обслуживания содержит прибор, который может последовательно обрабатывать поступающие заявки. Интервалы времени t между поступлениями заявок распределены в соответствии с функцией распределения $F_1(t)$. Продолжительность обслуживания заявок подчиняется закону распределения, определяемому функцией распределения $F_2(t)$. Следует установить закон распределения количества заявок, находящихся в очереди.

Укрупненная схема алгоритма моделирования работы системы массового обслуживания показана на рис. 6.20. Исходными данными для моделирования являются параметры функций распределения $F_1(t)$, $F_2(t)$ и продолжительность имитации работы системы T_k (блок *1*). Анализ состояния системы производится с шагом времени, равным единице.

В блоке *2* формируются начальные значения параметров состояния системы: текущее время моделирования принимается равным нулю; в очереди и в обслуживающем приборе заявки отсутствуют; первая заявка поступает на обработку в момент времени $t = 0$.

Блоки *3—10* моделируют поступление заявок на обслуживание. Если текущее время моделирования не достигло значения, равного моменту времени поступления заявки (блок *3*), моделирование не производится. При поступлении заявки проверяется занятость обслуживающего прибора (блок *4*). Если прибор занят, заявка становится в очередь (блок *8*), а если свободен, она поступает на обслуживание (блок *5*). С помощью датчика Дэвиса вырабатывается случайная продолжительность обслуживания Δt (блок *6*), после чего определяется момент времени завершения процесса (блок *7*).

В блоке *9* формируется случайная продолжительность времени до поступления следующей заявки и определяется момент времени ее поступления (блок *10*).

Блоки *11—17* моделируют обслуживание заявки прибором. Если прибор свободен (блок *11*) или обслуживание не завершено (блок *12*), моделирование не производится. По окончании обслуживания проверяется наличие заявок в очереди (блок *13*). В зависимости от состояния очереди прибор либо освобождается (блок *15*), либо принимает очередную заявку (блок *14*). При поступлении новой заявки в обслуживающий прибор формируется случайная продолжительность вре-

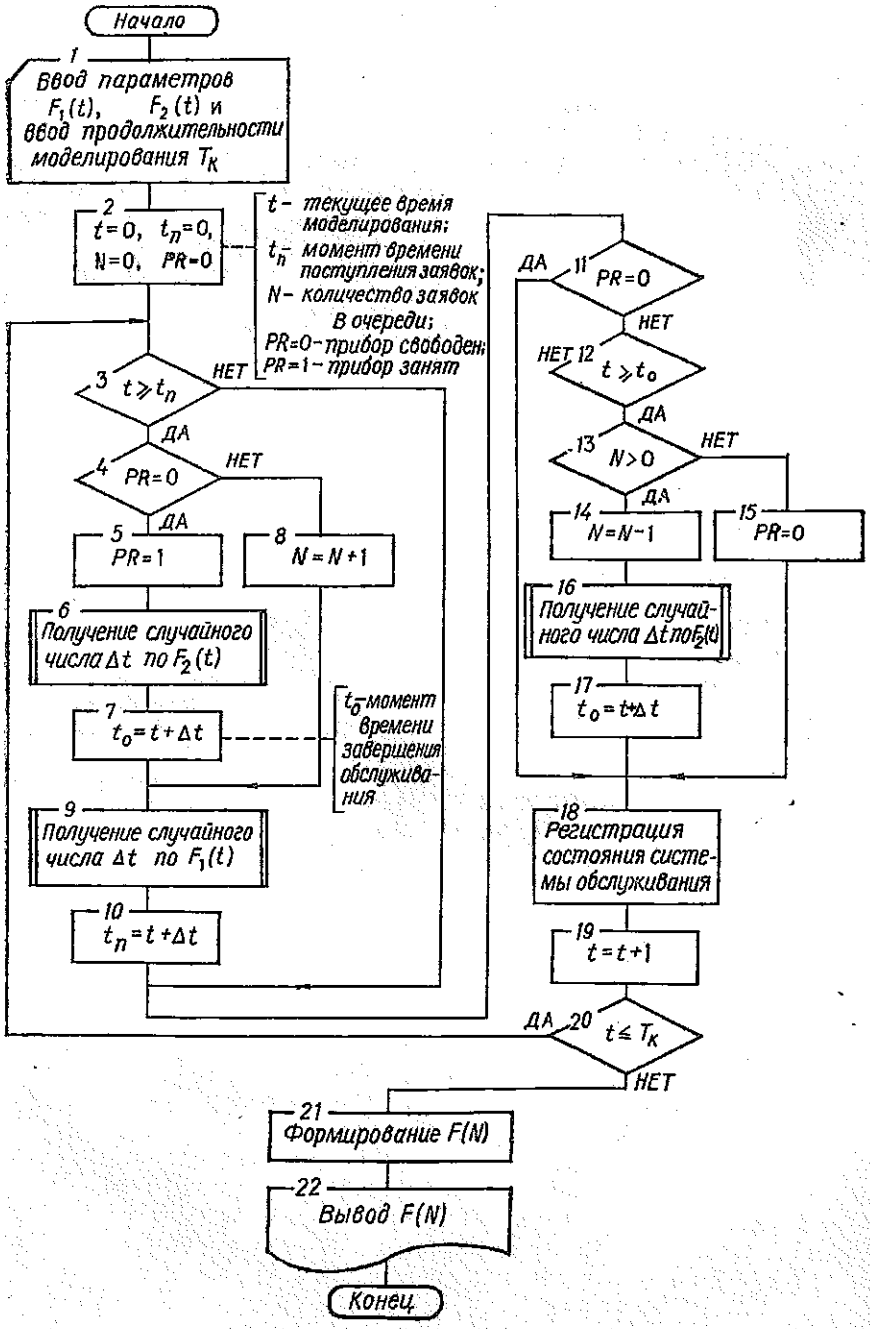


Рис. 6.20. Укрупненная схема алгоритма моделирования работы системы массового обслуживания методом статистических испытаний

мени ее обработки (блок *16*) и определяется момент времени завершения процесса для этой заявки.

Блок *18* регистрирует значения исследуемых параметров системы обслуживания, которые она имеет в текущий момент времени моделирования. В блоке *19* время моделирования увеличивается на единицу. Сравнение с конечным временем моделирования осуществляется в блоке *20*. Если моделирование не завершено, производится возврат в блок *3* для выполнения следующего шага имитации. По окончании моделирования выполняются статистическая обработка результатов (блок *21*) и вывод их на печать.

В зависимости от структуры системы массового обслуживания конкретная реализация алгоритма может отличаться от приведенной на рис. 6.19, однако общие принципы моделирования аналогичны рассмотренным.

Остановимся на способе регистрации состояния системы обслуживания. Пусть необходимо установить закон распределения количества заявок в очереди. На каждом шаге моделирования в очереди находится N заявок. Выберем интервалы длины очереди $(0 — k_1)$, $(k_1 — k_2)$, ..., $(k_{n-1} — k_n)$. Частоту попадания длины очереди в каждый интервал обозначим m_i $(i = 1, 2, ..., n)$. Очевидно, что после завершения моделирования $\sum_{i=1}^{n} m_i = T_k$. Определим на каждом шаге моделирования, к какому интервалу принадлежит значение N, и к соответствующей частоте m_i добавим единицу. Не исключено, что длина очереди может быть больше k_n, поэтому следует предусмотреть сообщение программы о таком случае и повторить моделирование с увеличенным шагом интервалов $k_{i-1} — k_i$ $(i = 1, 2, ..., n)$.

7. ПРИМЕРЫ ПРИМЕНЕНИЯ ЭВМ ДЛЯ РЕШЕНИЯ ЗАДАЧ ГОРНОГО ПРОИЗВОДСТВА

7.1. Прогнозирование показателей горного производства методом наименьших квадратов

Управление горным предприятием и планирование горного производства связаны с необходимостью прогнозировать горно-технические условия ведения работ, технико-экономические и другие показатели, подчиняющиеся вероятностным законам распределения. Для этих целей применяются методы прогнозирования, основанные на анализе и моделировании динамических рядов [1].

В основу прогнозирования положена идея экстраполяции динамических рядов, т. е. распространение на прогнозируемый период времени тех тенденций развития процесса, которые сложились к данному моменту. Численное значение любого показателя исследуемого процесса в момент времени t можно представить так:

$$V_t = W_t + \varepsilon_t \quad (t = 1, 2, \ldots), \tag{7.1}$$

[1] Динамический ряд представляет собой числовую последовательность, количественно характеризующую изменение исследуемого показателя во времени.

где V_t — динамический ряд; W_t — детерминированная составляющая динамического ряда (тренд); ε_t — случайная составляющая динамического ряда.

Таблица 7.1. Добыча угля шахтой за период 1978 — 1985 гг.

Год	t	Добыча угля V_t, тыс. т	Год	t	Добыча угля V_t, тыс. т
1978	1	803	1982	5	975
1979	2	843	1983	6	1002
1980	3	870	1984	7	991
1981	4	904	1985	8	1016

Прогнозирование динамического ряда сводится к прогнозированию тренда W_t и случайной составляющей ε_t. Метод наименьших квадратов позволяет определить параметры уравнения тренда, по которому путем экстраполяции можно осуществить прогнозирование детерминированной составляющей на краткосрочный период (1—3 шага во времени).

Предположим, уравнение тренда имеет вид

$$W_t = f(a_1, a_2, \ldots, a_n, t). \tag{7.2}$$

Здесь $a_1, a_2, \ldots, a_n$ — параметры уравнения тренда; t — независимая переменная (время).

Согласно методу наименьших квадратов сумма квадратов рассогласований между реальными значениями числовой последовательности и значениями, определенными по уравнению тренда, должна быть минимальной:

$$D = \sum_{t=1}^{T} (V_t - f(a_1, a_2, \ldots, a_n, t))^2 \to \min, \tag{7.3}$$

где T — количество точек исходного динамического ряда.

При построении модели прогнозирования принимают во внимание тот факт, что чем больше исходная информация удалена в прошлое, тем в меньшей степени она влияет на будущее значение исследуемого процесса. Для учета этого факта каждому отклонению процесса от тренда присваивается свой вес β_t ($t = 1, 2, \ldots, T$). Обычно принимают $\beta_t = \alpha^{T-t}$ ($\alpha < 1$). Очевидно, что с удалением в прошлое значение β_t убывает. Критерий такой модели можно записать следующим образом:

$$D = \sum_{t=1}^{T} \beta_t (V_t - f(a_1, a_2, \ldots, a_n, t))^2 \to \min. \tag{7.4}$$

Чтобы определить минимум (7.4), необходимо решить систему уравнений

$$\frac{\partial D}{\partial a_t} = 0, \quad (t = 1, 2, \ldots, n) \tag{7.5}$$

относительно параметров a_t.

Для прогнозирования динамических рядов уравнение тренда (7.2) задают полиномом $(n-1)$-й степени:

$$W_t = a_1 + a_2 t + a_3 t^2 + \cdots + a_n t^{(n-1)}, \tag{7.6}$$

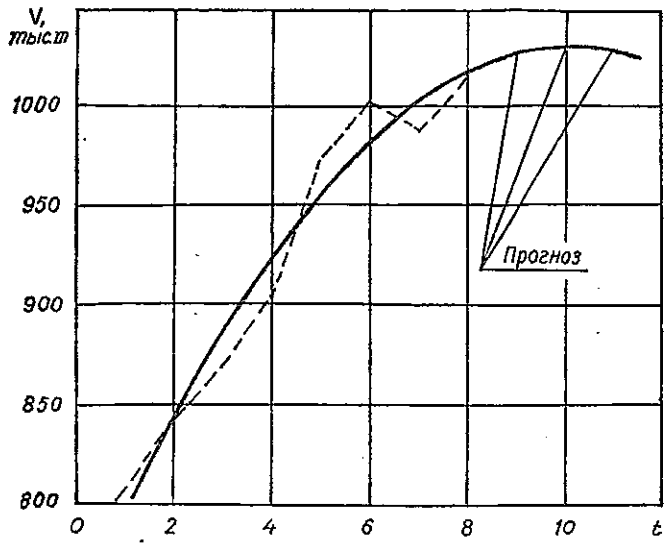


Рис. 7.1. Динамический ряд годовой добычи угля

При этом система уравнений (7.5) преобразуется следующим образом:

$$\begin{aligned}
&a_1\sum_{t=1}^{T}\beta_t + a_2\sum_{t=1}^{T}\beta_t t + \cdots + a_n\sum_{t=1}^{T}\beta_t t^{(n-1)} = \sum_{t=1}^{T}\beta_t V_t;\\
&a_1\sum_{t=1}^{T}\beta_t t + a_2\sum_{t=1}^{T}\beta_t t^2 + \cdots + a_n\sum_{t=1}^{T}\beta_t t^n = \sum_{t=1}^{T}\beta_t V_t t;\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
&a_1\sum_{t=1}^{T}\beta_t t^{(n-1)} + a_2\sum_{t=1}^{T}\beta_t t^n + \cdots + a_n\sum_{t=1}^{T}\beta_t t^{(2n-2)} = \sum_{t=1}^{T}\beta_t V_t t^{(n-1)}.
\end{aligned} \tag{7.7}$$

Решая систему (7.7) относительно параметров тренда a_1, a_2, ..., a_n, получаем модель прогнозирования:

$$W_{T+\tau} = f(a_1,\ a_2,\ \ldots,\ a_n,\ T+\tau). \tag{7.8}$$

Здесь $\tau = 1, 2, \ldots, \tau_p$ — интервал прогнозирования.

В качестве примера рассмотрим прогнозирование годовой добычи угля шахтой. Динамический ряд добычи угля за период 1978—1985 гг. приведен в табл. 7.1 и на рис. 7.1. На рисунке видно, что тренд динамического ряда может быть представлен полиномом второй степени:

$$W_t = a_1 + a_2 t + a_3 t^2. \tag{7.9}$$

Следовательно, систему уравнений относительно параметров прогнозирующей модели можно записать так:

$$a_1\sum_{t=1}^{8}\beta_t + a_2\sum_{t=1}^{8}\beta_t t + a_3\sum_{t=1}^{8}\beta_t t^2 = \sum_{t=1}^{8}\beta_t V_t;$$

$$a_1\sum_{t=1}^{8}\beta_t t+a_2\sum_{t=1}^{8}\beta_t t^2+a_3\sum_{t=1}^{8}\beta_t t^3=\sum_{t=1}^{8}\beta_t V_t t;$$

$$a_1\sum_{t=1}^{8}\beta_t t^2+a_2\sum_{t=1}^{8}\beta_t t^3+a_3\sum_{t=1}^{8}\beta_t t^4=\sum_{t=1}^{8}\beta_t V_t t^2.$$

В данной системе в каждую сумму помимо аргументов и функций входят весовые коэффициенты β_t, поэтому для ее формирования и решения нельзя использовать программу метода наименьших квадратов, приведенную в подразд. 6.4.

Рассмотрим алгоритм формирования коэффициентов системы уравнений (7.7). Для этого введем следующие матрицы:

$$T_t=\begin{bmatrix}1 & 1 & 1 & \dots & 1\\ 1 & 2 & 2^2 & \dots & 1^{(n-1)}\\ 1 & 3 & 3^2 & \dots & 3^{(n-1)}\\ \dots & \dots & \dots & \dots & \dots\\ 1 & T & T^2 & \dots & T^{(n-1)}\end{bmatrix};$$

$$T_\beta=\begin{bmatrix}\beta_1 & \beta_2 & \beta_3 & \dots & \beta_T\\ \beta_1 & 2\beta_2 & 3\beta_3 & \dots & T\beta_T\\ \beta_1 & 2^2\beta_2 & 3^2\beta_3 & \dots & T^2\beta_T\\ \dots & \dots & \dots & \dots & \dots\\ \beta_1 & 2^{(n-1)}\beta_2 & 3^{(n-1)}\beta_3 & \dots & T^{(n-1)}\beta_T\end{bmatrix};$$

$$V=\begin{pmatrix}V_1\\ V_2\\ V_3\\ \dots\\ V_T\end{pmatrix};\quad a=\begin{pmatrix}a_1\\ a_2\\ a_3\\ \dots\\ a_n\end{pmatrix}.$$

Обозначив $A=T_\beta T_t$, $B=T_\beta V$, перепишем систему (7.7) как $Aa=B$.

Таким образом, программа прогнозирования показателей горного производства должна состоять из таких этапов.

1. Ввод исходных данных n, T, α, τ_p, V.
2. Формирование матриц T_β, T_t.
3. Вычисление матрицы A.
4. Вычисление столбцовой матрицы B.
5. Решение системы уравнений $Aa=B$ с использованием модуля-подпрограммы RSUMJG.
6. Вывод на печать результатов вычислений — столбцовой матрицы B.
7. Вычисление по формуле (7.9) и вывод на печать прогнозируемых показателей при $t=T+\tau$ ($\tau=1, 2, \dots, \tau_p$).

Согласно алгоритму Жордана — Гаусса решение системы уравнений помещается на место столбца свободных членов, следовательно, в уравнение (7.6) подставляем $a_i=B_i$ ($i=1, 2, \dots, n$).

Программа прогнозирования показателей горного производства записывается следующим образом:

```
C ПРОГРАММА ПРОГНОЗА ПОКАЗАТЕЛЕЙ ГОРНОГО
C     ПРОИЗВОДСТВА МЕТОДОМ НАИМЕНЬШИХ КВАДРАТОВ
C          ПРИМЕЧАНИЕ
```

```
C     В ПРОГРАММЕ ИСПОЛЬЗУЕТСЯ
C      МОДУЛЬ-ПОДПРОГРАММА RSUMJG
      DIMENSION TT (50,5),TB (5,50),V (50),B (5),A (5,5)
      READ 1,N,IT,ALFA,ITP,(V (I),I = 1,IT)
    1 FORMAT (2I3,F4.2,I3/(10F8.2))
      PRINT 2,N,IT,ALFA,ITP,(V (I),I = 1,IT)
    2 FORMAT (' N =',I2,' IT =',I2,' ALFA =',F4.2,
     *' ITP =',I2/' V ='/(10F8.2))
C ФОРМИРОВАНИЕ МАТРИЦЫ TT
      DO 3 I = 1,N
    3 TT (1,I) = 1.
      DO 4 I = 2,IT
      TT (I,1) = 1.
      DO 4 J = 2,N
    4 TT (I,J) = TT (I,J — 1)*I
C ФОРМИРОВАНИЕ МАТРИЦЫ TB
      TB (1,IT) = 1.
      DO 5 I = 2,IT
    5 TB (1,IT — I + 1) = TB (1,IT — I + 2)*ALFA
      DO 6 I = 2,N
      DO 6 J = 1,IT
    6 TB (I,J) = TT (J,I)*TB (1,J)
C ВЫЧИСЛЕНИЕ МАТРИЦЫ A
      DO 7 I = 1,N
      DO 7 J = 1,N
      A (I,J) = 0.
      DO 7 K = 1,IT
    7 A (I,J) = A (I,J) + TB (I,K)*TT (K,J)
C ВЫЧИСЛЕНИЕ СТОЛБЦОВОЙ МАТРИЦЫ B
      DO 8 I = 1,N
      B (I) = 0.
      DO 8 J = 1,IT
    8 B (I) = B (I) + TB (I,J)*V (J)
C РЕШЕНИЕ СИСТЕМЫ УРАВНЕНИЙ A*X = B
C      МЕТОДОМ ЖОРДАНА — ГАУССА
C 1
C ПРЕОБРАЗОВАНИЕ МАТРИЦЫ A В ВЕКТОРНУЮ ФОРМУ
      K = 1
      L = 1
      DO 9 J = 1,N
      DO 9 I = 1,N
      A (K,L) = A (I,J)
      K = K + 1
      IF (K.LE.5) GO TO 9
      K = 1
      L = L + 1
    9 CONTINUE
C 2
C РЕШЕНИЕ СИСТЕМЫ УРАВНЕНИЙ
      CALL RSUMJG (N,A,B,IND)
      IF (IND.EQ.0) GO TO 10
      PRINT 11
   11 FORMAT ('ЗАДАЧА  НЕ ИМЕЕТ РЕШЕНИЯ')
      GO TO 12
   10 PRINT 13, (I,B (I),I = 1,N)
   13 FORMAT (' ПАРАМЕТРЫ МОДЕЛИ ПРОГНОЗА'/
     *(' A (',I1,') =',E11.4))
C ПРОГНОЗ ПОКАЗАТЕЛЕЙ ПРОИЗВОДСТВА
      DO 14 I = 1,ITP
      T = (IT + I)*1.
      R = B (1)
      DO 15 J = 2,N
```

```
15 R = R + B (J)*T** (J — 1)
14 PRINT 16 I,R
16 FORMAT (' ИНТЕРВАЛ ПРОГНОЗА',I2,
  *' ЗНАЧЕНИЕ ПРОГНОЗИРУЕМОГО ПОКАЗАТЕЛЯ',E14.4)
12 STOP
   END
```

В программе приняты такие обозначения:

Переменные	T_t	T_B	V	A	B	n	T	α	τ_p
Имена	TT	TB	V	A	B	N	IT	ALFA	ITP

Результаты решения задачи прогнозирования годовой добычи угля шахтой при $n = 3$, $T = 8$, $\alpha = 0{,}9$, $\tau_p = 3$ даны на рис. 7.1. Тренд хорошо аппроксимирует исходные данные, что позволяет считать точность прогнозирования удовлетворительной.

7.2. Определение вместимости перегрузочного бункера на концентрационном горизонте карьера методом имитационного моделирования

Рассмотрим схему транспортировки полезного ископаемого из карьера, применяемую при циклично-поточной технологии горных работ. Горная масса автосамосвалами доставляется из добычных забоев на перегрузочный пункт концентрационного горизонта (рис. 7.2). На грохоте горная масса разделяется на надгрохотный и подгрохотный продукты. Последний (размеры кусков менее 400 мм) поступает в бункер, откуда пластинчатым питателем подается на наклонный ленточный конвейер, расположенный на борту карьера. Надгрохотный продукт проходит через дробилку и после дробления также поступает на конвейер, который транспортирует горную массу на поверхность карьера.

Перегрузочный пункт выполняет функции усреднения грузопотока, чем обеспечивает относительно равномерную загрузку конвейера. Благодаря бункеру перегрузочного пункта дискретный грузопоток горной массы преобразуется в непрерывный, подчиненный нормальному закону распределения с известными математическим ожиданием и дисперсией.

Предположим, что законы распределения потока автосамосвалов, поступающих на разгрузку, грузопотока горной массы на конвейере и ее гранулометрического состава ус-

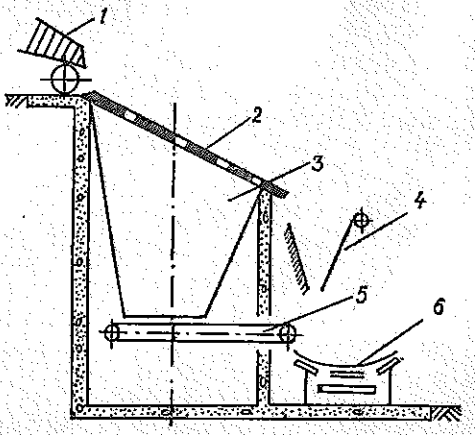


Рис. 7.2. Перегрузочный пункт концентрационного горизонта карьера:
1 — автосамосвал; *2* — грохот; *3* — бункер; *4* — щековая дробилка; *5* — пластинчатый питатель; *6* — конвейер

тановлены. Необходимо определить взаимосвязь между вместимостью бункера концентрационного горизонта и потерями времени на ожидание разгрузки автосамосвалами из-за отсутствия свободного места в бункере.

Очевидно, что рассматриваемая система представляет собой систему массового обслуживания, в которой перегрузочный пункт концентрационного горизонта выполняет функции обслуживающего прибора, а порции горной массы, доставляемой автосамосвалами,— функции заявок. Время обслуживания заявки определяется законом распределения гранулометрического состава порции горной массы и производительностью питателя. Поэтому для решения поставленной задачи воспользуемся методом имитационного моделирования систем массового обслуживания.

Схема алгоритма имитационного моделирования функционирования перегрузочного пункта на концентрационном горизонте карьера показана на рис. 7.3.

В схеме приняты следующие обозначения переменных (в скобках указаны их имена в программе): Q (Q) — грузоподъемность автосамосвала; t_k (TK) — продолжительность моделирования; λ (LAMBDA) — плотность потока автосамосвалов (количество машин, прибывающих на перегрузочный пункт в единицу времени); V_m (VM) — максимальная вместимость бункера; G (G) — математическое ожидание долевого состава горной массы крупностью менее 400 мм; SG (SG) — среднее квадратическое отклонение долевого состава горной массы крупностью менее 400 мм; P (P) — математическое ожидание производительности питателя бункера; SP (SP) — среднее квадратическое отклонение производительности питателя бункера; m_i, $i = 1, 2, ..., 11$ (M) — массив частот количества автосамосвалов, стоящих в очереди в ожидании разгрузки; t_t (TT) — текущее время моделирования; t_p (TP) — момент времени прибытия очередного автосамосвала на перегрузочный пункт; N (N) — количество автосамосвалов, стоящих в ожидании разгрузки; V (V) — количество горной массы в бункере; $V_б$ (VB) — вместимость бункера; GSL (GSL) — случайное значение долевого состава горной массы крупностью менее 400 мм; PSL (PSL) — случайное значение производительности питателя бункера; S (S) — средняя длина очереди автосамосвалов.

Блок *1* вводит исходные данные. Блоки *2—4* формируют начальные значения параметров моделирования (для вместимости бункера $V_б = 2Q, 3Q, ..., V_m$).

Если текущее время моделирования совпадает с моментом времени прибытия автосамосвала (блок *5*), последний устанавливается в очередь (блок *6*) и формируется момент времени прибытия следующего автосамосвала (блоки *7*, *8*). Контроль за очередью осуществляет блок *9*. При наличии очереди проверяется состояние бункера (блок *10*). В том случае, когда автосамосвал можно поставить под разгрузку, устанавливается случайное значение долевого состава горной массы крупностью менее 400 мм (блоки *11—13*). Определенная часть горной массы поступает в бункер (блок *14*), и количество автосамосвалов в очереди уменьшается на единицу.

Если бункер заполнен (блок *16*), формируется случайное значение производительности питателя (блок *17*) и соответствующее

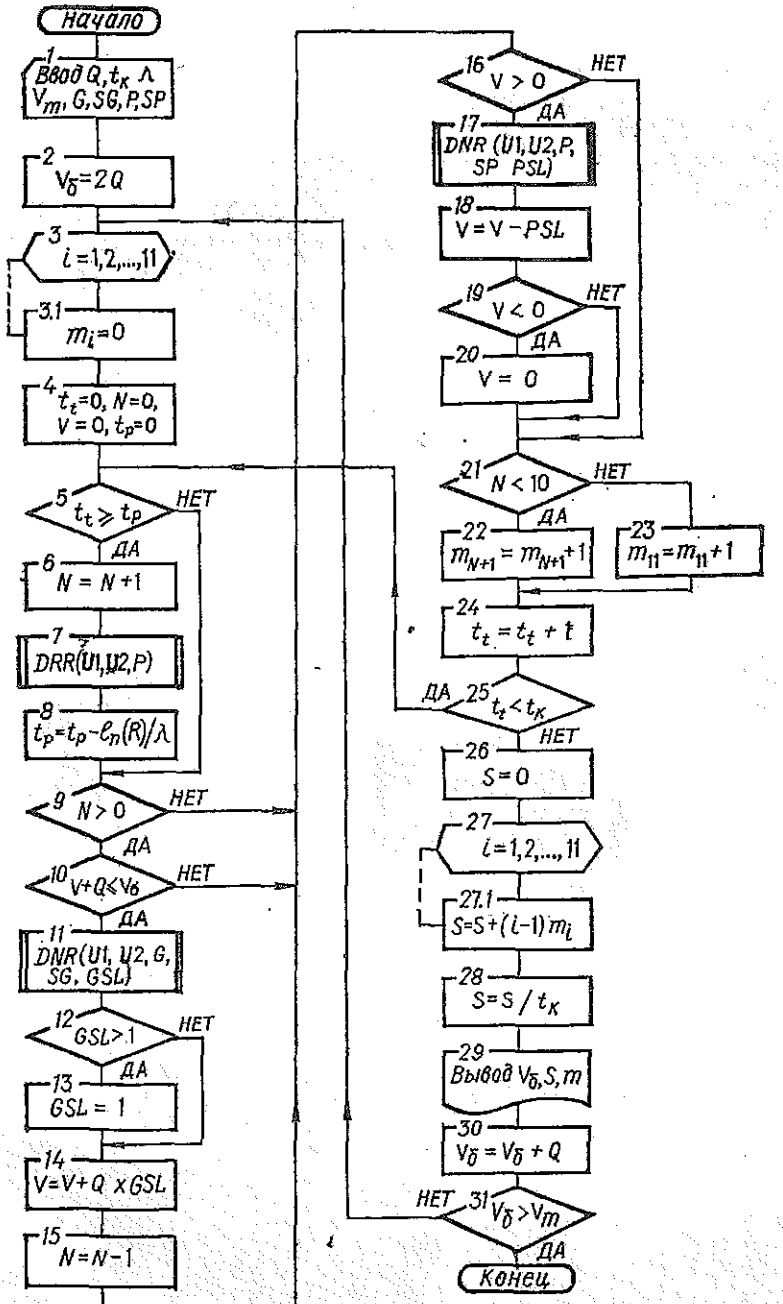


Рис. 7.3. Схема алгоритма имитационного моделирования перегрузочного пункта на концентрационном горизонте карьера

количество горной массы удаляется из бункера (блоки *18—20*). Блоки *21—23* фиксируют состояние очереди.

Блоки *5—23* реализуют имитацию процесса в течение единичного интервала времени. Увеличение текущего времени на единицу и сравнение с заданной продолжительностью моделирования выполняется блоками *24*, *25*. Если текущее время не достигло конечного значения, управление передается блоку *5* для осуществления очередного шага имитации процесса.

После завершения моделирования при заданной вместимости бункера вычисляется средняя длина очереди автосамосвалов (блоки *26—28*). Результаты моделирования выводятся на печать (блок *29*). Вместимость бункера увеличивается и сравнивается с конечным значением (блоки *30*, *31*). При $V_6 \leqslant V_m$ происходит возврат на повторное моделирование процесса в течение времени t_k.

В описанном алгоритме моделирования работы перегрузочного пункта предполагается, что интервалы времени между смежными прибытиями автосамосвалов подчинены экспоненциальному закону распределения, а гранулометрический состав горной массы и производительность питателя бункера — нормальному.

Программа имитационного моделирования работы перегрузочного пункта записывается так:

```
C ПРОГРАММА ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ РАБОТЫ
C ПЕРЕГРУЗОЧНОГО ПУНКТА НА КОНЦЕНТРАЦИОННОМ ГОРИЗОНТЕ
C КАРЬЕРА
C          ПРИМЕЧАНИЕ
C       ДЛЯ ВЫПОЛНЕНИЯ ДАННОЙ ПРОГРАММЫ
C       ТРЕБУЮТСЯ ДАТЧИКИ СЛУЧАЙНЫХ ЧИСЕЛ,
C       РЕАЛИЗОВАННЫЕ МОДУЛЬ-ПОДПРОГРАММАМИ DRR И DNR
      REAL LAMBDA
      DIMENSION M (11)
      U1 = 3.141592
      U2 =.5421Ø18
      READ 1,Q,TK,LAMBDA,VM,G,SG,P,SP
    1 FORMAT (F5.1,F8.1,F6.3,F5.1,2F5.3,2F6.2)
      PRINT 2,Q,TK,LAMBDA,VM,G,SG,P,SP
    2 FORMAT (' ИСХОДНЫЕ ДАННЫЕ'/
     1' ГРУЗОПОД''ЕМНОСТЬ ВАГОНЕТКИ',F5.1/
     2' ПРОДОЛЖИТЕЛЬНОСТЬ МОДЕЛИРОВАНИЯ',F8.1/
     3' ПЛОТНОСТЬ ПОТОКА ВАГОНЕТОК',F6.3/
     4' МАКСИМАЛЬНАЯ ВМЕСТИМОСТЬ БУНКЕРА',F6.1/
     5' РАСПРЕДЕЛЕНИЕ ГРАНУЛОМЕТРИЧЕСКОГО СОСТАВА ГОРНОЙ'/
     6' МАССЫ G =',F5.3,' SG =',F5.3/
     7' РАСПРЕДЕЛЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ПИТАТЕЛЯ',
     8' БУНКЕРА P =',F6.2,' SP =',F6.2)
      VB = 2.*Q
  1ØØ DO 3 I = 1,11
    3 M (I) = Ø
      TT = Ø.
      TP = Ø.
      N = Ø
      V = Ø.
C
C ИМИТАЦИЯ ПРИБЫТИЯ АВТОСАМОСВАЛА НА ПЕРЕГРУЗОЧНЫЙ ПУНКТ
    5 IF (TT.LT.TP) GO TO 9
      N = N + 1
      CALL DRR (U1,U2,R)
```

```
      TP = TP — ALOG (R)/LAMBDA
C ИМИТАЦИЯ ПОСТУПЛЕНИЯ ПОРЦИИ ГОРНОЙ МАССЫ В БУНКЕР
    9 IF (N.EQ.0) GO TO 16
      IF (V + Q.GT.VB) GO TO 16
      CALL DNR (U1,U2,G,SG,GSL)
      IF (GSL.GT.1.) GSL = 1.
      V = V + Q*GSL
      N = N — 1
C
C ИМИТАЦИЯ РАЗГРУЗКИ БУНКЕРА В ТЕЧЕНИЕ
C     ЕДИНИЧНОГО ИНТЕРВАЛА ВРЕМЕНИ
   16 IF (V.EQ.0) GO TO 21
      CALL DNR (U1,U2,P,SP,PSL)
      V = V — PSL
      IF (V.LT.0.) V = 0.
C
C ФИКСАЦИЯ СОСТОЯНИЯ ОЧЕРЕДИ АВТОСАМОСВАЛОВ
   21 IF (N.LT.10) M (N + 1) = M (N + 1) + 1
      IF (N.GE.10) M (11) = M (11) + 1
C
C ПЕРЕХОД К СЛЕДУЮЩЕМУ ШАГУ МОДЕЛИРОВАНИЯ
      TT = TT + 1
      IF (TT.LT.TK) GO TO 5
C
C МОДЕЛИРОВАНИЕ ПРИ ДАННОЙ ВМЕСТИМОСТИ БУНКЕРА ЗАВЕРШЕНО
C
C ВЫЧИСЛЕНИЕ СРЕДНЕЙ ДЛИНЫ ОЧЕРЕДИ АВТОСАМОСВАЛОВ
      S = 0.
      DO 27 I = 1,11
   27 S = S + (I — 1)*M (I)
      S = S/TK
C
C ВЫВОД РЕЗУЛЬТАТОВ МОДЕЛИРОВАНИЯ
      PRINT 29, VB,S,M
   29 FORMAT (' VB =',F5.1,' S =',F6.3/
     1' M =',11I5)
C
C ВОЗВРАТ НА ПОВТОРНОЕ МОДЕЛИРОВАНИЕ С НОВОЙ
C     ВМЕСТИМОСТЬЮ БУНКЕРА
      VB = VB + Q
      IF (VB.LE.VM) GO TO 100
      STOP
      END
```

Результаты имитационного моделирования при $Q = 40$ т, $\lambda = 0{,}25$ автосамосвала в минуту, $G = 0{,}95$, $SG = 0{,}03$, $P = 10$ т/мин, $SP = 1{,}5$ т/мин показаны на рис. 7.4 и 7.5. На рис. 7.4 видно, что увеличение вместимости бункера позволяет уменьшить очередь автосамосвалов, простои карьерного транспорта и, следовательно, повысить производительность системы в целом.

Согласно рис. 7.5 наиболее целесообразно принимать вместимость бункера $V_6 = 200$ т. В данном случае очереди автосамосвалов не будет.

Окончательно вместимость бункера следует устанавливать на основе экономического расчета, выполненного с учетом обеспечения минимизации функции

$$F(V) = З_1(V) + З_2(V) \to \min,$$

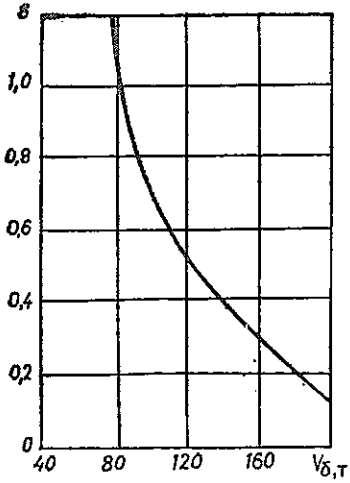


Рис. 7.4. Зависимость средней длины очереди автосамосвалов на перегрузочном пункте от вместимости бункера

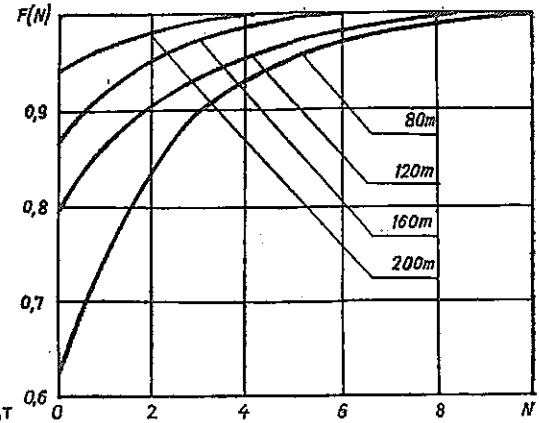


Рис. 7.5. Функции распределения количества автосамосвалов в очереди при разной вместимости бункера на перегрузочном пункте

где $З_1(V)$ — затраты, обусловленные снижением производительности системы из-за переполнения бункера; $З_2(V)$ — капитальные и эксплуатационные затраты на бункер.

7.3. Определение порядка отработки добычных заходок с учетом стабилизации качества рудного сырья

При работе двух добычных экскаваторов в забоях с переменным качеством полезного ископаемого возникает необходимость решать задачу обеспечения стабильного по качеству потока руды, поступающей на обогатительную фабрику. На рис. 7.6 показано взаимное расположение добычных забоев и изолинии распределения качества полезного ископаемого в отрабатываемых заходках. Следует найти такое взаимное расположение добычных экскаваторов в рабочих заходках, чтобы колебание качества полезного ископаемого в двух рудопотоках было минимальным.

Предположим, что экскаваторы имеют равную производительность. Разобьем отрабатываемые заходки на блоки равного объема и присвоим им номера в определенной последовательности. Каждый блок характеризуется стабильным содержанием полезного компонента в руде.

Обозначим α_{ij} — содержание полезного компонента в j-м блоке, отрабатываемом i-м экскаватором ($j = 1, 2, \ldots, n$; $i = 1, 2$). Пусть по условиям проведения вскрышных работ второй экскаватор будет отставать от первого или опережать его не более чем на m блоков. На начало планируемого периода отработки месторождения первый экскаватор установлен в j_1-м блоке, а второй — в j_2-м, причем $|j_1 - j_2| \leqslant m$. В данном случае рудопоток, сформированный при отработке добычных заходок, будет характеризоваться следующими

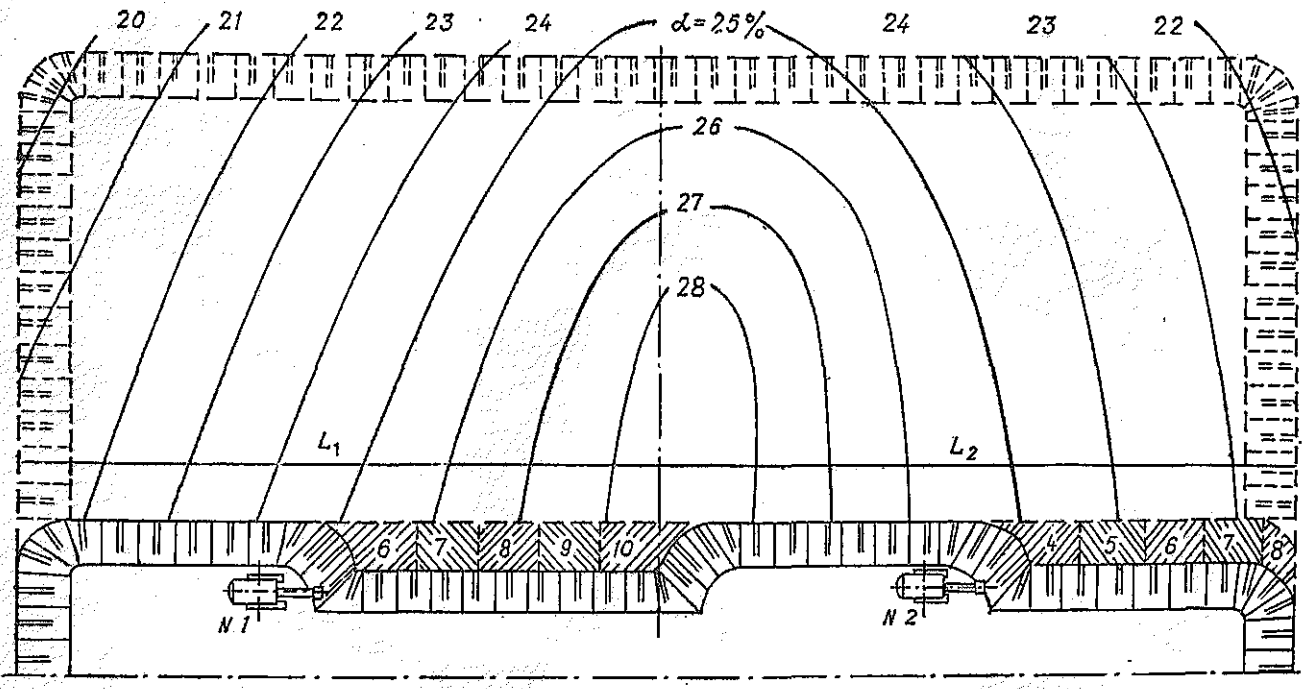


Рис. 7.6. Расположение добычных забоев при отработке горизонтально залегающего пласта полезного ископаемого

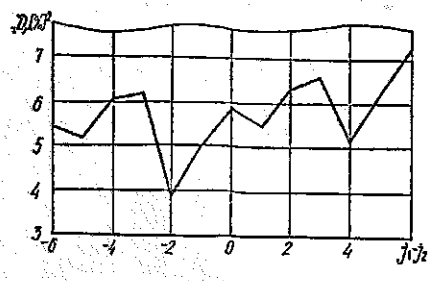


Рис. 7.7. Зависимость дисперсии качества результирующего рудопотока от взаимного расположения добычных забоев

показателями качества полезного ископаемого:

$$0{,}5\,(\alpha_{1,j_1}+\alpha_{2,j_2});$$
$$0{,}5\,(\alpha_{1,j_1+1}+\alpha_{2,j_2+1});$$
$$0{,}5\,(\alpha_{1,j_1+2}+\alpha_{2,j_2+2}),\ \ldots$$

Обозначим $a_k = 0{,}5\,(\alpha_{1,j_1+k-1} + \alpha_{2,j_2+k-1})$ $(k = 1, 2, \ldots, l)$. Дисперсия качества результирующего рудопотока

$$D = \sum_{k=1}^{l} a_k^2/l - \left(\sum_{k=1}^{l} a_k/l\right)^2, \quad (7.10)$$

где $l = n - j_1 + 1$ при $j_1 \geqslant j_2$ и $l = n - j_2 + 1$, если $j_2 > j_1$.

Примем $j_1 = m + 1$. Изменяя показатель $j_2 = 1, 2, \ldots, (2m + 1)$, получаем последовательность дисперсий $D_1, D_2, \ldots, D_{2m+1}$, из которых выберем минимальную. Ее индекс будет определять начальное положение второго добычного экскаватора.

Алгоритм решения рассмотренной задачи состоит из следующих шагов.

1. Ввести исходные данные m, n, α_{ij} $(i = 1, 2, j = 1, 2, \ldots, n)$.
2. Вычислить $j_1 = m + 1$.
3. Принять $j_2 = 1$.
4. Если $j_1 \geqslant j_2$, принять $l = n - j_1 + 1$; в противном случае $l = n - j_2 - 1$.
5. Вычислить $a_k = (\alpha_{1,j_1+k-1} + \alpha_{2,j_2+k-1})/2$ при $k = 1, 2, \ldots, l$.
6. Вычислить дисперсию D по формуле (7.10).
7. Отпечатать j_2, D.
8. Увеличить j_2 на единицу.
9. Если $j_2 \leqslant 2m + 1$, перейти к п. 4.
10. Конец.

Приведенному алгоритму соответствует такая программа:

```
C ПРОГРАММА ОПРЕДЕЛЕНИЯ ПОРЯДКА ОТРАБОТКИ ДОБЫЧНЫХ
C ЗАХОДОК С УЧЕТОМ СТАБИЛИЗАЦИИ КАЧЕСТВА РУДОПОТОКА
      DIMENSION AL (2,200),A (200)
C 1   ВВОД ИСХОДНЫХ ДАННЫХ
      READ 1,M,N,((AL (I,J),J = 1,N),I = 1,2)
    1 FORMAT (2I4/(10F5.2))
      PRINT 2,M,N,((AL (I,J),J = 1,N),I = 1,2)
    2 FORMAT (' M =',I2,' N =',I4/' AL ='/(10F5.2))
C 2
      J1 = M + 1
C 3
      J2 = 1
C 4
    4 IF (J1.GE.J2) L = N - J1 + 1
      IF (J2.GT.J1) L = N - J2 + 1
C 5   ВЫЧИСЛЕНИЕ КАЧЕСТВА ПОЛЕЗНОГО ИСКОПАЕМОГО
C     РЕЗУЛЬТИРУЮЩЕГО РУДОПОТОКА
      DO 5 K = 1.L
    5 A (K) = (AL (1,J1 + K - 1) + AL (2,J2 + K - 1))/2.
C 6   ВЫЧИСЛЕНИЕ ДИСПЕРСИИ РЕЗУЛЬТИРУЮЩЕГО РУДОПОТОКА
      S1 = 0.
      S2 = 0.
```

```
      DO 6 K = 1,L
      S1 = S1 + A (K)
    6 S2 = S2 + A (K) * * 2
      D = S2/L — (S1/L) * * 2
C 7   ВЫВОД НА ПЕЧАТЬ РЕЗУЛЬТАТОВ ВЫЧИСЛЕНИЙ
      FRINT 7, J2, D
    7 FORMAT (' J2 = ', I3, ' D = ', E18.4)
C 8
      J2 = J2 + 1
C 9
      IF (J2. LE. 2 * M + 1) GO TO 4
C 10
      STOP
      END
```

В программе приняты обозначения:

Переменные	m	n	α	a	j_1	j_2	l	D
Имена	N	N	AL	A	J1	J2	L	D

По данной программе были выполнены расчеты для одного из участков Покровского марганцово-рудного месторождения. Участок разделен на два крыла, в каждом из которых должен работать один экскаватор. Заходки разбиты на блоки объемом по 3 тыс. м³. Для каждого блока вычислено среднее содержание полезного компонента. В одном крыле выделено 120 блоков. Согласно условиям ведения вскрышных и добычных работ опережение между экскаваторами не должно превышать шести блоков (это составляет 300...350 м по фронту работ).

Результаты расчета дисперсии качества полезного ископаемого в рудопотоке в зависимости от взаимного расположения добычных забоев показаны на рис. 7.7. На рисунке видно, что второй экскаватор должен опережать первый на два блока относительно принятых точек начала отсчета (100...120 м по фронту работ).

Обратим внимание на одну существенную деталь: в зависимости от расположения добычных забоев дисперсия изменяется от 3,85 до 7,1. Следовательно, выбором правильного взаимного положения экскаваторов при отработке месторождения можно существенно (в данном примере — в два раза) уменьшить колебание качества полезного компонента в результирующем рудопотоке.

* * *

Итак, познакомились с основами программирования на алгоритмическом языке ФОРТРАН-4, изучили наиболее часто встречающиеся в горно-технических расчетах численные методы и на примерах типовых задач математического и горно-технического характера подкрепили свои практические навыки в программировании. Мы не сомневаемся также, что, работая над книгой, вы находили время и возможность самостоятельно составить схему алгоритма, написать (пусть даже небольшую) программу, отладить и решить ее на ЭВМ. При этом вы, наверное, заметили, что в первой вашей программе было много неточностей, и ЭВМ не пожелала ее воспринять, из-за чего вы потратили столько времени на

исправления, которого хватило бы на решение задачи вручную. Однако это не лишило вас инициативы. Вторая ваша задача была посложнее, но непроизводительных потерь времени — значительно меньше. После решения третьей задачи у вас появились некоторые профессиональные навыки и вы подумали, что если так дело пойдет и дальше, то вскоре вам не понадобятся услуги профессиональных программистов.

А зачем все это надо? Вы ведь изучали наше пособие не просто из любопытства к искусству программирования. Вы не хотите отстать от технического прогресса, который не в последнюю очередь определяется вычислительной техникой. И сегодня инженер может считаться квалифицированным, если он с компьютером на ты, так же как 30—40 лет назад не могло быть инженера, который не вдалел бы логарифмической линейкой. Поэтому ваше умение программировать и общаться с ЭВМ является непременным условием технической культуры.

Надеемся, что вы не зря потратили время. Труд программиста в определенном смысле сродни труду пианиста. Если пианист хотя бы день не упражнялся, его исполнительское мастерство ухудшается. Если программист делает длительный перерыв в программировании, он теряет навыки. Поэтому советуем искать сферу приложения своим знаниям в области программирования и поддерживать свою «спортивную форму». Вы увидите, что вскоре это станет вашей потребностью и принесет неоспоримую пользу в вашей практической деятельности.

В заключение мы хотим подчеркнуть, что наше учебное пособие является введением в программирование и численные методы. Для более углубленного изучения предмета в конце книги приведен список рекомендуемой литературы. Надеемся, что вы захотите углубить знания этого предмета и воспользуетесь нашими рекомендациями.

Перевод (с английского) основных терминов и расшифровка сокращений

BLKSIZE (block size)	размер блока
CATLG (catalogue)	каталогизировать
C (compile)	компилировать
COND (condition)	условие
CYL (cylinder)	цилиндр
DATA	данные
DCB (data control block)	блок управления данными
DD (data definition)	определение данных
DECK	перфорировать
DELETE	вычеркивать
DISP (disposition)	диспозиция
DSN (data set name)	имя набора данных
EXEC (execute)	выполнять
GO (go off)	начало, старт
JOB	задание
KEEP	сохранять
LET	позволять
LIB (library)	библиотека
LIST	список
LKED (link editor)	редактор связей
LOAD	загружать
MAIN	главный
MAP	карта, наносить на карту
MOD (modify)	модифицировать, видоизменять
MSGLEVEL (message level)	уровень сообщений
MVT (multiprogramming variable task)	мультипрограммный режим с переменным числом задач
NCAL (no call)	не вызывать
NEW	новый
OLD	старый
PARM (parameter)	параметр
PASS	передавать
PGM (program)	программа
PRINT	печатать
PROC (procedure)	процедура
REGION	область
RLSE (release)	освобождать
SER (serial)	серийный
SET	набор, комплект
SHR (share)	владеть совместно, разделять
SOURCE (source program)	источник, исходная программа
SPACE	пространство, область
STEP	шаг, этап
SYS (system)	система
SYSDA (system direct access)	системный прямой доступ
SYSSQ (system sequence)	системный последовательный доступ
TIME	время
TRK (track)	дорожка
UNIT	устройство
VOLUME	том
XREF (external reference)	внешняя ссылка

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Вычислительная техника в инженерных и экономических расчетах : Учебник для вузов / А. В. Петров, В. Е. Алексеев, М. Н. Титов и др.— М. : Высш. шк, 1984.— 320 с.
2. *Грунд Ф.* Принципы операционной системы ОС ЕС: Пер. с нем.— М. : Финансы и статистика, 1984.— 189 с.
3. *Демидович Б. П., Марон И. А.* Основы вычислительной математики.— М. : Наука, 1966.— 664 с.
4. *Землянский А. А., Персиц М. Г.* Основы операционной системы ЕС ЭВМ.— М. : Сов. радио, 1980.— 144 с.
5. Инженерные расчеты на ЭВМ : Справ. пособие / Под ред. В. А. Троицкого.— Л. : Машиностроение, 1979.— 288 с.
6. *Лебедев В. Н., Соколов А. П.* Введение в систему программирования ОС ЕС.— М. : Статистика, 1978.— 144 с.
7. *Мак-Кракен Д., Дорн У.* Численные методы и программирование на ФОРТРАНе.— М. : Мир, 1977.— 584 с.
8. *Потапов В. Д., Яризов А. Д.* Имитационное моделирование производственных процессов в горной промышленности.— М. : Высш. шк., 1981.— 191 с.
9. *Радчик А. Т.* Обработка экспериментальной информации на ЭЦВМ.— К. : Наук. думка, 1976.— 323 с.
10. *Сапицкий К. Ф., Мирошников С. И., Чеховский В. И.* Надежность технологических процессов эксплуатационного участка шахты.— М. : Недра, 1978.— 182 с.
11. ФОРТРАН ЕС ЭВМ / З. С. Брич, Д. В. Капилевич, С. Ю. Котик и др.— М. : Статистика, 1978.— 246 с.
12. ФОРТРАН. Программированное учебное пособие / Под ред. чл.-кор. АН УССР Е. Л. Ющенко.— К. : Вища шк. Головное изд-во, 1980.— 400 с.
13. Численные методы / И. И. Данилина, И. С. Дубровская, Г. Л. Смирнов и др.— М. : Высш. шк., 1976.— 368 с.
14. Электронная вычислительная машина ЕС-1020 / Под ред. А. М. Ларионова.— М. : Статистика, 1975.— 128 с.

Учебное пособие

Виктор Александрович Бунько
Александр Михайлович Эрперт
Владимир Иванович Жуковицкий

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

В ГОРНО-ТЕХНИЧЕСКИХ РАСЧЕТАХ

Редактор
И. К. Михайлова
Художественный редактор
И. Г. Хороший
Технический редактор
В. М. Авдеенко
Корректоры
Л. И. Зотова, Н. Н. Полунина

Информ. бланк № 9198

Сдано в набор 12.02.86. Подп. в печать 15.08.86. БП 04722. Формат 60×90¹/₁₆. Бумага типогр. № 2. Лит. гарн. Выс. печать. Печ. л. 10,0. Кр.-отт. 10,31. Уч.-изд. л. 11,34. Тираж 2000 экз. Изд. № 6762. Зак. 6—1296. Цена 55 к.

Головное издательство издательского объединения «Вища школа», 252054, Киев-54, ул. Гоголевская, 7.

Отпечатано с матриц Головного предприятия республиканского производственного объединения «Полиграфкнига», в Харьковской городской типографии № 16, Харьков-3, ул. Университетская, 16. Зак. 1771.

Уважаемые читатели!

В Головном издательстве издательского объединения «Вища школа» готовятся к выпуску в 1987 г. следующие книги:

Анисимов В. В., Донченко В. С., Закусило О. К. **Элементы теории массового обслуживания и асимптотического анализа систем:** Учеб. пособие 18 л. Яз. рус. 90 к.

Описаны основные классические модели систем массового обслуживания, некоторые специальные типы систем, а также системы, работающие в условиях большой нагрузки. Получены новые результаты по асимптотическому анализу потоков редких событий систем с малой вероятностью потери требования и асимптотическому укрупнению состояний стохастических систем. Приведены методы вероятностного моделирования.

Для студентов вузов, обучающихся по специальностям «Математика», «Прикладная математика», «Экономическая кибернетика».

Володарский Е. Т., Малиновский Б. Н., Туз Ю. М. **Планирование и организация измерительного эксперимента:** Учеб. пособие. 18 л. Яз. рус. 1 р.

Приведены основные термины и определения теории вероятностей и математической статистики, традиционные методы проведения эксперимента. Рассмотрены методы дисперсионного и регрессионного анализа экспериментальных данных, планирования эксперимента в оптимальных условиях. Даны общие и структурные принципы построения систем автоматизации измерительного эксперимента и использования в них различных интерфейсов.

Для студентов, обучающихся по специальности «Информационно-измерительная техника».

Антипенский В. Е., Билоусько В. С. **Вычислительные машины и программирование.** Практикум: Учеб. пособие. 15 л. Яз. рус. 70 к.

Приведены упражнения для работы на клавишных вычислительных машинах, суммирующих, табличных (бухгалтерских и фактурных), вычислительных перфорационных, а также по математическим основам и программированию на электронных цифровых вычислительных машинах.

Для студентов экономических специальностей высших учебных заведений.

Арисава Макото. **Что такое компьютер?** 8 л. Яз. укр. 30 к.

Книга знакомит учащихся с компьютером (электронно-вычислительной машиной). Большое внимание уделено программному обеспечению работы компьютера, рациональному использованию заложенных в нем возможностей. Описано общее устройство и принцип действия машины, обращение с ней, дана краткая история возникновения и перспектива развития вычислительных машин.

Для учащихся физико-математических школ и учащихся старших классов общеобразовательных школ.

Автоматизация процессов подземных горных работ: Учебник / Руководитель авт. кол. д-р техн. наук А. А. Иванов. 26 л. Яз. рус. 1 р. 20 к.

Описаны принципы действия и конструкции технических средств автоматизации подземных горных работ, системы автоматического управления технологическими процессами шахт и рудников. Особое внимание уделено математическому описанию технических средств и систем. Рассмотрены вопросы их надеж-

ности, экономической эффективности, безопасности при автоматизации технологических процессов.

Для студентов горных вузов, обучающихся по специальности «Электрификация и автоматизация горных работ». Может быть полезен инженерам.

Книги ИО «Вища школа» можно заказать в любом магазине облкниготорга или облпотребсоюза, а также в магазине «Книга — почтой».