ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«ИНСТИТУТ ПОДГОТОВКИ НАУЧНЫХ КАДРОВ
НАЦИОНАЛЬНОЙ АКАДЕМИИ НАУК БЕЛАРУСИ»

ОСНОВЫ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Под общей редакцией
кандидата физико-математических наук, доцента
В.В. Шкурко

Учебно-методическое пособие для магистрантов и аспирантов
физико-математических и технических специальностей

Минск
2008

ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
"ИНСТИТУТ ПОДГОТОВКИ НАУЧНЫХ КАДРОВ
НАЦИОНАЛЬНОЙ АКАДЕМИИ НАУК БЕЛАРУСИ"

Кафедра информатики и вычислительной техники

ОСНОВЫ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Под общей редакцией
кандидата физико-математических наук, доцента
В.В. Шкурко

Учебно-методическое пособие
для студентов магистратуры, аспирантов и соискателей
ученых степеней физико-математических и технических наук

Минск
2008

УДК 004(075.8)
ББК 32.97я73
0-75

Рекомендовано к опубликованию Ученым советом
Института подготовки научных кадров НАН Беларуси
протокол № 10 от 28.12.2007

Авторы:
Н.В. Батин, И.Ф. Богданова, А.М. Ковальчук,
С.Ф. Липницкий, В.В. Шкурко

Рецензенты:
заведующий лабораторией Института физики НАН Беларуси,
доктор физико-математических наук А.Н. Чумаков,
доцент кафедры информационных технологий
автоматизированных систем БГУИР, кандидат технических
наук, доцент А.М. Севернев

Батин, Н.В.

0-75 Основы информационных технологий: учеб.-метод. пособие / Н.В. Батин [и др.] ; под общ. ред. В.В. Шкурко. - Минск : Ин-т подгот. науч. кадров Нац. акад. наук Беларуси, 2008. - 235 с.

ISBN 978-985-6820-13-0.

Пособие содержит теоретический и практический материал по дисциплине «Основы информационных технологий» для студентов магистратуры, аспирантов и соискателей физико-математических и технических специальностей в рамках подготовки к сдаче кандидатского дифференцированного зачета.

В пособии рассматриваются основные понятия информационных технологий, основы математического и компьютерного моделирования, развитые возможности табличного процессора Excel, пакетов компьютерной математики Matlab и Mathematica, а также основные понятия объектно-ориентированного программирования и принципы программирования на языке C++.

УДК 004(075.8)
ББК 32.97я73

ISBN 978-985-6820-13-0

О Г Л А В Л Е Н И Е

ВВЕДЕНИЕ

Пособие содержит теоретический и практический материал по дисциплине «Основы информационных технологий» для студентов магистратуры, аспирантов и соискателей физико-математических и технических специальностей. Предназначено главным образом для использования в качестве учебно-методического материала при подготовке к сдаче кандидатского дифференцированного зачета по указанной дисциплине. Основная цель как дисциплины «Основы информационных технологий», так и данного пособия - изучение студентами магистратуры, аспирантами и соискателями возможностей применения современных информационных технологий в своей научно-практической деятельности и, в частности, в диссертационном исследовании. В то же время пособие может представлять интерес и для других категорий читателей: от студентов до специалистов, изучающих современные компьютерные информационные технологии или применяющих эти технологии для решения научных и практических задач.

Пособие ориентировано на читателей, знакомых с основными понятиями высшей математики и программирования (в объеме соответствующих дисциплин, по любой физико-математической или технической специальности, изучаемой в вузе), имеющих навыки работы на компьютере в среде Windows 95 и выше, а также опыт работы с программами, входящими в состав пакета Microsoft Office. Материал в пособии изложен таким образом, что оно может использоваться читателями, имеющими хотя бы минимальный уровень математической и компьютерной подготовки.

В соответствии с программой дисциплины «Основы информационных технологий» для физико-математических и технических специальностей, в пособие включены следующие темы:

- **Основные понятия информационных технологий** (глава 1): рассматриваются основные виды и тенденции развития современных информационных технологий [1-8];
- **Основы математического и компьютерного моделирования** (глава 2): рассматриваются технологии компьютерного моделирования на основе метода Монте-Карло и методов нейроматематики [9-16];
- **Современные пакеты компьютерной математики** (главы 3-5): рассматриваются развитые возможности табличного процессора Excel [17, 18], пакетов компьютерной математики Matlab [19-21] и Mathematica [22-23] для решения широко распространенных математических задач (решение уравнений различных видов, решение задач оптимизации, статистический анализ данных и т.д.);
- **Основы объектно-ориентированного программирования** (глава 6): рассматриваются основные понятия объектно-ориентированного программирования, принципы программирования и примеры программ на языке C++ [24-29].

Пособие подготовлено на основе материалов лекционных и лабораторных занятий, проводившихся авторами для студентов магистратуры, аспирантов и

соискателей ученых степеней физико-математических и технических специальностей на кафедре информатики и вычислительной техники Института подготовки научных кадров НАН Беларуси в 2003-2007 годах. Контрольные вопросы и/или практические задания, как подробно рассмотренные, так и предлагаемые для самостоятельного выполнения, помогут применять пособие не только в качестве вспомогательного учебно-методического материала в дополнение к аудиторным занятиям, но и для самостоятельного изучения, а также в качестве лабораторного практикума.

Глава 1 подготовлена В.В. Шкурко и И.Ф. Богдановой (совместно), глава 2 - С.Ф. Липницким, главы 3-5 - В.В. Шкурко и Н.В. Батиным (совместно), глава 6 - А.М. Ковальчук.

В пособии нашли отражение не только опыт авторов, но и результаты аналитического обзора периодических изданий, литературных источников и интернет-ресурсов в области современных информационных технологий.

ГЛАВА 1

ОСНОВНЫЕ ПОНЯТИЯ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

1.1 Информация, её виды, формы и свойства

В науке и практике современного мира понятие «информация» является одним из самых распространенных. Несмотря на это, еще нет единого общепринятого определения информации. Множество различных точек зрения, продолжающиеся споры и дискуссии привели к выработке различных определений информации, что обусловлено сложностью, специфичностью и многообразием подходов к толкованию сущности этого понятия.

1.1.1 Генезис понятия информации

Слово «информация» происходит от латинского *informatio,* означающего разъяснение, изложение. Первоначально это были сведения, передаваемые одними людьми другим устным, письменным или каким-либо другим способом (например, с помощью условных сигналов, с использованием технических средств и т.д.), а также сам процесс передачи или получения этих сведений.

Понятие «информация» используется в различных смыслах. Так, говорят об информации в смысле соответствия высказывания действительности относительно определенных взаимосвязей, событий или состояний нашего реального мира. В науках, изучающих информацию, стараются более абстрактно определить ее понятие, которое становится не зависящим от реального мира.

К середине XX века в связи с бурным развитием науки и техники роль информации неизмеримо возросла. Кроме того, к этому периоду относится лавинообразное нарастание массы разнообразной информации, получившее название «информационного взрыва». В связи с этим возникла потребность в научном подходе к информации, выявлении ее наиболее характерных свойств, что привело к двум принципиальным изменениям в трактовке понятия информации.

Во-первых, оно было расширено и включило обмен сведениями не только между человеком и человеком, но и между человеком и автоматом, автоматом и автоматом; обмен сигналами в животном и растительном мире. Передачу признаков от клетки к клетке и от организма к организму тоже стали рассматривать как передачу информации.

Во-вторых, была предложена количественная мера информации — в работах К. Шеннона и А.Н. Колмогорова, — что привело к созданию *теории информации*.

Из множества предлагаемых определений информации назовем некоторые.

Информация — это сведения об объектах и явлениях окружающей среды, их параметрах, свойствах и состоянии, которые уменьшают имеющуюся о них степень неопределенности, неполноты знаний.

Информация — это совокупность фактов, явлений, событий, представляющих интерес, подлежащих регистрации и обработке.

Информация — нематериальная сущность, при помощи которой с любой точностью можно описывать реальные (материальные), виртуальные (возможные) и понятийные сущности. Информация — противоположность неопределенности.

В теории информации под этим термином понимается такое *сообщение,* которое содержит факты, неизвестные ранее потребителю и дополняющие его представление об изучаемом или анализируемом объекте (процессе, явлении).

Согласно концепции К. Шеннона, *информация* — это снятая неопределенность, т. е. сведения, которые должны снять в той или иной степени существующую у потребителя до их получения неопределенность, расширить его понимание объекта полезными сведениями.

1.1.2 Условия существования информации

Понятие информации обязательно объединяет двух участников - ее *источник* и *приёмник*. Для существования информации необходим также *канал приёма-передачи* информации. При взаимодействии этих трех компонентов появляется информация.

Информация существует только в виде *сообщения,* но не все сообщения содержат информацию. Для этого необходима некоторая *степень новизны* сообщения.

Сообщение — в теории коммуникации — предназначенные для передачи высказывание, текст, изображение, физический предмет или поступок. Сообщения состоят из словесных или невербальных знаков.

1.1.3 Виды информации

Все многообразие окружающей нас информации группируют по различным признакам.

По признаку «область возникновения» информация, отражающая процессы, явления неодушевленной природы называется *элементарной* или *механической,* животного и растительного мира — биологической, человеческого общества — социальной.

По способу передачи и восприятия различают информацию, передаваемую:

- видимыми образами и символами — визуальную;
- звуками — аудиальную;
- ощущениями - тактильную;
- запахами и вкусами - органолептическую;
- информацию, выдаваемую и воспринимаемую средствами вычислительной техники, — машинную.

Информацию, *создаваемую и используемую человеком,* по общественному назначению делят на три вида:

- личную, предназначаемую для конкретного человека;
- массовую, предназначаемую для любого, желающего ею пользоваться (общественно-политическая, научно-популярная);
- специальную, предназначаемую для использования узким кругом лиц, занимающихся решением сложных социальных задач в области науки, техники, экономики.

В зависимости от области знаний различают *научную, техническую производственную, правовую, патентную* и иную информацию. Каждый вид информации имеет свои особые смысловые нагрузки и ценности, свои требования к ее точности и достоверности, преимущественные технологии обработки, формы представления и носители.

1.1.4 Формы представления информации

В роли источников и приемников информации могут выступать самые разнообразные объекты науки и техники, общества и природы. Разнообразие источников и потребителей информации привело к существованию различных форм ее представления. Основные из них — *символьная* (основанная на использовании символов: букв, цифр, знаков); *текстовая* (использует тексты, т.е. символы, расположенные в определенном порядке); *графическая* (различные виды изображений), звуковая.

Символьная форма - является наиболее простой, но практически она применяется только для передачи несложных сигналов о различных событиях. Примером может служить разный свет светофора, сообщающий о возможности начала или прекращения движения пешеходам или водителям автотранспорта; милицейский свисток, который дает информацию о действиях, которые должен предпринять человек, которому «адресован» этот звуковой сигнал и т.д.

Текстовая форма представления информации является более сложной. Она также использует различные символы: буквы, цифры, математические знаки. Но информация заложена не только в этих символах, но и в их сочетании, в порядке следования. Так, слова «кот» и «ток» состоят из одинакового количества одинаковых букв, но содержат различную информацию. Благодаря тому, что текстовая информация является изображением речи, она чрезвычайно удобна и широко используется (книги, журналы, газеты, различные документы, аудиозаписи).

Графическая форма представления информации является наиболее сложной. Сюда относятся фотографии, схемы, рисунки, чертежи, имеющие большое значение в деятельности человека.

1.1.5 Качественные характеристики информации

Качество информации — совокупность свойств, отражающих степень пригодности конкретной информации об объектах и их взаимосвязях для достижения ее пользователем целей деятельности.

Возможность и эффективность использования информации обусловливаются такими ее потребительскими *показателями качества,* как репрезентативность, содержательность, достаточность, доступность, актуальность, своевременность, точность, достоверность, устойчивость.

Репрезентативность информации связана с правильностью ее отбора и формирования в целях адекватного отражения свойств объекта. Важнейшее значение здесь имеют:

- правильность концепции, на базе которой сформулировано исходное понятие;
- обоснованность отбора существенных признаков и связей отображаемого явления.

Нарушение репрезентативности отбора информации нередко приводит к ее существенным погрешностям.

Содержательность информации отражает семантическую емкость, равную отношению количества семантической информации в сообщении к объему обрабатываемых данных.

С увеличением содержательности информации возрастает семантическая пропускная способность информационной системы, так как для получения одних и тех же сведений требуется преобразовывать меньший объем данных.

Достаточность (полнота) информации означает, что она содержит минимальный, но достаточный для принятия правильного решения состав (набор показателей). Понятие полноты информации связано с ее смысловым содержанием (семантикой) и прагматикой. Как неполная, т. е. недостаточная для принятия правильного решения, так и избыточная информация снижает эффективность принимаемых пользователем решений.

Доступность информации восприятию пользователя обеспечивается выполнением соответствующих процедур ее получения и преобразования. Например, в информационной системе информация преобразуется в доступную и удобную для восприятия пользователя форму. Это достигается, в частности, путем согласования ее семантической формы с тезаурусом пользователя.

Актуальность информации определяется степенью сохранения ее ценности для управления в момент использования и зависит от динамики изменения ее характеристик и от интервала времени, прошедшего с момента возникновения данной информации.

Своевременность информации означает ее поступление не позже заранее назначенного момента времени, согласованного со временем решения поставленной задачи.

Точность информации определяется степенью ее близости к реальному состоянию объекта, процесса, явления и т.п.

Достоверность информации определяется ее свойством отражать реально существующие объекты с необходимой точностью. Измеряется достоверность информации доверительной вероятностью необходимой точности, т. е. вероятностью того, что отображаемое информацией значение параметра отличается от истинного значения этого параметра в пределах необходимой точности.

• *Устойчивость* информации отражает ее способность реагировать на изменения исходных данных без нарушения необходимой точности. Устойчивость информации, как и репрезентативность, обусловлена выбранной методикой ее отбора и формирования

1.1.6 Свойства информации

Свойства информации рассматривают в трех аспектах:

- техническом — когда важны точность, надежность, скорость передачи сигналов;
- семантическом — на первом плане передача смысла текста с помощью кодов;
- прагматическом — оценивается эффективность влияния информации на поведение объекта.

1.2 Науки, изучающие информацию

Возникновение в середине XX века целого ряда наук, основным объектом исследования которых стала информация — одно из важнейших событий научного мира. Рассмотрим основные науки этой области знаний.

1.2.1 Теория информации

Теория информации представляет собой математическую теорию, посвященную измерению информации, ее потока, «размеров» канала связи, ее кодированию и декодированию.

Оформление этого научного направления связывают с именем выдающегося американского инженера и математика Клода Шеннона, опубликовавшего в 1948 г. монографию «Математическая теория связи». Значительный вклад в формирование теории информации внесли также многие ученые, среди которых необходимо отметить советских математиков Андрея Николаевича Колмогорова и Александра Яковлевича Хинчина.

Первоначально теория информации была посвящена каналу связи, определяемому длиной волны и частотой, реализация которого была связана с колебаниями воздуха или электромагнитным излучением. Соответствующий процесс обычно является непрерывным, но может быть и дискретным, если

информация кодируется, а затем декодируется. Клод Шеннон предложил вероятностный подход к измерению дискретной и непрерывной информации.

Теория информации изучает также методы построения кодов, обладающих полезными свойствами.

Базовыми понятиями теории информации являются: информация, канал связи, шум, кодирование.

Канал связи — это среда передачи информации, которая характеризуется в первую очередь максимально возможной для нее скоростью передачи данных (ёмкостью канала связи).

Шум — это помехи в канале связи при передаче информации.

Кодирование — преобразование дискретной информации одним из следующих способов: шифрование, сжатие, защита от шума.

Теория информации тесно связана с такими разделами математики, как теория вероятностей и математическая статистика, а также прикладная алгебра, которые предоставляют для нее математический фундамент. С другой стороны, теория информации исторически и практически представляет собой математический фундамент теории связи. Часто теорию информации вообще рассматривают как одну из ветвей теории вероятностей или как часть теории связи.

В настоящее время *теория информации* — это раздел кибернетики, в котором математическими методами изучаются способы измерения количества информации, содержащейся в каких-либо сообщениях, и передачи информации.

1.2.2 Кибернетика

В греческом языке слово «кюбернетес», от которого произошел термин «кибернетика», вначале означавшее «рулевой» или «кормчий», позднее стало обозначать также и «правитель над людьми». Предполагают, что первым этот термин использовал древнегреческий философ *Платон* в смысле искусства управления кораблём или колесницей. Известны также другие его сочинения, в которых он называет кибернетикой искусство управлять людьми.

Известный французский физик и математик, один из основоположников электродинамики, *Андре Мари Ампер* в работе «Опыт о философии наук, или Аналитическое изложение естественной классификации всех человеческих знаний», первая часть которой вышла в 1834 г., высказал мысль о том, что в будущем, вероятно, возникнет новая наука об общих закономерностях процессов управления. Он предложил именовать ее «кибернетикой».

В дальнейшем термин «кибернетика» употреблялся очень редко и был возрожден в современном его значении в 1948 г. американским математиком *Норбертом Винером,* не знавшим о более ранних его употреблениях, в качестве названия науки об управлении техническими, биологическими и социальными системами.

Основоположник современной кибернетики Норберт Винер фактически расширил толкование понятия « кибернетика», определив ее как науку об общих принципах управления динамическими объектами, включая разные

области человеческой деятельности. Важный вклад в формирование кибернетики как науки внес также К. Шеннон.

В 1948 г. Норберт Винер, труды которого по математической логике легли в основу зарождавшихся тогда вычислительной техники и программирования, опубликовал книгу «Кибернетика или управление и связь в животном и машине». Этот год и считается датой рождения кибернетики как современной науки.

Норберт Винер пытался найти удобный способ объединения различных наук, относящихся к коммуникациям и управлению, под одним именем, которое отражало бы их методологическое единство. Это единство зиждется на статистической идее информации как функции вероятности.

Кибернетика — это наука об управлении, связи и переработке информации. Основным объектом исследования кибернетики являются абстрактные кибернетические системы.

Примерами кибернетических систем являются автоматические регуляторы в технике, компьютеры, человеческий мозг, биологические популяции, человеческое общество. Каждая такая система представляет собой множество взаимосвязанных объектов, способных воспринимать, запоминать и перерабатывать информацию, а также обмениваться ею.

Существует большое количество различных определений понятия *«кибернетика»,* однако все они в конечном счете сводятся к тому, что кибернетика — это наука, изучающая общие закономерности строения сложных систем управления и протекания в них процессов управления. А так как любые процессы управления связаны с принятием решений на основе получаемой информации, то кибернетику часто определяют еще и как науку об общих законах получения, хранения, передачи и преобразования информации в сложных управляющих системах.

Н. Винер ввел основную категорию кибернетики — *управление,* показал существенные отличия этой категории от других, например, энергии; описал несколько задач, типичных для кибернетики, и привлек всеобщее внимание к особой роли вычислительных машин, считая их индикатором наступления новой научно-технической революции. Выделение категории управления позволило Винеру воспользоваться понятием информации, положив в основу кибернетики изучение законов передачи и преобразования информации.

Сущность *принципа управления* заключается в том, что движение и действие больших масс или передача и преобразование больших количеств энергии направляются и контролируются при помощи небольших количеств энергии, несущих информацию. Этот принцип управления лежит в основе организации и действия любых управляемых систем: автоматических устройств, живых организмов и др. Подобно тому как введение понятия энергии позволило рассматривать все явления природы с единой точки зрения и отвергло целый ряд ложных теорий, так и введение понятия информации позволило прийти к единой точке зрения на методику изучения самых различных процессов взаимодействия в природе.

В то же время первоначальное определение Н. Винером кибернетики как науки об управлении в живой природе и в технических системах оказалось

весьма спорным: объединение живой природы и технических систем в одной дисциплине привело к активному неприятию такого определения учеными многих стран.

В большинстве стран мира, где кибернетика привилась, дискуссия по этому вопросу привела к расколу в научных кругах на два лагеря. В США этому способствовала яростная критика книг и идей Винера. В Великобритании против расширенного истолкования кибернетики выступили физики, во Франции — математики.

Особенно резкой критике подверглась зарождавшаяся кибернетика в Советском Союзе, где в 50-е гг. она была объявлена лженаукой и формально запрещена, что не мешало, однако, развиваться всем ее важным разделам (в том числе и теории информации) вне связи с обобщающим понятием «кибернетика».

Основателем советской кибернетики считается член-корреспондент АН СССР А. А. Ляпунов. Кроме него в СССР значительный вклад в развитие кибернетики внесли многие ученые, среди которых самыми знаменитыми были академики А. И, Берг, А. Н. Колмогоров, В. М. Глушков.

Выдающиеся советские ученые, понимавшие, что запрет кибернетики как науки неминуемо приведет к ослаблению государства в целом, боролись за снятие с кибернетики ярлыка лженауки и приобретение ею статуса официально признанной науки.

Общественное движение «за кибернетику» возглавили ведущие советские ученые А. А. Ляпунов, С. Л. Соболев, А. Н. Колмогоров, А. И. Берг.

А. И. Китов и А. А. Ляпунов организовали серию выступлений на научных семинарах в академических институтах, высших учебных заведениях и в организациях, в которых методы кибернетики могли бы принести практическую пользу. К этой деятельности подключились их коллеги по работе в Вычислительном центре Министерства обороны и других военных организациях: М. Г. Гаазе-Рапопорт, Н. А. Криницкий, И. А. Полетаев и другие. В Московском университете идеи кибернетики нашли отклик у признанного в СССР авторитета в области математической логики А. А. Маркова, а в Институте автоматики и телемеханики эти работы были поддержаны М. А. Айзерманом, М. А. Гавриловым и А. А. Фельдбаумом.

Особенно большую роль в координации работ и формировании новых направлений исследований сыграл междисциплинарный семинар по кибернетике, организованный А. А. Ляпуновым в МГУ в 1956 г. Семинар объединил ученых различных специальностей. Он стал центром зарождения кибернетической мысли в Советском Союзе.

Большую работу по разъяснению действительного предназначения кибернетики провел академик А.И. Берг. Он выступал с лекциями, которые часто записывали на магнитофон и потом распространяли подобно выступлениям бардов. Эти лекции пользовались огромным успехом, особенно у молодежи.

В 1958 г. вышла в свет книга И. А. Полетаева «Сигнал» — первая советская монография по кибернетике, где не только излагались идеи и принципы кибернетики, но и использовался сам термин «кибернетика». Книга

сразу же привлекла внимание ученых всего мира и была многократно переведена и издана за рубежом. Эта работа не утратила своего значения и сегодня.

К 1958 г. борьба за признание кибернетики в основном была завершена. Завершающими актами борьбы выдающихся советских ученых за признание кибернетики стали опубликование в 1958г. в Большой Советской Энциклопедии (БСЭ) статьи А.Н. Колмогорова «Кибернетика», раскрывающей истинную суть этой науки, и учреждение в январе 1959 г. научного Совета по комплексной проблеме «Кибернетика» при Президиуме АН СССР. Совет возглавил инженер-адмирал и академик А. И. Берг.

В конце 50-х гг. советскими учеными был получен ряд результатов, соответствующих уровню мировых достижений. Наступил начальный период свободного и ускоренного развития советской кибернетики.

В это же время началась активная деятельность по созданию международных научных ассоциаций. Была сделана попытка образовать Международную федерацию кибернетики, не увенчавшаяся, однако, успехом. Отсутствие в ней специалистов США и Великобритании не позволило этой ассоциации занять сколько-нибудь заметное место в мире. Значительно большее влияние приобрели две другие федерации. В 1957 г. возникла ИФАК— Международная федерация по автоматическому управлению, а в 1962 г. — ИФИП — Международная федерация по процессам переработки информации, ставшая преемницей Международной федерации кибернетики.

Кибернетический подход к сложным объектам, заключающийся в рассмотрении их как больших кибернетических систем, состоящих из управляющих и управляемых элементов, между которыми существуют информационные прямая и обратная связи, привел к возникновению системного анализа — научного метода исследования структуры и функционирования сложных явлений, процессов и объектов природы и общества. Результатом анализа является математическое описание способов управления и распространения информации в системе. Анализ используется не только для моделирования управления системами и прогнозирования их поведения, но и для синтеза новых систем.

Особенно широко кибернетический подход стал применяться в связи с созданием и развитием электронных вычислительных машин. Это привело к образованию ряда научных направлений кибернетики.

В зависимости от области применения различают несколько видов кибернетики. Основными из них являются:

- общая (теоретическая) кибернетика,
- военная кибернетика,
- техническая кибернетика,
- биологическая кибернетика,
- медицинская кибернетика,
- экономическая кибернетика,
- социальная кибернетика.

В настоящее время часто кибернетику связывают с методами искусственного интеллекта, поскольку она разрабатывает общие принципы создания систем управления и систем автоматизации умственного труда. Основными разделами (они фактически абсолютно самостоятельны и независимы) современной кибернетики считаются:

- теория информации,
- теория алгоритмов, теория автоматов,
- исследование операций,
- теория оптимального управления,
- теория распознавания образов,
- теория формальных языков.

Стремительное развитие вычислительной техники вызвало большой интерес к кибернетике в 60-70-е гг. XX века и ее бурное развитие во всем мире. В 80-90-е гг. термин *«кибернетика»* был частично вытеснен термином *«информатика»,* имеющим отношение, прежде всего, к компьютерам и обработке информации. Однако в последние годы *кибернетика* вновь стала популярной в связи с развитием Интернета (киберпространство) и робототехники.

1.2.3 Информатика

Информатика — это наука и техника, связанные с машинной обработкой, хранением и передачей информации. Она занимается схематичным, «формализованным» представлением информации, ее обработкой, равно как и предписаниями по ее переработке, и машинами, обрабатывающими информацию.

Термин «информатика» возник в 60-х гг. XX века во Франции для названия автоматизированной обработки информации с помощью электронных вычислительных машин. Французский термин informatigue (информатика) образован путем слияния слов information (информация) и automatigue (автоматика) и означает: «информационная автоматика» или «автоматизированная обработка информации». В англоязычных странах этому термину соответствует синоним *«computer science»* (компьютерная наука).

Термин «информатика» в 60-х гг. XX века иногда использовался в русском языке для обозначения дисциплины, связанной с технологией накопления научно-технической и другой информации на основе печатных литературных источников и документов. Однако это словоупотребление осталось эпизодическим и сменилось на теперь уже общепринятое значение, связанное с информационными процессами, реализуемыми только с помощью электронных вычислительных машин (ЭВМ). В 1968 г. в Вычислительном центре Сибирского отделения Академии наук СССР под руководством А.П. Ершова было организовано Отделение информатики, объединяющее работы по теоретическому и системному программированию, а также по пакетам прикладных программ. Считается, что это было первое в Советском

Союзе официальное употребление слова «информатика» в его современном понимании.

Академику А.П. Ершову принадлежит следующее определение информатики: *«Информатика* — это фундаментальная естественная наука, изучающая процессы передачи и обработки информации».

Выделение информатики в отдельную область человеческой деятельности в первую очередь связано с развитием компьютерной техники.

Причем основная заслуга здесь принадлежит микропроцессорной технике, появление которой в середине 70-х гг. XX века положило начало второй электронной революции. С этого времени элементной базой вычислительной машины становятся интегральные схемы и микропроцессоры, а область, связанная с созданием и использованием компьютеров, получила мощный импульс развития. Сегодня термин *«информатика»* используется не только для отображения достижений компьютерной техники, но и связывается с процессами передачи и обработки информации.

В 70-е гг. XX века информатика начала формироваться как дополнение и конкретизация кибернетики в связи с использованием ЭВМ в науке, управлении, проектировании, образовании, сфере услуг и т.д.

В Советском Союзе подобная трактовка термина «информатика» утвердилась с момента принятия в 1983 г. Академией наук СССР решения об организации нового Отделения информатики, вычислительной техники и автоматизации. Информатика трактовалась как "комплексная научная и инженерная дисциплина, изучающая все аспекты переработки, проектирования, создания, оценки, функционирования основанных на ЭВМ систем переработки информации, их применения и воздействия на различные области социальной практики". Информатика в таком понимании нацелена на разработку общих методологических принципов построения информационных моделей. Поэтому методы информатики применимы всюду, где есть возможность описания объекта, явления, процесса и т.п. с помощью информационных моделей.

Существует множество определений информатики, что связано с многогранностью ее функций, возможностей, средств и методов. Приведем трактовку этого термина, являющуюся обобщением опубликованных в литературе по информатике определений:

Информатика — это область человеческой деятельности, связанная с процессами преобразования информации с помощью компьютера и их взаимодействием со средой применения.

Информатика занимается изучением процессов преобразования и создания новой информации в широком смысле слова, практически не решая задач управления различными объектами.

Информатика не занимается решением проблем, не связанных с использованием компьютерной техники. Информатика появилась благодаря развитию компьютерной техники, базируется на ней и совершенно немыслима без нее.

В *информатику* входит группа дисциплин, занимающихся различными вопросами, связанными с разработкой и применением вычислительной

техники: прикладная математика, программирование, искусственный интеллект, архитектура ЭВМ, вычислительные сети и др.

Современная прикладная информатика занимается специальными информационными системами, основанными на ЭВМ и реализующими машинные информационные технологии. Эти системы подразделяются на управленческие, административные, исследовательские, учебные, проектирующие, коммуникационные, системы обслуживания бытовой сферы, экологические, военные, медицинские и др.

Современная информатика охватывает все аспекты разработки, внедрения и влияния на развитие общества специальных информационных систем.

Развитие вычислительной техники позволило информатике перейти от изучения и разработки систем обработки данных к системам обработки знаний.

1.2.4 Информационные технологии

Понятие «технология» в процессе эволюции общества претерпевало естественные изменения, связанные с развитием производительных сил, т.е. с новыми этапами развития общества.

Древние греки считали, что технология (гр. *techne* — мастерство, искусство и *logos* — учение) — это мастерство (искусство) делать вещи.

Более ёмкое определение это понятие приобрело в процессе индустриализации общества. Существует множество современных определений понятия «технология».

Сегодня считается, что результатом технологического процесса может быть не только материальный объект, но и услуга, а само это понятие рассматривается с учетом его комплексности, системности современной технологии. Технология как строго научное понятие означает комплекс научных и инженерных знаний, воплощенных в способах, приёмах труда, наборах материально-вещественных факторов производства, способах их соединения для создания какого-либо продукта или услуги.

Таким образом, можно утверждать, что понятие «технология» применимо к любому процессу, имеющему искусственное происхождение.

Все сказанное относится к понятию технологии, интерпретируемому в *широком* смысле.

В *узком* смысле под технологией можно понимать набор способов, средств выбора и осуществления управляющего процесса, выделенный из множества возможных реализаций этого процесса.

Технологии управляемых процессов свойственны упорядоченность и организованность, которые противопоставляются стихийным процессам.

Необходимо отметить, что фактически технология появляется почти одновременно с каким-либо новым родом человеческой деятельности, однако такие ее характеристики, как целостность, комплексность, функциональная полнота, целесообразность, регламентированность, стандартизация и унификация процессов и средств присущи развитым технологиям, что в свою очередь требует соответствующих средств реализации.

Исторически обусловлено, что термин «технология» возник в сфере материального производства, развит и глубоко исследован применительно к производственным (промышленным) системам, поэтому, с одной стороны, обобщенная его трактовка основана именно на свойствах производственных технологий, с другой — производственная его интерпретация является конструктивной базой для определения новых, появляющихся в настоящее время технологий.

Цель технологии материального производства — выпуск продукции, удовлетворяющей потребности человека или системы.

Поскольку можно отметить некоторое сходство процессов обработки материальных и информационных объектов, уместна и аналогия в определении понятия информационных технологий.

Создание сложных кибернетических систем, а также развитие методов и средств анализа и реализации программного и информационного обеспечения, их функционирования с необходимостью обусловили осознание существования и развития новых видов технологий: технологии программирования и информационной технологии.

Как и любые вновь вводимые понятия, технология программирования и информационная технология (ИТ) разными авторам определяются с различными акцентами, причем определения содержат достаточно общего (чтобы сделать вывод о том, что определяется одно и то же понятие) и различного (чтобы оценить индивидуальность авторов).

Приведем несколько определений.

Информационная технология — это совокупность процессов сбора, передачи, переработки, хранения и доведения до пользователя информации, реализуемых на современных средствах. Информационная технология это, во-первых, совокупность процессов циркуляции и переработки информации и, во-вторых, описание этих процессов.

Информационная технология — это процесс, использующий совокупность средств и методов сбора, обработки и передачи данных (первичной информации) для получения информации нового качества о состоянии объекта, процесса или явления (информационного продукта).

Информационные технологии — это технологии, ориентированные на получение, обработку и распространение (передачу) информации.

Цель информационной технологии - производство информации для ее анализа человеком и принятия на его основе решения о выполнении какого-либо действия.

Выделяют следующие основные характеристики информационной технологии:

- предметом обработки (процесса) являются данные;
- целью процесса является получение информации;
- средствами осуществления процесса являются программные, аппаратные и программно-аппаратные вычислительные комплексы;

- процессы обработки данных разделяются на операции в соответствии с данной предметной областью;
- выбор управляющих воздействий на процессы должен осуществляться лицами, принимающими решения (ЛПР);
- критерием оптимизации процесса являются своевременность доставки информации пользователю, ее надежность, доступность, полнота.

1.2.5 Новая информационная технология

К настоящему времени информационная технология прошла несколько эволюционных этапов, смена которых определялась, главным образом, появлением новых технических средств переработки и передачи информации. В современном обществе основным техническим средством переработки информации служит персональный компьютер, который существенно повлиял как на концепцию построения и использования технологических процессов, так и на качество получаемой информации. Внедрение персонального компьютера в информационную сферу и применение телекоммуникационных средств связи определили новый этап развития информационной технологии и, как следствие, изменение ее названия за счет присоединения одного из синонимов: «новая», «компьютерная», «современная». В понятие новой информационной технологии включены также коммуникационные технологии, которые обеспечивают передачу информации разными средствами, а именно: с помощью телефона, телекоммуникаций, факса и др.

Новая информационная технология — это информационная технология с «дружественным» интерфейсом работы пользователя, использующая персональные компьютеры и телекоммуникационные средства.

Телекоммуникация — дистанционная передача данных на базе компьютерных сетей и современных технических средств связи.

Три основных принципа новой (компьютерной) технологии:

- интерактивный (диалоговый) режим работы с компьютером;
- интегрированность (стыковка, взаимосвязь) с другими программными продуктами;
- гибкость процесса изменения как данных, так и постановок задач.

1.2.6 Этапы развития информационных технологий

Этапы развития информационных технологий различаются:

По виду задач и процессов обработки информации:

- первый этап (1960-1970-е гг.) — обработка данных в вычислительных центрах в режиме коллективного пользования. Основным направлением развития информационной технологии

являлась автоматизация операционных рутинных действий человека;
- второй этап (с 1980-х гг.) - создание информационных технологий, направленных на решение стратегических задач.

По проблемам, стоящим на пути информатизации общества:

- первый этап (до конца 60-х гг.) характеризуется проблемой обработки больших объемов данных в условиях ограниченных возможностей аппаратных средств;
- второй этап (до конца 70-х гг.) связывается с распространением ЭВМ серии ЮМ/360. Проблема этого этапа — отставание программного обеспечения от уровня развития аппаратных средств;
- третий этап (с начала 80-х гг.) — компьютер становится инструментом непрофессионального пользователя, а информационные системы — средством поддержки принятия его решений. Проблемы — максимальное удовлетворение потребностей пользователя и создание соответствующего интерфейса работы в компьютерной среде;
- четвертый этап (с начала 90-х гг.) — создание современной технологии межорганизационных связей и информационных систем. Проблемы этого этапа весьма многочисленны. Наиболее существенными из них являются:
    - выработка соглашений и установление стандартов, протоколов для компьютерной связи;
    - организация доступа к стратегической информации;
    - организация защиты и безопасности информации.

1.3 Виды компьютерного обеспечения

Для функционирования компьютера необходимы как аппаратное обеспечение (АО), *(hardware),* так и программное обеспечение (ПО), *(software).*

Оба вида компьютерного обеспечения одинаково значимы. Цены на них на мировом рынке, как правило, соизмеримы. Программы отличаются лишь тем, что их легко скопировать, именно поэтому на программном рынке имеет место программное пиратство.

1.3.1 Аппаратное (техническое) обеспечение

Аппаратное обеспечение — это прежде всего компьютеры. Компьютеры могут работать со множеством дополнительных устройств, которые тоже относятся к аппаратуре (мышь, сканер, принтер и т.д.). Компьютеры могут также работать в сетях — локальных, региональных, глобальных. Для этого требуется сетевое и коммуникационное оборудование: сетевые платы, модемы, концентраторы, кабели, источники бесперебойного питания и многое другое.

«Компьютер» — достаточно широкое понятие, подразумевающее несколько классов разнообразных электронных вычислительных устройств. Приняты различные классификации современной компьютерной техники:

- по этапам развития (поколениям);
- по архитектуре;
- по условиям эксплуатации;
- по производительности;
- по количеству процессоров;
- по потребительским свойствам и др.

Четких границ между классами компьютеров не существует. По мере совершенствования структур и технологии производства появляются новые классы компьютеров, границы существующих классов существенно изменяются.

По условиям эксплуатации компьютеры делятся на два типа:

- офисные (универсальные или общего назначения);
- специализированные (специальные).

Офисные предназначены для решения широкого класса задач при нормальных условиях эксплуатации.

Специализированные компьютеры служат для решения более узкого класса задач или даже одной задачи, требующей многократного решения, и функционируют в особых условиях эксплуатации. Машинные ресурсы специальных компьютеров часто ограничены. Однако их узкая ориентация позволяет реализовать заданный класс задач наиболее эффективно.

Специализированные компьютеры управляют технологическими установками, работают в операционных отделениях или машинах скорой помощи, на ракетах, самолётах и вертолётах, вблизи высоковольтных линий передач или в зоне действия радаров, радиопередатчиков, в неотапливаемых помещениях, под водой (на глубине), в условиях пыли, грязи, вибраций, взрывоопасных газов и т.п.

По производительности и характеру использования компьютеры можно условно подразделить на *суперкомпьютеры, персональные компьютеры и мобильные компьютеры*.

Суперкомпьютеры предназначены для решения задач, требующих огромных вычислений, высокой скорости расчетов, больших объемов памяти. Это очень мощные компьютеры с производительностью свыше 100 мегафлопов (1 мегафлоп — миллион операций с плавающей точкой в секунду). Они называются *сверхбыстродействующими*. Эти машины представляют собой *многопроцессорные* и (или) *многомашинные* комплексы, работающие на общую память и общее поле внешних устройств.

Суперкомпьютеры используются для решения сложных и объемных научных задач (метеорология, гидродинамика и т. п.), расчетов космических полетов, в управлении, в качестве централизованных хранилищ информации и т.д.

К этой же группе относятся и *серверы* — компьютеры, предназначенные для «обслуживания» других компьютеров в различных вычислительных сетях. Серверы могут иметь разную производительность, и самые мощные из них приближаются к суперкомпьютерам.

Персональные компьютеры (ПК) — это микрокомпьютеры универсального назначения, рассчитанные на одного пользователя и управляемые одним человеком, которые могут работать самостоятельно или в составе сети.

В свою очередь персональные компьютеры конструктивно делятся на две группы — *настольные и переносные (портативные).* В зависимости от размера переносные компьютеры принято называть *Laptop* (наколенный компьютер) — массой 5-7 кг , Notebook (размером с книгу) — массой 2-4 кг и Subnotebook — массой менее 2 кг.

Переносные компьютеры представляют собой компактные компьютеры, содержащие все необходимые компоненты (в том числе монитор) в одном небольшом корпусе, как правило складывающемся в виде книжки. Эти компьютеры приспособлены для работы в дороге, на небольшом свободном пространстве. Для достижения малых размеров в них применяются специальные технологии.

Они имеют развитые средства подключения к проводным и беспроводным сетям, встроенное мультимедийное оборудование (динамики, часто микрофон и веб-камеру). Вычислительная мощность их лишь незначительно уступает настольным ПК, однако объем памяти и накопителей может быть меньше. Некоторые очень компактные модели не содержат CD/DVD-накопителя.

Переносные компьютеры обычно нужны специалистам, которым приходится работать вне офиса: дома, на презентациях или во время командировок. Клавиатура переносных компьютеров в отличие от стационарных менее удобна в работе, в результате чего ввод информации в этих компьютерах осуществляется значительно медленнее, чем при работе с обычной клавиатурой.

К переносным ПК относят и *планшетные,* которые аналогичны ноутбукам, но содержат чувствительный к нажатию экран и не имеют механической клавиатуры и мыши. Ввод текста и управление осуществляются через экранный интерфейс, часто доработанный специально для удобного управления пальцами. Информация может вводиться также с помощью стилуса (специального пера). Некоторые модели могут распознавать рукописный текст, написанный на экране.

В планшетных ПК корпус чаще всего не раскрывается, как у ноутбуков, а экран расположен на внешней стороне верхней поверхности. Существуют комбинированные модели переносных ПК, у которых корпус может поворачиваться на оси и раскрываться, предоставляя доступ к расположенной внутри клавиатуре.

Вычислительная мощь этих компьютеров уступает настольным ПК, так как для длительной работы без внешнего источника питания приходится использовать энергосберегающие процессоры, накопители и экран.

Мобильные компьютеры, или персональные цифровые помощники (PDA — Personal Digital Assistant). Это сверхпортативные устройства, умещающиеся в кармане и всегда находящиеся под рукой. Этот класс компьютеров называют также наладонниками (Palmtop PC) и карманными ПК (Pocket PC). Это самые маленькие современные персональные компьютеры, умещаются на ладони и в кармане, отсюда и их названия.

Управление такими компьютерами, как правило, осуществляется с помощью небольшого по размерам и разрешению экрана, чувствительного к нажатию пальца или специального пера (стилуса), а клавиатура и мышь, чаще всего, отсутствуют. Однако существуют и модели, которые содержат миниатюрную фиксированную или выдвигающуюся из корпуса клавиатуру.

Мобильные компьютеры обычно используют для чтения книг, различных справочников, работы с базами данных, электронной почтой, всевозможных записей, выхода в Интернет при наличии модуля GPS (глобальной системы позиционирования) и специальных программ для музейной и туристической навигации. Мобильные компьютеры могут также иметь встроенные фотоаппарат или видеокамеру, функцию телефона с возможностями отправки SMS, MMS и звонками и многое другое.

В настоящее время все большее применение и распространение получают *смартфоны* (от англ. smartphone — интеллектуальный телефон) — устройства, совмещающее функции мобильного телефона и карманного персонального компьютера.

Основным отличием смартфона от мобильного телефона является наличие достаточно развитой операционной системы, что дает возможность устанавливать дополнительное программное обеспечение, расширяющее возможности данного устройства

Еще одной разновидностью современных мобильных компьютеров являются *коммуникаторы*. Это карманные персональные компьютеры, оснащенные функциями GSM-связи. Коммуникаторы позволяют не только совершать звонки, но и подключаться к Интернету.

В настоящее время существует тенденция размывания границ между понятиями «смартфон» и «коммуникатор».

Каждая группа современных компьютеров развивается самостоятельно и тенденции развития у них индивидуальные.

1.3.2 Программное обеспечение

Программное обеспечение (ПО) — это совокупность программ обработки данных и необходимых для их эксплуатации документов.

При построении классификации ПО необходимо учитывать тот факт, что стремительное развитие вычислительной техники и расширение сферы приложения компьютеров резко ускорили процесс эволюции программного обеспечения. В связи с этим существуют различные классификации программного обеспечения.

Следует иметь в виду, что ни одну из этих классификаций нельзя считать исчерпывающей, однако они, как правило, более или менее наглядно отражают

современные тенденции развития и совершенствования программного обеспечения.

В настоящем пособии рассматриваются два вида классификации программного обеспечения: по назначению и по способу распространения.

По назначению программное обеспечение разделяется на *системное ПО, прикладное ПО и инструментальное обеспечение разработки программ.*

Системное программное обеспечение представляет собой совокупность взаимосвязанных программ, которые обеспечивают функционирование средств вычислительной техники как таковых без выполнения операций по реализации функций офисных технологий. Системные программы абсолютно необходимы для работы. Они, в сущности, являются продолжением аппаратного обеспечения компьютера.

Системное программное обеспечение подразделяется на *базовое и сервисное.*

Базовое ПО включает:

- операционные системы;
- командно-файловые процессоры (операционные оболочки);
- системные утилиты.

Операционная система — это совокупность программных средств, обеспечивающая управление аппаратной частью компьютера и прикладными программами, а также их взаимодействие между собой и пользователем.

Командно-файловые процессоры или операционные оболочки — это специальные программы, предназначенные для облегчения общения пользователя с командами операционной системы (например, Norton Commander, Total Commander). Операционные оболочки имеют текстовый и графический варианты пользовательского интерфейса.

Системные утилиты (от латин. utilitas — польза) — программы, служащие для вспомогательных операций обработки данных или обслуживания компьютера (диагностика, тестирование аппаратных или программных средств, оптимизация использования дискового пространства, восстановления разрушенной на магнитном диске информации и т.д.). Часть утилит входит в состав операционной системы, а часть функционирует независимо от нее, т.е. автономно.

Подмножеством системного программного обеспечения являются *сервисные* программы, которые не являются такими жизненно важными, как ОС, но также помогают управлять компьютером и оптимизировать использование его ресурсов.

Сервисное программное обеспечение включает в себя следующие программы (утилиты):

- диагностики;
- антивирусные;
- обслуживания носителей;
- архивирования;
- обслуживания сети.

Прикладное программное обеспечение представляет собой совокупность программных комплексов, обеспечивающих решение конкретных задач при реализации тех или иных функций офисных технологий. Прикладное программное обеспечение работает только при наличии системного программного обеспечения.

Прикладное программное обеспечение включает *пакеты прикладных программ,* которые называют также *приложениями,* и *прикладные программы пользователей.*

Пакеты прикладных программ (ППП) — это комплекс взаимосвязанных программ для решения задач определенного класса конкретной предметной области. ППП могут быть *общего* или *специального назначения.*

К пакетам прикладных программ общего назначения относятся:

- текстовые процессоры;
- табличные процессоры;
- системы управления базами данных;
- графические редакторы и др.

Пакеты прикладных программ специального назначения ориентированы на решение задач в определенной предметной области. К ним относятся:

- пакеты компьютерной математики для научно-технических расчетов;
- пакеты моделирования и анализа;
- пакеты статистической обработки данных;
- экспертные системы;
- обучающие программы;
- бухгалтерские и экономические пакеты и др.

Прикладные программы пользователей служат для решения практических задач в различных предметных областях. Они автоматизируют практически все виды человеческой деятельности.

Третий вид программного обеспечения — это *инструментальное обеспечение разработки программ,* включающее различные системы программирования, с помощью которых могут разрабатываться и адаптироваться к конкретным условиям применения те или иные функциональные программы (для офисных технологий).

Системы программирования — это совокупность программ для разработки, отладки и внедрения новых программных продуктов. Системы программирования обычно содержат:

- трансляторы;
- среду разработки программ;
- библиотеки справочных программ (функций, процедур);
- отладчики;
- редакторы связей и др.

1.4 Классификация операционных систем

Операционные системы (ОС) относятся к специальному виду системного программного обеспечения, без которого невозможно осмысленное взаимодействие пользователя и компьютера. Операционная система нужна для того, чтобы пользователь мог управлять компьютером, она определяет общие правила запуска программ, управления данными и доступа к ресурсам компьютера.

Операционная система, как правило, загружается при включении электрического питания компьютера.

Операционная система выполняет следующие функции:

- управление работой каждого блока компьютера и их взаимодействием;
- управление выполнением программ;
- организацию хранения информации во внешней памяти;
- взаимодействие пользователя с компьютером, т.е. поддержку интерфейса пользователя.

Операционная система является программной надстройкой над архитектурой компьютера, которая обеспечивает удобный пользовательский интерфейс, берет на себя функции автоматического управления рядом его подсистем и предоставляет готовые процедуры управления внутренними и внешними ресурсами. Это значит, что операционная система является некоей автоматизированной системой управления работой и ресурсами компьютера, повышающей удобство и эффективность его использования.

Операционные системы классифицируют по различным признакам:

По особенностям алгоритмов управления ресурсами:

- *локальные,* управляющие ресурсами отдельного компьютера;
- *сетевые,* участвующие в управлении ресурсами сети.

По количеству одновременно выполняемых задач:

- однозадачные, выполняющие функцию предоставления пользователю виртуальной вычислительной машины, обеспечивая его простым и удобным интерфейсом взаимодействия с компьютером, средствами управления периферийными устройствами и файлами;
- многозадачные, кроме вышеперечисленных функций, управляют разделением совместно используемых ресурсов, таких как процессор, оперативная память, файлы и внешние устройства.

Многозадачность также означает, что пользователь может работать с несколькими программами одновременно. Например, слушать звуковой файл, рисовать что-нибудь в графическом редакторе и печатать нужный документ.

По количеству одновременно работающих пользователей:

- однопользовательские;
- многопользовательские.

Основным отличием многопользовательских систем от однопользовательских является наличие средств защиты информации каждого пользователя от несанкционированного доступа других пользователей.

Программы, созданные в среде одной операционной системы, не функционируют в среде другой операционной системы, если в ней не обеспечена возможность конвертации (преобразования) программ.

1.5 Открытое и закрытое программное обеспечение

По способу распространения различают открытое и закрытое программное обеспечение.

Закрытое программное обеспечение — это такая модель программного обеспечения, при которой автор (или иной правообладатель) удерживает за собой определенные права (в частности, запрещение повторного распространения, изменения программ или очень жесткого их ограничения). Для большинства программ исходный код недоступен, что делает невозможной или по крайней мере нетривиальной задачу модификации программ.

Открытое программное обеспечение — это такая модель программного обеспечения, при которой дается гарантия свободно распространять копии программы вместе с исходным кодом, изменять программу или использовать ее части в новых открытых разработках. Фактически это дает возможность без оплаты законно использовать, модифицировать для собственных нужд полнофункциональные программы: от операционных систем и серверных решений — до офисных приложений, графических редакторов и т.д.

Открытое ПО не значит бесплатное. «Без оплаты» в данном случае означает: без оплаты обязательной лицензии на использование и без защиты от копирования.

Следует отличать *открытое ПО (Open Source Software)* от *бесплатного ПО (Freeware).* Бесплатное ПО, как правило, это программы, которые можно свободно распространять, однако их исходный код недоступен.

Термин *«свободное ПО» (Free Software)* является практическим аналогом термина «открытое ПО».

Открытые программы получили широкое распространение в сфере профессионального использования — это, прежде всего, инструментальные средства (программы, используемые в самом процессе создания ПО, включая написание, отладку, модификацию программ), затем — серверные программы и — как частный случай последних — сетевые (Интернет) сервисы, в которых операционные системы (Linux, Free BSD и др.) и прикладные программы (Web-сервер Apache, почтовая программа Sendmail и др.) сегодня лидируют с большим отрывом.

Положительные моменты использования открытого ПО, кроме имеющих явно экономический характер:

- возможность модификации программ под конкретные задачи;

- высокая надежность и защита ПО, в частности, от несанкционированного доступа;
- возможность поддержки ПО любой группой квалифицированных программистов, а не только малодоступной компанией, имеющей монопольные права.

Недостатки закрытого ПО:

- дороговизна,
- неэффективность,
- недостаточная безопасность и надежность решений на базе закрытого ПО.

В развитых экономиках выход из ситуации, создавшейся в сфере ПО, найден. Ряд государств (Германия, Норвегия, Япония, Китай, Индия, Бразилия и др.) переходит в сфере государственного управления на открытое ПО. Эта же тенденция наблюдается и в коммерческих компаниях. Примером такого перехода может служить проект Академии наук Китая и компании New Margin Venture Caputal, в рамках которого была разработана операционная система Red Flag Linux (RFL) (на базе операционной системы Linux). Система Red Flag Linux активно поддерживается государством и распространяется во всех государственных учреждениях с целью обеспечить программную независимость от Запада и обезопасить себя от обвинений в пиратстве и нарушении лицензий. Сегодня в Китае операционная система RFL широко применяется в национальном и местном самоуправлении, образовании, энергетике, связи, транспорте, управлении финансами, СМИ.

Примером использования открытого ПО в бизнесе является компания Siemens, которая перевела часть своих компьютеров под операционную систему Linux. Положительный опыт рассматривается этой компанией как перспектива отказа от использования платформы Windows.

Ведущие японские компании также используют в своей деятельности открытое ПО.

1.6 Информатизация общества

Во второй половине XX века возникла необходимость решения многочисленных проблем доступа к информации, вызванных возникновением так называемого «информационного кризиса», обусловленного следующими причинами:

- возникновением противоречий между ограниченными возможностями человека по восприятию и переработке информации и существующими мощными потоками и массивами хранящейся информации;
- существованием огромного количества избыточной информации, которая затрудняет восприятие полезной для потребителя информации;

- возникновением разнообразных экономических, социальных и других барьеров, препятствующих распространению информации.

Таким образом создалась парадоксальная ситуация: из-за ограниченности своих возможностей люди не могут воспользоваться в полном объеме огромным информационным потенциалом, накопленным в мире. Информационный кризис поставил общество перед необходимостью поиска путей выхода из создавшегося положения. Широкое внедрение электронных вычислительных машин, современных средств переработки и передачи информации в различные сферы деятельности положило начало новому эволюционному процессу в развитии человеческого общества, получившего название *«информатизация»*.

Информатизация — это процесс, обеспечивающий удовлетворение нужд и интересов граждан, общества и государства в информационных ресурсах и возможность доступа к ним посредством современных информационных технологий и развитой информационной инфраструктуры. Информатизация, основанная на базе внедрения компьютерных и телекоммуникационных технологий, представляет собой инструмент эффективного использования информации и знаний в различных областях человеческой деятельности.

В современном обществе уровень информатизации характеризует уровень социально-экономического развития государства.

1.6.1 Общие сведения

История развития информатизации началась в США с 1960-х гг, затем с 70-х гг. — в Японии и с конца 70-х — в Западной Европе.

Информатизация является одной из ведущих тенденций социально-технического прогресса.

Этот термин все настойчивее вытесняет широко используемый до недавнего времени термин «компьютеризация общества». Между этими терминами имеется существенное различие.

При *компьютеризации общества* основное внимание уделяется развитию и внедрению технической базы компьютеров, обеспечивающих оперативное получение результатов переработки информации и ее накопление.

При *информатизации общества* основное внимание уделяется комплексу мер, направленных на обеспечение полного использования достоверного, исчерпывающего и своевременного знания во всех видах человеческой деятельности.

Таким образом, «информатизация общества» является более широким понятием, чем «компьютеризация общества», и направлена на скорейшее овладение информацией для удовлетворения своих потребностей.

Компьютеры являются базовой технической составляющей процесса информатизации общества.

Информатизация на базе внедрения компьютерных и телекоммуникационных технологий является реакцией общества на

потребность в существенном увеличении производительности труда в стремительно увеличивающемся информационном секторе общественного производства.

Объектами информатизации являются:

- общество;
- государство;
- экономика;
- наука и образование;
- производство;
- личность.

Основным объектом информатизации является *общество*. Процесс информатизации воздействует на него либо непосредственно, либо через другие объекты информатизации, в первую очередь через государство. В связке «общество — государство» общество является ведущим.

Особенности *общества* как объекта информатизации:

- информационные системы (в том числе СМИ) общественного назначения всегда имеют открытый характер, что определяет необходимость применения технологий информатизации, обеспечивающих открытый информационный объем;
- информатизация объектов социальной сферы практически всегда ориентирована на общение с людьми, что требует применения во всех технологиях информатизации простого интерфейса «человек — ЭВМ»;

Особенности *личности* как объекта информатизации определяются тем положением, которое она занимает в связке «общество — личность». Значение личности по мере повышения общей цивилизованности общества и уровня жизни постоянно возрастает, поэтому услуги, которые оказываются обществу, частично индивидуализируются, причем степень индивидуализации все время растет. Важное место среди таких услуг начинают занимать те, которые обеспечивают все более возрастающие потребности личности. К таким услугам можно отнести:

- свободное получение, распространение и использование общественно значимой информации;
- свободное получение информации, необходимой для образования, систематической профессиональной подготовки и переподготовки;
- защиту сознания личности от нежелательных воздействий, в том числе и через СМИ.

1.6.2 Классификация технологий информатизации

Технологии информатизации — это совокупность технических, программных и организационно-экономических средств, объединенных структурно и функционально для решения той или иной задачи информатизации, т.е. для повышения эффективности функционирования данного типа объектов или отдельного объекта информатизации.

Все технологии информатизации разделяются на три основных класса: базовые, прикладные, обеспечивающие.

Базовые технологии информатизации создают возможности решения отдельных компонентов той или иной функциональной задачи на объекте информатизации, а также служат основой формирования прикладных технологий информатизации.

К *базовым технологиям информатизации* относятся:

- технологии программирования, включая языки программирования;
- телекоммуникационные технологии;
- базы данных, в том числе специализированные;
- технологии обработки изображений (распознавание образов, компьютерная графика, визуализация информатизации);
- технологии человеко-машинного интерфейса, в том числе на естественном языке;
- технологии распознавания речи;
- экспертные системы;
- методы и алгоритмы параллельных вычислений;
- моделирование процессов, в том числе в реальном масштабе времени;
- моделирование ситуаций, в том числе в реальном масштабе времени;
- программные и технические технологии информационной безопасности;
- технологии хранения и обработки сверхбольших массивов информации;
- технологии криптографии;
- сетевые технологии предоставления информационных услуг;
- технологии цифро-аналогового преобразования.

Прикладные технологии информатизации формируются на основе базовых и ориентированы на полную информатизацию объекта, т.е. комплексное решение функциональной задачи. Эти технологии составляют основную массу продаваемых на рынке продуктов:

- системы автоматизированного проектирования (САПР);
- автоматизированные системы управления производством (АСУП);
- системы поддержки принятия решений (СППР);
- географические информационные системы (ГИС);

- системы обеспечения банковской деятельности;
- системы автоматизации офиса;
- мультимедиа технологии, технологии виртуальной реальности;
- системы управления процессами в реальном масштабе времени (РМВ);
- системы контроля качества;
- системы управления запасами;
- системы управления трафиком (связь, наземный, воздушный, водный виды транспорта);
- издательские системы;
- медицинские информационные системы;
- бухгалтерские системы;
- системы автоматизации торговли;
- системы машинного перевода;
- технологии формирования и распространения информации для средств массовой информации.

Обеспечивающие технологии информатизации создают возможность реализации технологий двух первых классов. На рынке имеются, как правило, лишь их отдельные элементы:

- современная микроэлектронная база средств вычислительной техники (ВТ), информатики, телекоммуникаций;
- перспективные вычислительные средства (оптические, транспьютеры, нейрокомпьютеры, компьютеры нетрадиционной архитектуры и др.);
- технологии интеграции средств информатизации и отдельных технологий в функционально ориентированные среды (открытые системы);

- технологии организации вычислительного процесса.

1.7 Информационное общество

Информатизация общества сегодня признается большинством специалистов глобальной мировой тенденцией, затрагивающей в той или иной степени все страны и характеризующей процесс перехода от индустриального типа общества к информационному, т.е. такому, где большинство трудоспособного населения занято в сфере создания, распределения информации и обмена ею, а каждый член общества может получать необходимый информационный продукт или услугу в любом месте и в любое время. Основой процесса информатизации является широкое внедрение современных информационных технологий во все сферы жизни общества.

1.7.1 Общие сведения

Понятие *«информационное общество»* появилось во второй половине XX века. Впервые этот термин был использован в Японии в 1966 г. в докладе группы по научным, техническим и экономическим исследованиям, в котором утверждалось, что информационное общество представляет собой общество, в котором имеется в изобилии высокая по качеству информация, а также есть все необходимые средства ее распределения.

Конец XX и начало XXI века ознаменовались внедрением во все сферы жизни информационно-коммуникационных технологий (ИКТ) и созданием глобальной информационной компьютерной сети — Интернета. ИКТ и Интернет создали технологическую основу для перехода развитых стран от индустриального общества к информационному.

Основными отличительными признаками информационного общества являются:

- информационная экономика;
- высокий уровень информационных потребностей всех членов общества и фактическое их удовлетворение для основной массы населения;
- высокая информационная культура;
- свободный доступ каждого члена общества к информации, ограниченный только информационной безопасностью личности, общественных групп и всего общества.

Информационному обществу присущи:

- единое информационное пространство;
- доминирование в экономике новых технологических укладов, базирующихся на массовом использовании сетевых информационных технологий, перспективных средств вычислительной техники и телекоммуникаций;
- ведущая роль информационных ресурсов в обеспечении устойчивого поступательного развития общества;
- возрастание роли инфраструктуры (телекоммуникационной, транспортной, организационной) в системе общественного производства и усиление тенденций к совместному функционированию в экономике информационных и денежных потоков;
- фактическое удовлетворение потребностей общества в информационных продуктах и услугах;
- высокий уровень образования, обусловленный расширением возможностей систем информационного обмена на международном, национальном и региональном уровнях и соответственно — повышенная роль квалификации, профессионализма и способностей к творчеству как важнейших характеристик труда;

- высокая значимость проблем обеспечения информационной безопасности личности, общества и государства, наличие эффективной системы обеспечения прав граждан и социальных институтов на свободное получение, распространение и использование информации.

Общепринятого определения информационного общества пока нет. Множество существующих определений этого понятия обусловлено сложностью, специфичностью и многообразием подходов к его толкованию. Суть большинства определений в основном сводится к тому, что *информационное общество (ИО) — это такое общество, в котором производство и потребление информации является важнейшим видом деятельности, а информация признается наиболее значимым ресурсом; новые информационные и телекоммуникационные технологии и техника становятся базовыми технологиями и техникой, а информационная среда наряду с социальной и экологической — новой средой обитания человека.*

1.7.2 Национальные стратегии перехода к информационному обществу

Современный технологический прогресс, повышение роли, качества и объемов информации как автономного ресурса (наряду с материей и энергией) человеческого бытия, с одной стороны, и формирование теории информационного общества, с другой — создают условия для создания, обоснования и реализации национальных стратегий и тактик перехода к информационному обществу.

Хотя глобальное информационное общество формируется локально и в разных странах этот процесс идет с различной интенсивностью и особенностями, движение к информационному обществу — это общая тенденция как для развитых, так и для развивающихся стран.

Каждая страна разрабатывает свою концепцию вхождения в информационное общество, исходя из своих собственных конкретных условий (развитости телекоммуникационной инфраструктуры, информационной индустрии, законодательной базы и т.д.). В настоящее время национальные стратегии перехода к информационному обществу разработаны и реализуются в США, Великобритании, Канаде, Финляндии, Франции, Японии, Италии, ФРГ, Дании и ряде других развитых стран, а также на уровне международных союзов — ЕС (Электронная Европа), АСЕАН (Электронная Азия).

Разработаны и постоянно совершенствуются стратегии перехода к информационному обществу в Беларуси, России, Украине, Молдове, Армении и других странах СНГ. Соответствующие государственные программы информатизации этих стран носят названия «Электронная Беларусь», «Электронная Россия», «Электронная Украина», «Электронная Молдова» и т.д.

1.8 Технологии доступа к электронным информационным ресурсам

Одной из основных особенностей развития общества в конце XX - начале XXI века является резкое возрастание социальной значимости информации. Увеличились информационные потребности людей, а информация превратилась в массовый продукт. Возник информационный рынок, платным товаром на котором выступает информация.

Другая особенность современного этапа развития общества — бурный рост информационных потоков. Для своевременного принятия правильных решений сегодня необходим свободный доступ к разнообразной, в том числе и научной, информации.

Информационные преимущества становятся важной социальной силой: обладающие информацией — обладают властью. В условиях непрерывно ускоряющегося динамизма общественных изменений резко возрастает потребность в оперативной информации о происходящих изменениях для обеспечения своевременной реакции на них. Происходит сдвиг совокупного спроса в сторону информационных потребностей. Возрастают вес информационной экономики и ее доля в суммарном рабочем времени.

1.8.1 Электронное правительство

Стремительное развитие информационных и коммуникационных технологий (ИКТ) привело во всем мире к большим переменам не только в промышленности, но и в секторе услуг. Проникая в различные сферы жизни, ИКТ необратимо влияют на все общество, включая организацию деятельности центральных и местных правительств и всех ветвей государственной власти. Работы в этом направлении ведутся на разных уровнях, начиная с национальных проектов, принятых в развитых странах, и заканчивая программой построения глобального информационного общества стран «Большой восьмерки». (Здесь и далее под правительством понимается его центральный аппарат и все подразделения: министерства, комитеты и др.)

Характерной особенностью взаимодействия правительств большинства стран со своими гражданами является стремительно возрастающее (по некоторым оценкам в полтора раза ежегодно) количество лиц, организаций и институтов, имеющих представительство в глобальной сети Интернет и активно его использующих.

В связи с этим в большинстве стран мира организованы так называемые *электронные правительства,* представляющие собой использование интернет-технологий для обеспечения взаимодействия власти с населением и организациями гражданского общества.

Электронное правительство (e-government) — термин, который стал популярен в последние 20 лет прошлого века. Сегодня существует множество определений этого термина, каждое из которых акцентирует внимание на отдельных функциях электронного правительства. Обобщенно электронное

правительство можно охарактеризовать как автоматизацию процесса предоставления государственных услуг.

Под термином *«электронное правительство»* понимают также непрерывную оптимизацию процесса предоставления услуг, политического участия граждан и управления путем изменения внутренних и внешних отношений при помощи технических средств, Интернета и современных средств массовой информации.

История создания электронных правительств идет параллельно с развитием интернет-технологий. Как полагают эксперты, введение информационно-коммуникационных технологий в государственное управление позволит ускорить развитие экономики, снизить затраты на бюрократические процедуры, повысить эффективность работы и производительность труда государственных ведомств, расширить возможности населения в формировании гражданского общества за счет улучшения доступа к различной информации, создания более прозрачной работы государственных служб, ослабления бюрократических барьеров.

Таким образом, создание электронного правительства предполагает достижение следующих основных целей:

- оптимизации предоставления правительственных услуг населению и бизнесу;
- повышения степени участия всех избирателей в процессах руководства и управления страной;
- поддержки и расширения возможностей самообслуживания граждан;
- роста технологической осведомленности и квалификации граждан; снижения воздействия фактора географического местоположения.

Электронное правительство состоит из двух частей: первая — взаимодействие власти и общества и вторая — внутреннее взаимодействие разных уровней (исполнительной, законодательной, судебной).

В настоящее время во многих странах мира, и в первую очередь в экономически развитых, предпринят целый ряд мер, направленных на формирование и развитие электронных правительств. В 2005 г. из почти 200 государств-членов ООН национальные правительства 175 стран, так или иначе, использовали интернет-технологии для предоставления различной информации и услуг. Большинство государств в настоящее время имеют правительственные сайты.

Электронные правительства различных стран способствуют непрерывной оптимизации процесса предоставления услуг, политического участия граждан и управления, а также реализации принципа доступности правительства каждому гражданину или организации в любом месте, в любое время.

1.8.2 Развитие правовой информатизации. Доступ к правовой информации

В сфере распространения правовой информации в последнее десятилетие произошли значительные изменения. Возникла и продолжает увеличиваться массовая потребность в правовой информации как у специалистов, так и у широких слоев населения. Откликом на эту потребность стали предложенные эффективные и качественные методы и средства работы с правовой информацией и соответствующие информационные услуги. Причем особое, опережающее развитие получили компьютерные технологии распространения правовой информации, которая в условиях перехода к информационному обществу, где главной целью является удовлетворение социальных, экономических и индивидуальных потребностей граждан в доступе к разнообразным информационным ресурсам, приобретает исключительно важное значение.

Основными направлениями в этих процессах являются широкое использование компьютерных и интернет-технологий и обеспечение максимально широкого доступа заинтересованных лиц и организаций к правовой информации.

С помощью компьютера пользователь может получить правовую информацию, обратившись к справочным правовым системам, которые предоставляют полную систематизированную и оперативно обновляемую информацию по законодательству, а также программные средства поиска, анализа и обработки этой информации. Справочные правовые системы являются эффективным инструментом для работы с большим объемом документов. В сущности, справочные правовые системы представляют собой объемные удобные справочники, где пользователь может найти информацию, необходимую для анализа конкретной ситуации.

Пользование справочными правовыми системами зачастую осуществляется на платной основе и является малодоступным для рядовых граждан. Рядовой гражданин данными системами может воспользоваться по месту работы (если соответствующая база данных установлена на предприятии или в организации) или в публичных центрах правовой информации. Кроме того, следует отметить, что компьютерные правовые системы рассчитаны, прежде всего на лиц, имеющих юридическое образование.

Развивающимся источником получения гражданами правовой информации является сеть Интернет. В Интернете сосредоточена разнообразная по содержанию и качеству правовая информация, включая электронные библиотеки правовой информации. Пользователи могут воспользоваться информационными сайтами министерств и ведомств и поисковыми системами правовой информации; получить юридическую консультацию; поучаствовать в работе дискуссионных клубов по правовой тематике; подписаться и прочитать юридические интернет-журналы, прослушать, прочитать и посмотреть передачи по правовой тематике, задать вопросы и высказать свое мнение на сайтах различных организаций и интернет-форумах.

Большая часть интернет-ресурсов рассчитана на специалистов, однако некоторые ресурсы по правовой тематике доступны и рядовым гражданам. Отсутствие материальных средств, необходимых для получения доступа как к Интернету (оплата услуг провайдера и др.), так и к определенной информации ограничивает доступ к данному источнику правовой информации.

Большинство экспертов сходятся во мнении, что Интернет в настоящее время нельзя рассматривать как универсальный и доступный источник правовой информации. Это связано как с объективными (недостаточно качественный и количественный уровень правовой информации, размещенной в Интернете, неразвитость информационной инфраструктуры), так и с субъективными причинами (отсутствие у потенциальных пользователей необходимых навыков работы в Сети).

Из существующих каналов получения правовой информации самыми надежными источниками являются справочные правовые системы и публичные центры правовой информации.

Считается, что система распространения правовой информации в стране решает свои задачи, если она обеспечивает каждому гражданину принципиальную возможность получать правовую информацию в электронном виде или базовый уровень доступа к ней.

Принципиальная возможность получать правовую информацию в электронном виде, или базовый уровень доступа к правовой информации, может считаться реализованной, если выполняются следующие условия:

- любой гражданин, имеющий доступ в сеть Интернет и способный оплатить услуги этой сети, должен иметь возможность получать полные тексты официальных правовых документов в базовом формате из правовой базы данных, выставленной на условиях бесплатного доступа;
- гражданин, не имеющий доступа в сеть Интернет, должен иметь возможность получить бесплатно либо по цене, покрывающей стоимость копирования, тексты необходимых ему официальных правовых документов в базовом формате из правовой базы данных.

Широкое внедрение информационных технологий во все правовые сферы общества как основы правовой информатизации требует тесного сотрудничества и взаимодействия многих стран на международном уровне.

В настоящее время принят ряд международных правовых актов в сфере информатизации, в которых особое значение уделяется реализации прав граждан и организаций на доступ к информации.

Большое влияние на развитие процессов правовой информатизации оказывает европейский опыт правовой информатизации и рекомендации Комитета министров Совета Европы и Организации Объединенных Наций по вопросам образования, науки и культуры (ЮНЕСКО), принятые в последние годы.

В настоящее время в Республике Беларусь, как и в России, уже полностью сформировалась система распространения правовой информации в

электронном виде, которая по многим показателям эффективности не уступает аналогам большинства ведущих стран Европы.

Одним из мероприятий в рамках программ правовой информатизации Беларуси и России было создание публичных центров правовой информации (ПЦПИ), как правило, на базе системы общедоступных библиотек.

Основу кадрового состава ПЦПИ составляют библиографы, и лишь в некоторых Центрах в штат включены профессиональные юристы и профессиональные программисты, что, естественно, сказывается на качестве и объеме получаемой гражданами правовой информации.

ПЦПИ оказывают различные информационные услуги, в том числе и бесплатную юридическую помощь.

Центры правовой информации — составная часть инфраструктуры общественного доступа к открытой информации электронного правительства.

Сегодня правовая информированность граждан является одним из приоритетных факторов динамичного развития любого цивилизованного государства. Информатизация правовой сферы деятельности позволяет привлечь к процессам принятия решений все большее количество необходимых информационных ресурсов и, вследствие этого, сделать принятие решений более объективным и соответствующим намеченным целям.

1.8.3 Доступ к электронным информационным ресурсам в Республике Беларусь

Построение информационного общества в Беларуси рассматривается в качестве одной из приоритетных задач национального развития.

В республике в соответствии с Планом действий, разработанным в ходе Саммита по информационному обществу, состоявшегося в декабре 2003 г. в Женеве, реализуется стратегия электронного развития.

Беларусь неуклонно наращивает свой технологический потенциал. Разработка сложного специализированного программного обеспечения, его внедрение и системная интеграция, а также обучение с использованием ИКТ остаются приоритетными направлениями информационного развития.

Информационную инфраструктуру Республики Беларусь формируют телекоммуникационные и компьютерные сети и распределенные базы данных и знаний.

Общенациональная стратегия перехода к информационному обществу нашла свое отражение в *Государственной программе информатизации Республики Беларусь на 2003-2005 гг. и на перспективу до 2010 г. «Электронная Беларусь»*.

В период 2006-2010 гг. должны быть завершены работы по созданию общегосударственной автоматизированной информационной системы, сформирована единая информационная и телекоммуникационная инфраструктура, обеспечено внедрение системы электронной торговли для государственных нужд на республиканском уровне, стандартизованного электронного документооборота и систем обеспечения национальной безопасности. Кроме того, одним из важных результатов программы станет

увеличение численности пользователей сети Интернет и объемов получаемых с ее помощью услуг.

Таким образом, программа «Электронная Беларусь» определяет основные направления электронной стратегии развития информационного общества, основанного на широком распространении и обмене информацией, а также на подлинном участии всех заинтересованных сторон (правительства, частного сектора и гражданского общества) в процессах включения страны в мировое информационное пространство.

Беларусь уже прошла первоначальную стадию процесса информатизации. Практически во всех отраслях экономики, оборонной, социальной и культурной сферах созданы информационные системы, формируются и используются государственные информационные ресурсы.

В республике ведется активная работа по внедрению технологий электронного правительства. Разработана Концепция создания Единого государственного информационного ресурса. Министерством связи и информатизации координируется деятельность по созданию автоматизированных систем обеспечения функционирования органов государственного управления в рамках Государственной программы информатизации «Электронная Беларусь».

Электронное правительство Республики Беларусь составляют web-сайты Президента Республики Беларусь, правительства *(http://president.gov.by/, http://www.government.by/),* министерств и других органов власти.

Качество официальных интернет-ресурсов оценивается экспертами ООН по рейтингу электронного правительства. Беларусь в этом рейтинге сделала стремительное восхождение с 81-го места в 2003-м г. на 51-е место в 2005-м г. (среди 191 страны), а по субиндексу электронного участия граждан она оказалась 22-й в мире.

При этом динамика индекса, учитывающего качество правительственных web-сайтов, телекоммуникационной инфраструктуры и человеческого капитала, одна из лучших.

Если в 2002 г. индекс готовности Беларуси к электронному правительству в точности равнялся среднемировому, то в 2005 г. он уже был выше среднемирового. Из наших соседей наиболее высокие места у Украины (48-е) и России (50-е).

Использование возможностей электронного правительства рассматривается органами управления Беларуси как стратегически важное направление повышения эффективности государственной власти.

Развитая инфраструктура сетей электросвязи в Беларуси позволяет обеспечить и совершенствовать подключение к Интернету школ, вузов, учреждений здравоохранения, библиотек, почтовых отделений и других доступных для населения учреждений.

С 1998 г. в стране реализуется республиканская программа «Информатизация системы образования», которая определяет основные направления работы по оснащению учебных заведений средствами информационных технологий, совершенствованию управления в сфере

образования, повышению уровня подготовки учащихся в области информационных технологий.

Важной стороной информатизации образования в Республике Беларусь, как и в других странах мира, является формирование белорусского образовательного сегмента сети Интернет.

Сегодня практически все высшие учебные заведения Беларуси, как и большинство вузов мира, представлены в Интернете. Информация, представленная на образовательном сегменте белорусского Интернета, адресована всем заинтересованным лицам и организациям: выпускникам вузов, аспирантам и соискателям ученых степеней, их научным руководителям, всем специалистам и организациям, занимающимся подготовкой научных кадров высшей квалификации.

Таким образом, современная общемировая тенденция резкого возрастания интереса населения к образованию, и в первую очередь к высшему, находит отражение в системе образовательных сайтов белорусского сегмента Интернета.

Свое представительство в Интернете имеют также Министерство образования Республики Беларусь *(http://www.minedu.unibel.by)* и другие ведомства и учреждения, занимающиеся наукой и образованием, наиболее значимыми среди которых являются Высшая аттестационная комиссия Республики Беларусь *(http://www.vak.org.by),* Государственный комитет по науке и технологиям Республики Беларусь *(http://www.gknt.org.by/),* Национальная академия наук Беларуси *(http://www.ac.by),* отраслевые научно-исследовательские учреждения.

Информационно-правовая политика белорусского государства активно направлена на создание условий для эффективного и качественного обеспечения правовой информацией на самых разных уровнях: от государственных органов до каждого гражданина. В стране формируется единое информационно-правовое пространство путем создания государственной системы правовой информации (ГСПИ).

Ее основное назначение — оперативное обеспечение полной и достоверной правовой информацией всех государственных органов, юридических и физических лиц; содействие оптимизации правотворческой, правоохранительной и правоприменительной деятельности, росту правосознания и правовой культуры.

Основные составляющие ГСПИ:

- Национальный центр правовой информации Республики Беларусь (НЦПИ) — центральное государственное научно-практическое учреждение в области правовой информатизации *(http://www.pravo.by),* основной источник правовой информации Беларуси в сети Интернет;
- автоматизированная система формирования государственного информационного нормативно-правового ресурса;
- комплексная система распространения правовой информации, обеспечивающая распространение информации на электронных и

бумажных носителях с использованием различных организационных и информационно-технологических форм ее распространения;

- система межгосударственного обмена правовой информацией.

В совокупности все составляющие ГСПИ образуют уникальную государственную межведомственную автоматизированную систему управления правовым ресурсом, которая позволила сформировать в стране единый официальный, полный, актуализируемый, общедоступный государственный правовой информационный ресурс.

Активно набирает темпы процесс формирования публичных центров правовой информации (ПЦПИ) на базе публичных библиотек. Информационные ресурсы центров позволяют удовлетворять различные запросы обращающихся. Каждый ПЦПИ комплектуется электронной копией эталонного банка данных правовой информации, Национальным реестром правовых актов Республики Беларусь и другими официальными изданиями НЦПИ. Большинство ПЦПИ установили дополнительные банки данных: «Судебная практика» и «Законодательство Российской Федерации». К услугам пользователей — справочно-библиографический аппарат библиотеки, в состав которого входит картотека законов Республики Беларусь, фактографическая картотека, фонд исполненных заявок. В последнее время в целях обеспечения доступа граждан к актам местных органов управления и самоуправления наметилась тенденция представления городскими (районными) исполкомами в ПЦПИ копий текстов принятых ими нормативных правовых актов на бумажных носителях.

В ПЦПИ граждане могут бесплатно или за минимальную цену получить интересующую их правовую информацию (тексты нормативных актов). Как правило, поиск нормативных актов в ПЦПИ осуществляет специально подготовленный сотрудник данной библиотеки, что обеспечивает квалифицированное обслуживание посетителей. В Белорусском университете культуры, на библиотечном факультете, готовятся специалисты для работы в ПЦПИ.

Помимо предоставления текстов нормативных актов в ряде ПЦПИ для разъяснения законодательства привлекаются адвокаты, организуются семинары по различным областям общественных отношений с участием соответствующих специалистов.

В 2004 г. в Беларуси на базе публичных библиотек функционировало 135 пунктов правовой информации, которые обеспечивают свободный доступ граждан к официальной правовой информации. Они действуют во всех областных и практически в каждой из 129 центральных библиотек городских и районных централизованных библиотечных систем. С 2004 г. пункты правовой информации начали действовать и в филиалах библиотечных систем, т. е. в районных и сельских библиотеках

В 2007 г. в республике действовало уже более 200 ПЦПИ на базе всех библиотек, входящих в централизованную библиотечную систему. В настоящее время развернута работа по организации ПЦПИ во всех сельских библиотеках.

Кроме перечисленных возможностей доступа к информации, в Беларуси развиваются телемедицинские системы, электронные системы поддержки экспорта. Положено начало внедрению систем биометрического паспортно-визового контроля.

Одним из основных условий перехода к информационному обществу является обеспечение свободного и равноправного доступа к информации, главным образом, к той, что находится в государственных информационных ресурсах публичных организаций. Организация в Республике Беларусь доступа к этой информации демонстрирует признание его одной из важнейших ценностей современной цивилизации.

1.9 Электронные книги

Современная информационная революция привела к возникновению электронной информационной среды: 93 % вновь создаваемой информации является уже цифровой. Множество специалистов занято сегодня переводом различных видов информации в цифровую форму.

Электронные книги — это книги, представленные в цифровых форматах.

Книги сегодня все чаще приобретают электронную форму. Возникли даже издательства, которые занимаются исключительно электронными книгами (например, «Электронная книга»).

Цифровые книги являются важным элементом компьютерной культуры. Они не только выполняют культурные функции обучения и развлечения, значительно расширяя доступ к культурному наследию человечества, но и позволяют создавать пользователям свои новые произведения, а также участвовать в коллективном творчестве. Кроме текста, цифровые книги могут содержать иллюстрации, звуковое сопровождение и другие элементы, обеспечивающие читателю наиболее эффективное восприятие представленного материала.

Чаше всего сегодня электронные книги подразделяют на *аудиокниги* и *визуальные,* т. е. электронные книги, которые слушают и электронные книги, которые читают.

Аудиокнигой принято называть фонограмму, содержащую прочитанное диктором, профессиональными актерами или авторами литературное произведение, текст которого слово в слово соответствует бумажному изданию.

В большинстве случаев аудиокниги предлагаются в виде аудиофайлов сжатых форматов (например, MP3). Для прослушивания книг в звуковом формате чаще всего используют компьютеры и портативные аудиоплееры.

В настоящее время в США и западноевропейских станах доля издаваемых в звуковом формате литературных произведений уже достигла примерно 20% количества книг, выпускаемых в традиционном печатном виде.

В странах СНГ, по данным социологических опросов, популярность аудиокниг у различных категорий населения также постоянно возрастает.

В глобальной сети Интернет .представлены различные виды цифровых книг. Это аудиокниги и визуальные (линейные и гипертекстовые) электронные книги.

Чтение *визуальных электронных книг* осуществляется в среде текстового редактора или интернет-браузера.

Электронные визуальные книги (линейные и гипертекстовые) могут быть как художественными, так и нехудожественными, к которым относят справочники, телефонные книги, «инструкции пользователя» и различные энциклопедии.

Электронные визуальные книги бывают *сетевыми* (расположенными в сети) и *изолированными* (расположенными на компакт-дисках). Классическим примером *изолированной* электронной книги является роман «После полудня» Майкла Джойса, выполненный по гипертекстовой технологии, который продается только в виде компакт-диска. В то же время имеются произведения, изданные как в печатном, так и в электронном виде.

К *сетевым* электронным книгам относят гиперроманы (романы, выполненные по технологии гипертекста), «живущие» только в Сети. Эти произведения не только могут быть доступны одновременно многим пользователям, но зачастую объединяются в целые сайты подобных себе произведений.

Сетевые электронные книги в свою очередь делятся на книги, обеспечивающие:

- только чтение;
- чтение с комментариями;
- чтение/письмо.

Чтение с комментарием - наиболее часто встречающийся вид размещения произведений в Интернете. Читателю/зрителю (ведь речь идет не только о текстах, но и о картинах, фотографиях, звуковых фрагментах и т. д.) предоставляется возможность в специальной «гостевой книге» выразить свое восхищение, несогласие и прочие чувства.

Популярные сегодня «гостевые книги» некоторые специалисты склонны рассматривать в качестве нового вида искусства *сетературы,* т. е. сетевой литературы.

Чтение/письмо — это постоянно развивающиеся сетевые проекты. Они делятся на:

- проекты одного автора;
- проекты с возможностью коллективного творчества.

Проект одного автора реализуется в том случае, когда доступ к тексту, расположенному на сайте, защищен и дозволен только самому автору или с его разрешения. К этой же разновидности относятся все произведения, выходящие на компакт-дисках. В Сети такова, например, «Набережная Житинского», сайт, на котором петербургский писатель А.Н. Житинский размещает свои гипертекстовые произведения.

Доступ к «многоавторным» проектам доступен всем желающим. Одним из таких проектов на русском языке является «Сад расходящихся хокку» «Журнала.Ру».

Анализ существующих сегодня электронных книг показывает, что чаще всего мы имеем дело лишь с электронной версией печатного издания, которое не подготавливалось с учетом всех особенностей и возможностей электронного издания и обычно не имеет гиперссылок, аудиовизуального материала, не отличается интерактивностью.

В настоящее время в Интернете существует огромное количество изданий, которые хотя и именуются электронными, но реально представляют собой всего лишь тексты, набранные в каком-либо текстовом редакторе. Атрибуты издания в них полностью отсутствуют, зачастую нет даже титульного листа. В таких текстах не редки всевозможные ошибки — грамматические, синтаксические, множество опечаток и т.д.

Бесспорно, развитие электронной книги и электронных изданий находится на начальном этапе. Очевидны преимущества, предоставляемые нам ее электронной формой (в ее полном объеме): наличие поисковых систем, возможность персональных изданий, компактность, доступность и оперативность, возможность построения собственной канвы повествования, экономичность (электронные книги дешевле печатных), легкость работы с содержанием.

Однако у электронных книг имеется и целый ряд недостатков. К ним относят проблемы защиты авторского права, отсутствие единых стандартов на форматы электронных книг, неудобство чтения с экрана, быстрое старение компьютерных технологий, все еще дорогую инфраструктуру, зависимость от электричества и Интернета, невозможность классической литературной критики гипертекстовых произведений, поскольку в гипертексте отсутствует жесткая фиксированность текста, являющаяся фундаментом теории и практики литературной критики.

Таким образом, речь идет о переходе от жестко фиксированного письменного текста к «мягкому» тексту на экране с мгновенной возможностью его трансформации. Особенно большие возможности для развития компьютерной культуры предоставляет цифровая технология. К примеру, американский изобретатель Дж. Джакобсон разработал суперкнигу, которая способна заменить собой тысячи письменных томов. Все ее листы в момент покупки будут чисты. Читатель выбирает то или иное литературное произведение из огромного списка на обороте обложки книги, вводит необходимый код и получает изображение нужного текста на чистых листах электронной книги. Книга позволяет подобрать соответственно остроте зрения читателя размер букв и шифры, использовать иллюстрации и звуковое сопровождение, делать пометки на полях, вставлять закладки и даже изменять текст. Подсветка позволяет читать текст, как при любом освещении, так и при полном его отсутствии.

На многих сайтах предлагаются электронные книги самой разнообразной тематики: популярные, художественные, научные, учебники, руководства по различным производственным и коммерческим процессам, произведения

русских и зарубежных классиков. В основном на сайтах интернет-магазинов можно получить нужный текст после оплаты (используется система электронных платежей), но можно найти и сайты, которые предлагают электронные книги бесплатно.

1.10 Электронные библиотеки

Современные информационные технологии позволяют не только создавать новые информационные ресурсы в электронном виде, но и приступить к широкомасштабному переводу накопленной человечеством информации в электронную форму.

В настоящее время в мире в среднем на душу населения производится ежегодно 800 мегабайт новой информации. Что такое один мегабайт информации? Объем в один мегабайт (Мб) имеет небольшой роман. Полное собрание сочинений Шекспира содержит 5 Мб. Один метр книжных полок содержит около 100 Мб информации. По этой мере в год на душу населения в мире производится информации объемом около 160 полных собраний сочинений Шекспира или 800 романов, или приблизительно 8 м книжных полок.

Рост объема электронной информации в мире имеет более интенсивный характер по сравнению с печатной. Несомненно, что работать с такими объемами информации традиционными способами (учитывать, хранить, распространять, искать и т.д.) не только не эффективно, но и невозможно.

Осознание указанных проблем, а также качественные изменения в развитии современных информационных технологий и средств получения, обработки, передачи и накопления информации различного рода привели к необходимости поиска новых подходов и решений проблем создания хранилищ информационных ресурсов, их организации, средств и способов доступа к ним пользователей. В обобщенном виде такие подходы сегодня стали трактовать как создание «цифровых» или «электронных» библиотек, развитие которых началось в конце XX века.

Электронная библиотека (ЭБ) — это управляемая коллекция информации, хранящаяся в цифровых форматах и доступная по сети в совокупности с соответствующими сервисами.

Современные электронные библиотеки являются прежде всего новым классом сложных информационных систем, возможности которых рассчитаны не на дублирование функций обычных библиотек, а на новое качество работы с неоднородной информацией.

Создание электронных библиотек и соответствующих информационных инфраструктур активно осуществляется во всём мире. Проекты по созданию электронных библиотек начинались, как правило, с оцифровки каталогов.

Работы по электронным библиотекам были начаты в США в 80-х гг., в Великобритании — в начале 90-х гг. XX века. Обычно такие работы начинались с выполнения проектов небольшими группами специалистов, но в течение нескольких лет они приобретали статус национальных программ и международных проектов. Примерами могут служить проект создания ЭБ для

стран «Большой семерки», в котором принимает участие также Россия, программы «DLI» в США и «eLib» в Великобритании. В Японии ведутся работы по реализации проекта «Электронные библиотеки XXI века». В Германии создается электронная библиотека «Global-Info».

Указанные проекты имеют существенную государственную финансовую поддержку. Множество других проектов, связанных с созданием конкретных электронных ресурсов и их программно-аппаратным обеспечением, в том числе через Интернет, поддерживаются рядом государственных научно-технических программ практически во всех странах мира. Существует также огромное количество программ, в том числе и международных, по объединению электронных ресурсов университетских библиотек.

С 2005 г. Европейской Комиссией ведутся работы по созданию Европейской цифровой библиотеки, начавшей свое функционирование на базе Европейской библиотеки *(http://www.theeuropeanlibrary.org/portal/index.html)*.

По состоянию на сентябрь 2007 г. Европейская библиотека включала цифровые фонды 23 национальных библиотек, в числе которых библиотеки Австрии, Великобритании, Хорватии, Кипра, Чешской Республики, Дании, Эстонии, Финляндии, Франции, Германии, Венгрии, Италии (Рима), Италии (Флоренции), Латвии, Литвы, Нидерландов, Мальты, Польши, Португалии, Сербии, Словакии, Словении и Швейцарии. Доступ предоставляется к 150 млн записей, и количество ссылочных цифровых коллекций постоянно растет.

В сентябре 2007 г. Российская государственная библиотека также присоединилась к Европейской цифровой библиотеке на правах полноправного участника.

В России работы в области создания ЭБ ведутся с 1995 г., в Беларуси с 1996 г., также при значительной государственной поддержке.

С конца 1998 г. при содействии Российского фонда фундаментальных исследований (РФФИ) осуществляется проект информационной и научно-методической поддержки проектов межведомственной программы РЭБ. В рамках проекта созданы и функционируют Web-сайт программы Российские электронные библиотеки (РЭБ) *(http://www.iis.ru/RDLP/);* научный электронный журнал «Электронные библиотеки» *(http://www.elbib.ru/)*.

Примером наиболее известной российской ЭБ является электронная библиотека Мошкова, существующая по адресу *http://www.lib.ru*. Она содержит более тридцати тысяч полных текстов книг, и количество их постоянно увеличивается.

Широко известна фундаментальная электронная библиотека (ФЭБ) «Русская литература и фольклор» *(http://feb-web.ru/)*. ФЭБ представляет собой сетевую многофункциональную информационно-поисковую систему, обеспечивающую сбор, хранение и распространение произведений русской словесности и результатов научных исследований в области русской литературы и фольклора.

Пять крупнейших библиотек России (Российская национальная библиотека, Российская государственная библиотека, Научная библиотека МГУ, Парламентская библиотека и Библиотека иностранной литературы)

объединили свои электронные ресурсы, создав виртуальную электронную библиотеку. В мае 2005г. была создана Российская ассоциация электронных библиотек «Электронные библиотеки России» (сайт организации расположен по адресу *http://www.elibra.ru/)*.

С 1996 г. по адресу *http://kn.ihi. com/* существует Белорусская электронная библиотека «Беларуская палічка». Это крупнейшее собрание белорусских онлайн-текстов. Сайт содержит литературные произведения белорусских писателей, а также тексты по истории и культуре Беларуси. Хотя большинство текстов в собрании на белорусском языке, часть текстов доступны на русском, польском, английском и немецком языках.

Другим видом белорусской электронной библиотеки является Белорусская цифровая библиотека *(http://library.by/)*.

Создание электронных библиотек —- весьма масштабная задача, требующая объединения усилий специалистов целого ряда областей: информатики, вычислительной техники, информационных и коммуникационных технологий, библиотечного, архивного и музейного дела и т.д. Уже сегодня она в достаточной мере продемонстрировала свой интегрирующий потенциал.

Основное достоинство электронных библиотек заключается в уверенности в том, что они позволят доставлять информацию лучше, чем в прошлом. Кроме того, электронные библиотеки, по мнению специалистов, обладают следующими потенциальными *преимуществами:*

- «доставляют» библиотеку к пользователю, т. е. информация поступает непосредственно на «рабочий стол» пользователя, будь то дома или на работе. Теперь библиотека — там, где есть компьютер, подключенный к сети;
- электронные документы удобнее, чем их бумажные аналоги, искать и анализировать;
- отсутствуют проблемы, связанные с невозможностью получить книги, занятые другими читателями; при расположении книг в сети, их количество не имеет значения;
- можно получить доступ к самым труднодоступным тестам;
- экономят значительные финансовые средства.
- в них легче поддерживать актуальность информации;
- информация доступна всегда. Материалы не могут быть выданы не тому пользователю или поставлены не на ту полку, не могут находиться в удалённом книгохранилище, не могут быть украдены. (Кроме того, исследования, проведенные в Британском университете, показали, что примерно половина посещений ЭБ приходится на то время, когда традиционные библиотеки уже закрыты);
- в электронных библиотеках доступны новые многообразные виды информации;
- пользователь электронной библиотеки, помимо того, что он обслуживается в телекоммуникационном режиме, чаще

обращается ко всему пространству Интернета для получения необходимых ему документов и данных.

В то же время создание и развитие электронных библиотек наталкивается на целый ряд *проблем:*

- быстрое старение компьютерных технологий;
- недолговечность веб-сайтов и соответственно электронных ресурсов, на них расположенных (средний срок жизни интернет-ресурса — от 4 месяцев до *2* лет);
- недолговечность современных машинных носителей информации (эта проблема в настоящее время решена только в Библиотеке Конгресса США, которая приняла решение о сохранении всех компьютеров соответствующих параметров для хранения электронных документов соответствующих форматов);
- все еще дорогостоящая инфраструктура;
- зависимость от электричества и Интернета.

Список этих проблем можно продолжать, но тем не менее электронные библиотеки уже стали реальностью современной жизни, т. е. преимущества электронных библиотек значительнее их недостатков.

В то же время печатные документы все еще остаются важной частью цивилизации, и их роль в хранении и передаче информации может снижаться лишь постепенно.

Поэтому наиболее распространенной формой современной библиотеки является так называемая *гибридная* библиотека, сочетающая в себе как фонды на бумажных носителях, так и цифровые документы. Большинство крупных современных библиотек во всем мире являются *гибридными.*

Таким образом, на смену информационному обслуживанию на печатных носителях приходит обслуживание, основанное на электронном представлении самой разнообразной информации, тиражируемой в неограниченном количестве и оперативно доступной по глобальным компьютерным сетям независимо от времени обращения к ней и местонахождения пользователей. Со своего персонального компьютера пользователь может обратиться к материалам, хранящимся на различных компьютерах в различных странах мира.

1.11 Научные электронные библиотеки (НЭБ)

Результаты научно-технической деятельности отражаются в документированной информации, т. е. оформляются в виде научных сообщений, статей, монографий и т.д.

Эффективные научные исследования сегодня немыслимы без современного информационного обеспечения. Это предполагает широкий доступ ученых к мировой научной информации и, прежде всего, к ведущим научным журналам. Информационное обеспечение является основой высокого творческого потенциала ученых, ознакомления их с новыми идеями и

тенденциями в мировой науке, развития взаимовыгодного международного научного сотрудничества.

Как показывает мировой опыт, в сфере научно-технической информации магистральный путь науки пролегает через электронные, в первую очередь — научные библиотеки.

1.11.1 Научная электронная библиотека РФФИ

Одной из крупнейших электронных научных библиотек мира является *Научная электронная библиотека Российского фонда фундаментальных исследований* (НЭБ РФФИ, *http://elibrary.ru),* не имеющая аналогов в России по числу пользователей (более 372 тысяч зарегистрированных читателей), объему и качеству предоставляемой научной информации. Она остается единственной практически реализованной некоммерческой научной электронной библиотекой общероссийского масштаба. По объему и структуре данных, степени их организованности НЭБ РФФИ является крупнейшим центром научной информации.

Научная электронная библиотека РФФИ была построена по принципу on-site представления информационных услуг через организацию технического оператора и обеспечивала бесплатный доступ к многочисленным журналам компаний Elsevier, Kluwer, Springer и ряда других издательств научной литературы для целого ряда библиотек, практически всех институтов Российской академии наук, многих университетов и вузов. В результате деятельности РФФИ тысячи российских ученых получили быстрый и эффективный доступ к зарубежной научной информации.
В настоящее время в библиотеке в полнотекстовом виде имеются 7611 периодических изданий, включающих 9,5 млн статей, а доступ к ее ресурсам стал возможен и для подписчиков из других стран СНГ и дальнего зарубежья. Среди 3664 организаций, пользующихся ее ресурсами, шесть белорусских, в том числе Центральная научная библиотека Национальной Академии наук Беларуси.

Полнотекстовая коллекция НЭБ РФФИ представлена журналами по всем отраслям знания и содержит более 500 российских журналов, часть которых (более 220 наименований) находится в открытом доступе.

Каталог бесплатных ресурсов НЭБ РФФИ можно найти по адресу: http://www.elibrary.ru/projects/subscription/RussJour_Catalogue_2007.xls.

Кроме научной периодики, НЭБ содержит целый ряд баз данных (БД) научной направленности (в частности, БД научного цитирования Science Citation Index Expanded, медицинские, математические БД, в ближайшее время планируется установка Social Citation Index и патентной БД Derwent). Эти базы данных не содержат полных текстов статей, поэтому там, где возможно, будут сделаны ссылки из их интерфейса непосредственно на полные тексты статей, если они уже имеются в НЭБ. Также делаются ссылки на полные тексты в НЭБ из внешних баз данных и поисковых систем.

Таким образом, НЭБ РФФИ фактически превратилась в мощный центр электронной научной информации общенационального. и даже

международного масштаба, став одним из немногих проектов, действительно востребованных научным сообществом, в которых наиболее отчетливо проявились преимущества современных информационных технологий. Анализ тематической направленности журналов, размещенных в этой НЭБ, показывает, что максимальным спросом потребителей пользуются естественнонаучные журналы.

1.11.2 Электронная библиотека диссертаций Российской государственной библиотеки

Не требует доказательства тот факт, что в интересах развития науки должен быть обеспечен доступ не только к научным журналам и монографиям, но и к таким видам научных публикаций, как диссертации и их авторефераты. Крайне ограниченное количество хранилищ диссертаций и небольшой тираж авторефератов существенно затрудняют возможности ознакомления с ними. Самое большое собрание диссертаций и авторефератов в СНГ находится в отделе диссертаций Российской государственной библиотеки (РГБ), располагающей фондом подлинников диссертаций, защищенных в стране с 1944 г., по всем специальностям, кроме медицины и фармации.

Необходимость обеспечения широкой доступности и сохранности этого фонда на основе современных информационных технологий и средств передачи данных привели специалистов РГБ к выводу о целесообразности создания электронной (цифровой) библиотеки диссертаций (ЭБД). *Электронная библиотека диссертаций РГБ* (http://www.diss.rsl.ru/) открыта на Web-сайте РГБ, а также в ее локальной сети, в 2004 г.

В настоящее время РГБ организовала работу нескольких десятков виртуальных залов для работы с диссертациями по всей России, а также в Беларуси (через национальную библиотеку Беларуси, а также библиотеки Витебского и Могилевского государственных университетов).

Пользователям виртуальных читальных залов диссертации или авторефераты предоставляются в электронном виде, соответственно единовременно ею могут воспользоваться очень многие.

Доступ к основному массиву электронных диссертаций ограничен: он возможен только в зале Интернет и электронных документов РГБ и в виртуальных залах электронной библиотеки РГБ, создаваемых в других организациях. Однако, кроме этого варианта, существует и доступная любому конечному пользователю возможность бесплатно в любое время со своего компьютера работать с достаточно большим количеством недавно защищенных диссертаций в открытой Русской электронной библиотеке OREL (Open Russian Electronic Library, *http://orel.rsl.ru),* также являющейся одной из электронных коллекций РГБ.

Теперь каждому автору диссертации обеспечена возможность опубликовать свой труд или его автореферат на сайте РГБ в электронной библиотеке OREL.

В свободном доступе на сайте РГБ (в открытой электронной библиотеке) находятся только те диссертации, на которые имеются соответствующие

разрешения авторов, оформленные авторскими договорами. В открытой ЭБД также имеется свой каталог.

Все электронные каталоги РГБ размещены в свободном доступе для пользователей сети Интернет.

Таким образом, Российская государственная библиотека предоставляет три возможности доступа к электронным диссертациям и авторефератам в своих электронных библиотеках: в открытой электронной библиотеке OREL на Web-сайте РГБ; в зале Интернет и электронных документов РГБ; в виртуальных залах электронной библиотеки РГБ, создаваемых в других организациях и странах по договорам с РГБ.

Вопросы для самоконтроля

1 Приведите определение понятия «информация».
2 При выполнении каких условий можно говорить об информации?
3 Перечислите качественные характеристики информации.
4 Какие науки изучают информацию?
5 Кто считается основоположником современной кибернетики?
6 Назовите современные виды кибернетики.
7 Дайте определение понятию «информационные технологии».
8 Что такое «новая информационная технология»?
9 Назовите основные виды компьютерного обеспечения.
10 Что входит в состав аппаратного обеспечения компьютера?
11 Какие классы современных компьютеров вы знаете?
12 Какие виды программного обеспечения вы знаете?
13 Что такое операционная система?
14 Какие бывают операционные системы?
15 Что входит в состав системного программного обеспечения?
16 Что такое «прикладные программы пользователей»?
17 Чем отличается открытое программное обеспечение от закрытого?
18 Перечислите преимущества открытого программного обеспечения.
19 Что такое «информатизация»?
20 Какие технологии информатизации вы знаете?
21 Перечислите современные технологии доступа к электронным информационным ресурсам.
22 Что такое «публичные центры правовой информации»?
23 Что такое «информационное общество»?
24 Что такое «электронные книги»?
25 Укажите известные вам типы электронных книг.
26 Перечислите преимущества электронных книг.
27 Назовите недостатки электронных книг.
28 Дайте определение понятию «электронная библиотека».
29 Перечислите преимущества использования электронных библиотек.
30 Назовите недостатки электронных библиотек.
31 Перечислите известные вам электронные библиотеки.

ГЛАВА 2

МАТЕМАТИЧЕСКИЕ МОДЕЛИ

2.1 Сущность математического моделирования

Моделирование - это построение и исследование моделей, т. е. условных образов систем, процессов и явлений. По свойствам модели можно судить о свойствах объекта исследования.

Различают *физическое* и *математическое* моделирование.

При *физическом моделировании* предполагается, что в качестве модели используется либо сама исследуемая система, либо иная система подобной физической природы. Типичным примером такого моделирования является модель летательного аппарата, продуваемого в аэродинамической трубе. При невозможности построения таких уменьшенных по размеру аналогов прибегают к *математическому моделированию.*

Важнейшим достоинством математического моделирования как метода познания является возможность количественного анализа, т. е. получения числовых параметров исследуемой системы. Среди следующих его преимуществ можно выделить такие:

-возможность исследования систем, физическое моделирование которых экономически не оправданно или трудноосуществимо. Например, требуется уникальное дорогостоящее оборудование;

- возможность исследования систем, физическое моделирование которых связано с опасными для здоровья человека условиями. Например, при высокой радиации или токсичности;

- исследование систем на стадии их проектирования;
- исследование труднодоступных объектов. Например, моделирование атмосферы планет;
- исследование ненаблюдаемых объектов из-за их размеров или длительности существования. Например, модели макро- и микромира;
- исследование экономических, социальных и биологических систем.

Существуют *два основных подхода* к математическому моделированию:

- формулируется задача, затем ее пытаются формализовать в виде известной математической модели, которая должна быть *адекватной* исходной задаче;
- построение наиболее адекватной математической модели и последующий поиск соответствующего метода решения.

По форме реализации математические модели подразделяют на аналитические и компьютерные.

Для *аналитического моделирования* характерно, что процесс функционирования системы записывается в виде некоторых математических соотношений. Например, в виде алгебраических или дифференциальных уравнений.

При *компьютерном моделировании* математическая модель системы представляется в виде компьютерной программы, позволяющей проводить с ней вычислительные эксперименты.

2.2 Этапы построения математической модели

***Этап* 1** *Содержательное описание системы* - словесное изложение закономерностей ее функционирования. Содержательное описание включает сведения о физической природе и количественных характеристиках элементов системы, а также о степени и характере взаимодействий между ними. Это описание составляют, как правило, специалисты соответствующей предметной области без участия математиков.

Этап 2 *Составление формализованной схемы системы* (необязательный этап.) Эта схема разрабатывается в тех случаях, когда из-за сложности системы непосредственный переход от содержательного описания к математической модели оказывается невозможным или нецелесообразным.

***Этап* 3** *Преобразование формализованной схемы в математическую модель.* Преобразование выполняется математическими методами без использования дополнительной информации о системе. Для преобразования необходимо записать в аналитической форме все соотношения и логические условия. При моделировании системы на компьютере числовой материал используется обычно не в первоначальном виде, а в форме аппроксимирующих выражений, удобных для вычислений. Например, таблицы и графики заменяют интерполяционными многочленами.

2.3 Моделирование нейронных сетей

Идея нейронных сетей возникла в результате попыток промоделировать способность биологических нервных систем обучаться и исправлять ошибки.

Нейронные сети - это модели биологических нейронных сетей мозга, в которых нейроны имитируются относительно простыми, часто однотипными, элементами (искусственными нейронами).

2.3.1 Биологический нейрон

Мозг человека состоит из серого и белого веществ. Серое вещество образуется телами и отростками нейронов (нервных клеток), а белое - нервными волокнами, т. е. частями аксонов в виде длинных отростков нейронов, покрытых белой оболочкой (рисунок 2.1).

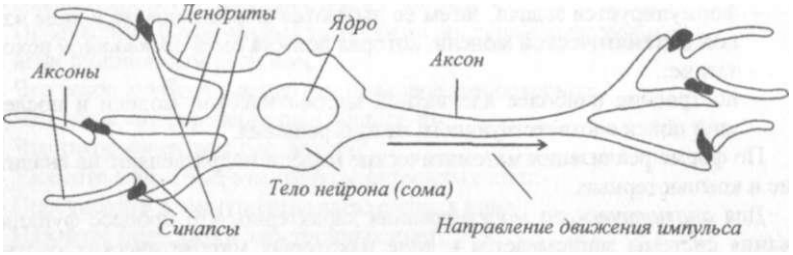


Рисунок 2.1 - Биологический нейрон

Нейрон является особой биологической клеткой, которая обрабатывает информацию. Он состоит из *тела нейрона,* или *сомы,* и двух типов древопо-

добных ветвей: *аксона* и *дендритов*. Тело нейрона включает *ядро,* которое содержит информацию о наследственных свойствах, и *плазму,* производящую необходимые нейрону материалы.

Кора головного мозга состоит из нейронов и имеет толщину от 2 до 3 мм и занимает площадь около 2200 $см^2$. Это в два раза больше стандартной клавиатуры компьютера. Размер тела нейрона составляет от 3 до 100 микрон. Длина дендрита обычно не более 1 мм. Длина аксона в отдельных случаях достигает десятков сантиметров и даже метров, как например у кальмара. (Именно благодаря наблюдению за гигантским аксоном кальмара толщиной около 1 мм стало возможным выяснить механизм передачи нервных импульсов между нейронами).

Каждый нейрон получает информацию в виде импульсов через свои дендриты и передает ее дальше через единственный аксон, разветвляющийся в конце на тысячи *синапсов.* При прохождении синапса сила импульса возрастает или ослабевает в несколько раз *(вес синапса).* Импульсы, поступившие к нейрону одновременно по нескольким дендритам, суммируются. Если суммарный импульс превышает некоторый порог, нейрон, возбуждаясь, формирует собственный импульс и передает его далее по аксону. Изменения веса синапсов вызывают изменения поведения соответствующего нейрона.

Каждый импульс представляет собой электрохимический сигнал с частотой от нескольких единиц до сотен герц. Скорость перемещения сигнала (от 100 до 1000 см в секунду) очень медленная по сравнению с современными компьютерами. Но в тоже время человеческий мозг гораздо быстрее машины может обрабатывать аналоговую информацию, например, узнавать изображения, звуки, чувствовать вкус, читать чужой почерк, оперировать качественными параметрами. Все это реализуется посредством сети нейронов, соединенных между собой синапсами.

Общее число нейронов в центральной нервной системе человека достигает 10^{10} - 10^{11}, при этом каждая нервная клетка связана в среднем с 10^3 - 10^4 другими нейронами. Установлено, что в головном мозгу на площади порядка 1 $мм^2$ совокупность нейронов формирует *относительно независимую* локальную сеть, несущую *определенную функциональную нагрузку.*

2.3.2 Свойства биологических нейронных сетей

Основные свойства биологических нейронных сетей следующие:

- *способность к обучению на примерах,* т. е. адаптация сети к условиям функционирования при неизменности ее структуры;
- *способность к обобщению.* Так, когда человеку говорят «дерево», он вспоминает не все виденные им деревья, а создает в мозгу некоторый идеальный образ абстрактного дерева. Сравнивая с ним любой объект, он может сказать, похож он на дерево или нет;
- *параллельность обработки информации.* Мы не считываем картинку по пикселям, а видим ее всю сразу, и наш мозг целиком ее обрабатывает. Другими словами, мозг - это система из *параллельных процессоров,* работающая гораздо эффективнее, чем популярные сегодня последовательные вычисления;

- *поразительная надежность мозга*. К старости некоторые структуры мозга утрачивают до 40% нервных клеток, но при этом многие люди сохраняют здравый ум и твердую память;
— *ассоциативность человеческой памяти* - способность мозга находить нужную информацию по ее малой части.

2.3.3 Пример задачи, решаемой с помощью нейронной сети

Модели нейронных сетей появились в 40-е гг. прошлого века. Для моделирования процессов, происходящих в мозгу человека, были созданы специальные аппаратные и программные средства, которые имитировали функции биологического нейрона и его соединений. Поэтому большая часть терминологии в области нейронных сетей заимствована из неврологии, включающей исследования мозга и памяти.

С помощью моделей нейронных сетей успешно решаются такие «нечеткие» задачи, как распознавание речи, печатного и рукописного текста, классификации, прогнозирования.

Рассмотрим в качестве примера решение задачи распознавания рукописных символов.

Имеется растровое черно-белое изображение рукописной буквы размером 30x30 пикселей, т. е. вектор из 900 двоичных символов (30 x 30 = 900). Требуется распознать, какая это буква (в русском алфавите 33 буквы).

Для решения задачи нужно построить нейронную сеть с 900 входами и 33 выходами, которые помечены буквами. Если на входе сети есть изображение буквы *«А»,* то максимальное значение выходного сигнала достигается на выходе «Л». Аналогично сеть работает для всех 33 букв.

2.3.4 Модель нейрона

Основным элементом искусственной нейронной сети является *формальный нейрон* (модель нейрона), который осуществляет операцию нелинейного преобразования суммы произведений входных сигналов на весовые коэффициенты, что можно представить в виде формулы (2.1).

$$y = F\left(\sum_{i=1}^{n} w_i x_i - \theta\right), \tag{2.1}$$

где $X = (x^1, x^1, \ldots, x^n)$- вектор входного сигнала; $W = (w^1, w^2, \ldots, w^n)$ - весовой вектор, компонентами которого являются веса синапсов w^i, называемые *синаптическими коэффициентами;* в - пороговое значение формального нейрона; *F*— оператор нелинейного преобразования.

Схема нейронного элемента изображена на рисунке 2.2.

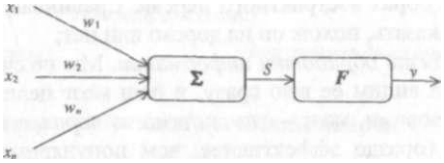


Рисунок 2.2 - Нейронный элемент

Она состоит из сумматора и блока нелинейного преобразования *F*. Каждому i-му входу нейрона соответствует весовой коэффициент w^i, который характеризует силу синаптической связи по аналог

Сумма произведений входных сигналов на синаптические коэффициенты называется *взвешенной суммой*. Она представляет собой скалярное произведение вектора весов на входной вектор (формула (2.2)):

$$S = \sum_{i=1}^{n} w_i x_i = (W, X) = |W| \cdot |X| \cdot \cos\alpha, \qquad (2.2)$$

где $\backslash W \backslash$, $\backslash X \backslash$ - соответственно длины векторов ***W*** и ***X***, $\alpha = (W, X)$ - угол между векторами ***W*** и ***X***.

Длины весового и входного векторов определяются через их координаты:

$$|W| = \sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}, \; |X| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}.$$

Если сила связи w^i является отрицательной, то такая связь называется *тормозящей*. Иначе синаптическая связь является *усиливающей*.

Оператор нелинейного преобразования *F* называется *функцией активации* нейронного элемента. Вектор входного сигнала иногда называют *паттерном входной активности* нейронной сети, а вектор выходного сигнала - *паттерном выходной активности*. Рассмотренная здесь модель формального нейрона лишь отдаленно напоминает биологический нейрон. Она используется для построения искусственных нейронных сетей.

2.3.5 Функции активации нейронных элементов

В качестве оператора нелинейного преобразования *F* могут использоваться различные функции, которые определяются в соответствии с решаемой задачей и типом нейронной сети. Рассмотрим наиболее распространенные функции активации нейронных элементов.

Пороговая функция:

$$y = \begin{cases} 0, & S < \theta; \\ 1, & S \geq \theta. \end{cases}$$

Используется в классическом формальном нейроне. Нейроны с такой функцией активации требуют малых вычислительных затрат.

Сигмоидная функция:

$$y = \frac{1}{1 + e^{-S}}.$$

Применяется в сетях с непрерывными сигналами. Получила широкое распространение, поскольку она монотонная и всюду дифференцируемая.

Гиперболический тангенс:

$$y = \text{th}\, S = \frac{e^{S} - e^{-S}}{e^{S} + e^{-S}}.$$

Также применяется в сетях с непрерывными сигналами.

Линейная функция:

$$y = kS,$$

где k - числовой коэффициент. Применяется для тех моделей сетей, где не требуется последовательное соединение слоев нейронов.

Выбор функции активации обычно связан с проектом нейронной сети для моделируемой системы. Помимо перечисленных, могут применяться и другие функции активации, которые адекватно отражают решаемую задачу. Определение функции активации и ее параметров осуществляется *экспериментальным путем*.

2.3.6 Модель нейронной сети

Совокупность нейронных элементов и связей между ними называется *нейронной сетью*. Нейроны в сетях группируются по слоям. *Слоем нейронной сети* называется множество нейронных элементов, на которые параллельно поступает информация от других нейронных элементов.

Рассмотрим нейронные сети, включающие *один слой* нейронных элементов, который обрабатывает входную информацию. Такие сети принято изображать в виде *двухслойной нейронной сети,* в которой первый слой нейронных элементов является *распределительным,* а второй - *обрабатывающим*. Распределительный слой передает входные сигналы на обрабатывающий слой, который преобразует входную информацию в соответствии с синаптическими связями и функцией активации (рисунок 2.3).

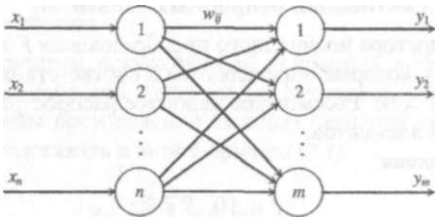


Рисунок 2.3 - Топология однослойной нейронной сети

При этом каждый нейрон распределительного слоя имеет синаптические связи со всеми нейронами обрабатывающего слоя.

Выходное значение *j-ro* нейронного элемента второго слоя можно вычислить по формуле (2.3):

$$y_j = F(S_j) = F\left(\sum_{i=1}^{n} w_{ij} x_i - \theta_j\right), \qquad (2.3)$$

где θ_j - порог *j-ro* нейронного элемента выходного слоя; w_{ij} - сила синаптической связи между i-м нейроном распределительного слоя и j-м нейроном обрабатывающего слоя.

Рассмотрим нейронную сеть с двумя нейронами входного и одним нейроном выходного слоя (рисунок 2.4).

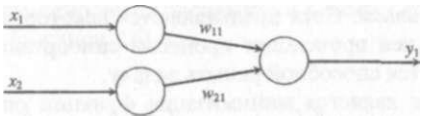


Рисунок 2.4 - Сеть с одним выходным нейроном

В этой сети взвешенная сумма имеет вид:

$$S = w_{11}x_1 + w_{21}x_2 - \theta_1.$$

Выходное значение нейронной сети (при использовании пороговой функции активации) таково:

$$y_1 = \begin{cases} 0, & S < 0; \\ 1, & S \geq 0. \end{cases}$$

Такая сеть осуществляет линейное разделение входного пространства сигналов на два класса при решении задач классификации образов. Уравнение разделяющей прямой определяется следующим образом:

$$w_{11}x_1 + w_{21}x_2 - \theta_1 = 0.$$

Эта прямая отделяет область решений, соответствующую одному классу, от другого класса и называется *дискриминантной линией*

2.3.7 Как построить нейронную сеть. Обучение сети

Процесс построения нейронной сети включает два этапа:

- выбор типа (архитектуры) сети;
- подбор весов (обучение) сети.

На первом этапе решаются следующие задачи:

- какие нейроны следует использовать (число входов, функции активации);
- каким образом следует соединить их между собой;
- что взять в качестве входов и выходов сети.

При выборе типа сети необязательно ее придумывать «с нуля». Уже насчитывается несколько десятков различных нейросетевых архитектур, причем эффективность многих из них доказана математически.

Затем необходимо «обучить» выбранную сеть, т. е. подобрать такие значения ее весов, чтобы она работала эффективно. В используемых на практике нейронных сетях количество весов может составлять несколько десятков тысяч, поэтому обучение - весьма сложный процесс.

Различают следующие три вида обучения нейронной сети.

Обучение с учителем. При таком обучении сети предъявляется набор обучающих примеров. Каждый обучающий пример представляют собой пару: вектор входных значений и желаемый выход сети. В ходе обучения весовые коэффициенты подбираются таким образом, чтобы по этим входам давать выходы, максимально близкие к правильным.

Обучение с поощрением. При этом виде обучения сети не указывается точное значение желаемого выхода, но ей выставляется оценка качества ее работы.

Обучение без учителя. Сети предъявляются некоторые входные векторы и при их обработке в ней происходят процессы самоорганизации, в результате которых она становится способной решать задачу.

Целью обучения является минимизация функции ошибок на множестве примеров путем выбора значений синаптических весов. Достижение минимума функции ошибок называется *сходимостью* процесса обучения. Известно более сотни различных обучающих алгоритмов, отличающихся друг от друга стратегией оптимизации и метрикой для вычисления погрешности обучения.

Проиллюстрируем процесс обучения с учителем по так называемому методу обратного распространения ошибки на примере обучения нейронной сети распознаванию букв русского алфавита (рисунок 2.5).

Рисунок 2.5 - Схема обучения нейронной сети

Имеется некоторая база данных, содержащая набор рукописных изображений букв. Предъявляя изображение буквы *«А»* на вход нейронной сети, получаем от нее некоторый ответ, не обязательно верный. Известен и верный ответ - в данном случае желательно, чтобы на выходе нейронной сети с меткой *«А»* уровень сигнала был максимален. Обычно в качестве желаемого выхода берут набор <1,0,0,...>, где 1 стоит на выходе с меткой *«А»,* а 0 - на всех остальных выходах. Вычисляя разность между желаемым ответом и реальным ответом сети, мы получаем 33 числа - *вектор ошибки.* После многократного предъявления примеров синаптические веса стабилизируются, причем нейронная сеть дает правильные ответы на все (или почти все) примеры из базы данных. В таком случае говорят, что «нейронная сеть *обучена»,* или «нейронная сеть *натренирована».*

2.3.8 Правило обучения Хебба

Это правило имеет биологические предпосылки и является основой многих методик обучения нейронных сетей. Согласно правилу Хебба обучение происходит в результате усиления связи (синаптического веса) между одновременно активными нейронами. Часто используемые связи (между возбужденными ней-

ронами) усиливаются, что объясняет феномен обучения живых существ путем повторения и привыкания.

Пусть имеются два нейронных элемента i и ***j,*** между которыми существует сила связи, равная w^{ij} (рисунок 2.6).

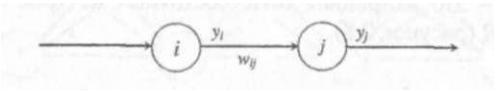


Рисунок 2.6 - Взаимосвязь двух нейронных элементов

Правило обучения Хебба для этой сети можно записать следующим образом (2.4):

$$w^{ij}(t + 1) = w^{ij}(t) + y^i y^j, \quad (2.4)$$

где *t* - номер итерации обучения; y^i и y^j - выходные значения соответственно i-го и *j-го* нейронов. В начальный момент времени предполагается, что $w^{ij}(t=0) = 0$.

Рассмотрим применение правила Хебба для простейшей нейронной сети, состоящей из двух входных и одного выходного нейрона (см. рисунок 2.4). В такой сети порог θ_1 выходного нейронного элемента является скрытым в этом элементе. При операциях с нейронными сетями порог нейронного элемента можно вынести за его пределы и изобразить как синаптическую связь с весовым коэффициентом, равным значению θ_1 (рисунок 2.7).

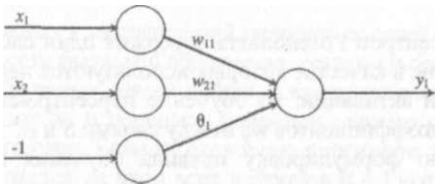


Рисунок 2.7 - Представление порогового значения в виде синаптической связи

Так как входное значение, подаваемое на дополнительный нейрон, равняется: - 1 , то взвешенная сумма определяется как:

$$S=w_{11}x_1+w_{21}x_2-\theta_1.$$

Обучение нейронной сети происходит путем настройки весовых коэффициентов и порогов нейронных элементов. Поэтому рассмотренное выше преобразование позволяет настраивать весовые коэффициенты и пороги сети как единое целое.

Правило Хебба для нейронной сети, изображенной на рисунке 2.7, можно представить в виде следующих выражений (2.5):

$$\begin{aligned} w^{11}(t + 1) &= w^{11}(t) + x^1 y^1, \\ w^{21}(t + 1) &= w^{21}(t) + x^2 y^1, \quad (2.5) \\ \theta_1(t + 1) &= \theta_1(t) - y_1. \end{aligned}$$

Аналогично правило Хебба записывается для нейронной сети большей размерности.

2.3.9 Процедура обучения персептрона Розенблатта

Персептрон - это нейронная сеть, состоящая из трех слоев нейронных элементов S, A и R (рисунок 2.8).

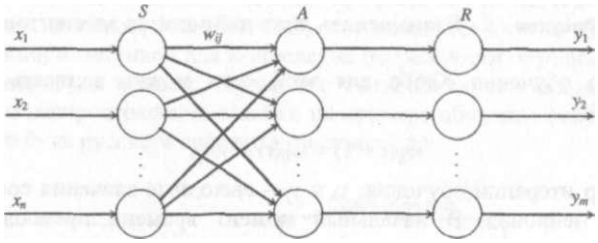


Рисунок 2.8 - Структура персептрона

Нейроны слоя S называются *сенсорными* и предназначены для формирования входных сигналов в результате внешних воздействий. Нейроны слоя *A* называются *ассоциативными* и предназначены для непосредственной обработки входной информации. Нейроны слоя *R* называются *эффекторными*. Они служат для передачи сигналов возбуждения к соответствующему объекту, например, к мышцам.

Поскольку персептрон Розенблатта содержит один слой обрабатывающих нейронных элементов, в качестве которых используются нейронные элементы с пороговой функцией активации, то обучение персептрона происходит путем настройки весовых коэффициентов w^{ij} между слоями 5 и *A*.

Математическую формулировку правила обучения Розенблатта можно представить в виде:

$$W^{ij}(t + 1) = W^{ij}(t) + \alpha x_i t_j, \quad (2.6)$$

где x^i - i-Й вход нейронной сети; t^j - эталонное значение *j-го* выхода; а - коэффициент скорости обучения сети, $0 < \alpha \leq 1$.

Процедура обучения Розенблатта называется *алгоритмом обучения с подкреплением*. Она характеризуется тем, что весовые коэффициенты нейронной сети изменяются только в том случае, если ее выходная реакция не совпадает с эталонной.

2.3.10 Многослойные нейронные сети

Нейронные сети с многочисленными слоями обладают более широкими возможностями по сравнению с однослойными. Архитектура многослойной нейронной сети состоит из множества слоев нейронных элементов (рисунок 2.9).

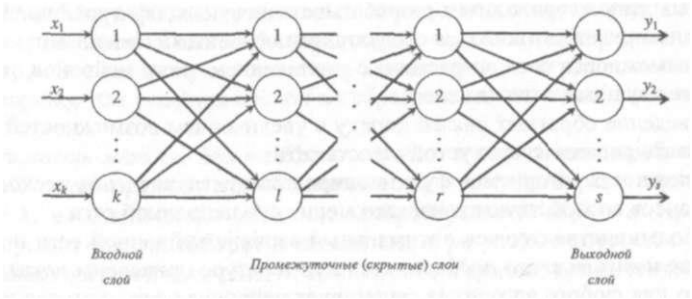


Рисунок 2.9 - Многослойная нейронная сеть

Входной слой нейронных элементов выполняет распределительные функции. *Выходной слой* нейронов служит для обработки информации от предыдущих слоев и выдачи результатов. Слои нейронных элементов, расположенные между входным и выходным слоями, называются *промежуточными,* или *скрытыми.* Как и выходной слой, скрытые слои являются обрабатывающими. Выход каждого нейронного элемента предыдущего слоя нейронной сети соединен синаптическими связями со всеми входами нейронных элементов следующего слоя.

В качестве функции активации нейронных элементов здесь обычно используются гиперболический тангенс или сигмоидная функция.

2.3.11 Классификация нейронных сетей

Одна из возможных классификаций нейронных сетей - по направленности связей. Нейронные сети бывают с *обратными связями* и *без обратных связей.*

В *сетях без обратных связей* нейроны входного слоя получают входные сигналы, преобразуют их и передают нейронам первого скрытого слоя и т. д. вплоть до выходного слоя. Если не оговорено противное, то каждый выходной сигнал i-го слоя подается на вход всех нейронов (i + 1)-го слоя; однако возможен вариант соединения i-го слоя с произвольным (i + j)-м слоем.

Главным достоинством сетей без обратных связей является простота их реализации и гарантированное получение ответа после прохождения данных по слоям.

В *сетях с обратными связями* информация с последующих слоев передается как на последующие слои, так и на предыдущие.

Существенным недостатком сетей с обратными связями является сложность обучения, вызванная большим числом нейронов.

Сети можно классифицировать также *по числу слоев* нейронов. Теоретически число слоев и число нейронов в каждом слое может быть произвольным, но фактически оно ограничено ресурсами компьютера, на котором реализуется нейронная сеть. Чем сложнее сеть, тем более сложные задачи она может решать.

Для решения отдельных типов задач уже созданы оптимальные конфигурации нейронных сетей. Если же задача не может быть сведена **ни** к одному из

известных типов, приходится разрабатывать новую конфигурацию. При этом необходимо руководствоваться следующими основными правилами:

- возможности сети возрастают с увеличением числа нейронов, плотности связей между ними и числа слоев;

- введение обратных связей наряду с увеличением возможностей сети может вызвать динамическую устойчивость сети;

- сложность алгоритмов функционирования сети, введение нескольких типов синапсов способствуют усилению мощности нейронной сети.

В большинстве случаев оптимальный вариант нейронной сети получается на основе *интуитивного подбора,* хотя в литературе приведены доказательства того, что для любого алгоритма существует нейронная сеть, которая может его реализовать.

2.3.12 Нейроматематика и нейрокомпьтеры

Нейроматематика - это раздел вычислительной математики, связанный с разработкой методов и алгоритмов решения задач с использованием нейросетевых алгоритмов. Под *нейросетевым,* или *нейронным, алгоритмом* понимают вычислительную процедуру, основная часть которой может быть реализована в виде нейронной сети на нейрокомпьютере.

Типы нейрокомпьютеров:

- большие универсальные компьютеры, построенные на множестве нейрочипов;

- нейроимитаторы, представляющие собой программы для обычных компьютеров, имитирующие работу нейронов. В основе такой программы заложен алгоритм работы нейрочипа с определенными внутренними связями.

Преимущества нейрокомпьютеров:

- высокое быстродействие, связанное с тем, что алгоритмы нейроинформатики обладают высокой степенью параллельности;

- устойчивость к помехам и разрушениям;

- устойчивые и надежные нейросистемы могут создаваться из ненадежных элементов, имеющих значительный разброс параметров.

Недостатки нейрокомпьютеров:

- нейрокомпьютеры создаются специально под конкретные задачи, решение которых на обычных компьютерах возможно только численными методами;

- нейрокомпьютеры из-за их уникальности достаточно дорогостоящи.

2.4 Статистическое моделирование. Метод Монте-Карло

Численный метод решения математических задач при помощи моделирования случайных величин называется *методом статистических испытаний,* или *методом Монте-Карло,* получившим название от известного центра игры в рулетку. (Рулетка - одно из простейших устройств получения случайных чисел, на использовании которых основан этот метод).

В основе метода лежит следующий факт: если имеется механизм генерирования значений случайной величины, равновероятно распределенной на промежутке [0, 1), то можно получить случайные значения другой случайной величины, *распределенной по любому заданному закону.*

Метод Монте-Карло применяется во многих областях науки и техники (например, в статистической физике, теории массового обслуживания, теории игр и др.). Его используют также для вычисления интегралов, в особенности многомерных, для решения систем алгебраических уравнений высокого порядка.

Сущность метода Монте-Карло состоит в следующем: требуется найти значение *a* некоторой изучаемой величины. Для этого выбирают случайную величину *X,* математическое ожидание которой равно *a: M(X) = a.* Практически же поступают так: производят n испытаний, в результате которых получают n возможных значений величины *X;* вычисляют их среднее арифметическое

$$\bar{x} = \frac{\sum x_i}{n}$$

и принимают его в качестве оценки **(т.** е. приближенного значения) искомого числа *a:* $a \approx \bar{x}$.

Поскольку при моделировании случайных элементов метод Монте-Карло требует проведения большого количества испытаний, его часто называют *методом статистических испытаний.*

2.4.1 Пример вычисления площади методом Монте-Карло

Требуется вычислить площадь плоской фигуры *S* (рисунок 2.10). Предположим, что она расположена внутри единичного квадрата со стороной *a* = 1. Выберем внутри квадрата n случайных точек:

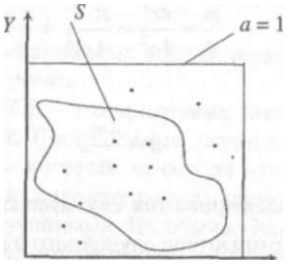


Рисунок **2.10 - Иллюстрация метода вычисления площади плоской фигуры**

Координаты этих точек являются реализациями двумерного случайного вектора, равномерно распределенного в единичном квадрате. Обозначим через m число точек, попавших внутрь *S.* Тогда, очевидно, площадь *S* приближенно равна *m/n.* Чем больше и, тем больше точность этой оценки.

2.4.2 Пример вычисления числа тс методом Монте-Карло

Рассмотрим четверть круга единичного радиуса (рисунок 2.11). Площадь четверти круга, очевидно, равна $\pi r_2/4$ или (при r = 1) равна $\pi/4$. **Площадь** же всего единичного квадрата равна 1.

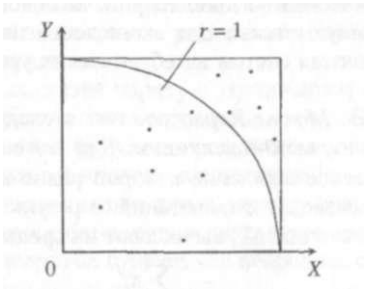


Рисунок 2.11 - Иллюстрация метода вычисления числа *л*

Будем случайным образом выбирать точки внутри квадрата со стороной *a* = **1** (0 x 1,0 y 1)и подсчитывать количество таких точек, попадающих внутрь четверти круга (x^2+y^2 1).

Пусть всего было испытано n точек, и из них m попало в четверть круга. Рассмотрим отношение количества точек, попавших в четверть круга, к количеству точек, попавших в квадрат *(m/n)*. Очевидно, что чем больше случайных точек мы испытаем, тем это отношение будет ближе к отношению площадей круга и квадрата. Таким образом, для достаточно больших и верно равенство (2.7):

$$\frac{m}{n} \approx \frac{\pi r^2}{4a^2} = \frac{\pi \cdot 1^2}{4 \cdot 1^2} = \frac{\pi}{4}, \qquad (2.7)$$

откуда следует, что:

$$\pi \approx \frac{4m}{n}.$$

2.4.3 Принципы моделирования случайных элементов

Моделирование на компьютере случайного элемента подчиняется *двум основным принципам:*

- сходство между случайным элементом-оригиналом и его моделью состоит в совпадении вероятностных законов распределения или числовых характеристик;
- всякий случайный элемент «конструируется» как некоторая функция простейших случайных элементов, называемых базовыми случайными величинами.

Простейшим для моделирования является случайный эксперимент, заключающийся в бросании точки наудачу в промежуток [0,1). Результат этого эксперимента - координата точки. Непрерывную случайную величину *R,* равномерно распределенную на полуинтервале [0, 1), называют *базовой случайной величиной* (БСВ).

Наряду с простейшим экспериментом будем рассматривать составной случайный эксперимент, получающийся в результате многократного повторения простейших экспериментов.

2.4.4 Типы датчиков базовой случайной величины

Датчик БСВ - это устройство, позволяющее по запросу получать реализацию r или несколько реализаций $r^1, r^2, \ldots, r^n$ БСВ *R*. Реализации r^i называют *случайными числами*. Существует три типа датчиков - *табличные, физические* и *программные*.

Табличный датчик - это таблица случайных чисел, т. е. выборка реализаций случайной величины, равномерно распределенной в промежутке [0, 1). Эти реализации получают экспериментально (например, с помощью специальной электронной рулетки).

Недостатки табличных датчиков:

- небольшой объем; для моделирования часто требуются миллионы случайных чисел;
- большой расход оперативной памяти компьютера для хранения таблиц.

Физический датчик БСВ - это специальное радиоэлектронное устройство, являющееся приставкой к компьютеру. Выходной сигнал этого устройства имитирует случайную величину.

Недостатки физических датчиков:

- невозможность повторения некоторой, ранее полученной, реализации r случайной величины *R;*
- схемная нестабильность, вызывающая необходимость постоянного контроля работы датчика.

По этим причинам на современных компьютерах физические датчики случайных чисел используются редко.

Программный датчик БСВ - это программа, имитирующая на компьютере реализации $r^1 r^2, \ldots, r^n$ БСВ *R*. Программные датчики строятся путем применения рекурсивных формул, по которым на основе i-го случайного числа вычисляется (i + 1)-е. Поскольку последовательность чисел вычисляется детерминированно они не являются случайными. Их обычно называют *псевдослучайными числами*. Но так как они удовлетворяют ряду требований (тестов), то их можно использовать в качестве реализаций случайной величины *R*. Сформулируем эти *требования:*

- числа равномерно распределены на промежутке [0, 1) и корреляция между ними отсутствует;
- генерируется достаточно большое количество неповторяющихся чисел, т. е. период (цикл) датчика достаточно длинный;
- последовательность случайных чисел воспроизводима, т. е. каждая их последовательность однозначно определяется начальным значением;
- датчик достаточно быстродействующий, поскольку для моделирования может потребоваться большое количество случайных чисел.

Построение программных датчиков. Рассмотрим на примере один **из** первых методов -*метод середины квадратов.*

Возьмем некоторое число r_0. Пусть $r_0 = 0{,}9876$. Возведем его в квадрат ($r^{0^2} = 0{,}97535376$). Выберем четыре средние цифры этого числа и положим $r_1 = 0{,}5353$. Затем возводим r^1 в квадрат ($r^{1^2} = 0{,}28654609$) и снова выбираем четыре средние цифры. Получаем $r_2 = 0{,}6546$. Далее находим $r^{2^2} = 0{,}42850116$, $r^3 = 0{,}8501$, $r^{3^2} = 0{,}72267001$, $r^4 = 0{,}2670$ и т. д. Последовательность чисел $r^1, r^2, \ldots$ принимают за последовательность значений БСВ.

Существуют и другие алгоритмы получения псевдослучайной последовательности.

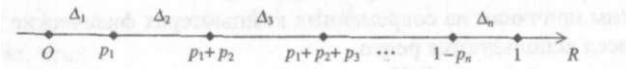
Методы построения программных датчиков просты и экономичны. Однако они имеют ряд недостатков. Наиболее существенный из них - трудность теоретической оценки статистических свойств псевдослучайной последовательности.

2.4.5 Моделирование дискретной случайной величины

Требуется промоделировать дискретную случайную величину *X,* т. е. получить последовательность ее возможных значений x^i - $(i = \overline{1, n})$, зная закон распределения величины *X:*

X	*X*	*x²*		*xⁿ*
p	*P1*	*P2*		*Pn*

Пусть *R* - БСВ, а r_i- ее возможные значения, т. е. случайные числа. Разобьем промежуток [0, 1) на оси *OR* точками с координатами p^1, p^1+p^2, $p^1+p^2+p^3, \ldots, p^1+p^2+p^3+\ldots+p^{n-1}$ на n частичных промежутков $\Delta_1, \Delta_2, \ldots, \Delta_n$:



$$\begin{aligned}
&\Delta_1 = p_1 - 0 = p_1,\\
&\Delta_2 = (p_1 + p_2) - p_1 = p_2,\\
&\Delta_3 = (p_1 + p_2 + p_3) - (p_1 + p_2) = p_3,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\Delta_n = 1 - (p_1 + p_2 + \ldots + p_{n-1}) = p_n.
\end{aligned} \qquad \textbf{(2.8)}$$

Таким образом, длина каждого промежутка с индексом ***i*** равна вероятности с тем же индексом: $\Delta_i = p_i$.

Теорема. *Если каждому случайному числу* r_i $(0 \quad r_i \quad 1)$, *которое попало в промежуток* Δ_i, *ставить в соответствие возможное значение* x^i, *то моделируемая случайная величина X будет иметь заданный закон распределения.*

Правило моделирования дискретной случайной величины. Для того чтобы промоделировать дискретную случайную величину, заданную законом распределения

X	*x¹*	*x²*		*xⁿ*
P	*p¹*	*p²*		*pⁿ*

требуется выполнить следующее.

1 Разбить промежуток [0, 1) оси OR на n частичных промежутков:

Δ_1 - $[0, p^1)$,

Δ_2 - $[p^1, p^1+p^2)$,

Δ_n - $[p^1+p^2+...+p^{n-1}, 1)$.

2 Промоделировать с помощью датчика БСВ реализацию r^i случайной величины R. Если число r^i попало в промежуток Δ_i, то моделируемая дискретная случайная величина приняла возможное значение x^i.

Пример моделирования дискретной случайной величины. Промоделировать 8 значений дискретной случайной величины X, закон распределения которой задан в виде таблицы

X	3	11	24
p	0,25	0,16	0,59

Решение.

1 Разобьем промежуток [0, 1) оси OR точками с координатами 0,25; 0,25 + 0,16 = 0,41 на 3 частичных промежутка: Δ_1 - [0; 0,25), Δ_2 - [0,25; 0,41), Δ_3 - [0,41; 1).

2 Промоделируем 8 значений БСВ: $r^1 = 0,10$; $r^2 = 0,37$; $r^3 = 0,08$; $r^4 = 0,99$; $r^5 = 0,12$; $r^6 = 0,66$; $r^7 = 0,31$; $r^8 = 0,85$.

Случайное число $r^1 = 0,10$ принадлежит промежутку Δ_1, поэтому моделируемая дискретная случайная величина приняла возможное значение $x^1 = 3$. Случайное число $r^2 = 0,37$ принадлежит промежутку Δ_2, поэтому моделируемая величина приняла значение $x^2 = 11$. Аналогично получим остальные возможные значения: $x^3 = 3$, $x^4 = 24$, $x^5 = 3$, $x^6 = 24$, $x^7 = 11$, $x^8 = 24$.

2.4.6 Моделирование противоположных событий

Пусть требуется промоделировать испытания, в каждом из которых событие A появляется с известной вероятностью p и, следовательно, не появляется с вероятностью $q = 1 - p$.

Введем в рассмотрение дискретную случайную величину X с двумя возможными значениями и соответствующими им вероятностями $p^1 = p$, $p^2 = q$. Для определенности примем $x^1 = 1$, $x^2 = 0$. Условимся считать, что если в испытании величина X приняла возможное значение $x^1 = 1$, то событие A наступило; если же $X = x^2 = 0$, то событие A не наступило, т. е. появилось противоположное событие $\bar{A}$.

Таким образом, моделирование противоположных событий A и $\bar{A}$ сведено к моделированию дискретной случайной величины X с заданным законом распределения:

X	1	0
p	p	p

Правило моделирования противоположных событий. Для того чтобы промоделировать испытания, в каждом из которых вероятность появления события

А равна *p,* а вероятность наступления противоположного события Ā равна 1 - *p,* надо выбрать с помощью датчика БСВ реализацию r^j (j = 1,2,...) случайной величины *R*; если $r^j < p$, то событие Л наступило; если r^j p, то появилось противоположное событие Ā.

Пример моделирования противоположных событий. Промоделировать 6 испытаний, в каждом из которых событие *A* появляется с вероятностью *p* = 0,35.

Решение. Выберем с помощью датчика БСВ 6 случайных чисел, например: r^1 =0,10; r^2 = 0,36; r^3 = 0,08; r^4 = 0,99 r^5 = 0,12; r^6 = 0,06. Считая, что при $r^j < 0,35$ событие *A* появилось, а при r^j 0,35 наступило противоположное событие А, получим искомую последовательность событий: А, А, Л, Л, Л, Л.

2.4.7 Моделирование полной группы событий

Напомним, что множество событий $A^1,A^2,...,A^n$ образуют *полную группу событий,* если:

- они попарно несовместны;
- появление одного и только одного из них является достоверным событием.

Например, события $A^1,A^2,...,A^6$, обозначающие число выпавших очков при подбрасывании игрального кубика, образуют полную группу.

Моделирование полной группы событий $A^1,A^2,...,A^n$ (n>2), вероятности которых $p^1,p^2,...,p^n$ известны, можно свести к моделированию дискретной случайной величины.

Правило моделирования полной группы событий. Для того чтобы промоделировать испытания, в каждом из которых наступает одно из событий $A^1,A^2,...,A^n$ полной группы с известными вероятностями $p^1,p^2,...,p^n$, достаточно промоделировать дискретную случайную величину *X* со следующим законом распределения:

X	1	2	...	*n*
p	p^1	p^1	...	p^n

Если в испытании величина *X* приняла возможное значение $x^i = i$, то наступило событие A_i.

Пример моделирования полной группы событий. Заданы вероятности четырех событий, образующих полную группу: $p_1 = P(A_1) = 0,19$; $p^2 = P(A^2) = 0,21$; $p^3 = P(A_3) = 0,34$; p^4 - $P(A^4) = 0,26$. Промоделировать пять испытаний, в каждом из которых появляется одно из четырех заданных событий.

Решение. В соответствии с правилом моделирования надо промоделировать дискретную случайную величину *X,* закон распределения которой

X	1	2	3	4
p	0,19	0,21	0,34	0,26

В соответствии с этим правилом, разобьем промежуток [0, 1) на четыре частичных промежутка: Δ_1 - [0; 0,19), Δ_2 - [0,19; 0,40), Δ_3 - [0,40; 0,74), Δ_4 - [0,74; 1). Выберем с помощью датчика БСВ пять случайных чисел, на-

пример: $r^1 = 0{,}66$; $r^2 = 0{,}31$; $r^3 = 0{,}85$; $r^4 = 0{,}63$ $r^5 = 0{,}73$. Так как случайное число $r^1 = 0{,}66$ принадлежит интервалу Δ_3, то $x = 3$, следовательно, наступило событие A^3. Аналогично найдем остальные события. Искомая последовательность событий такова: A^3, A^2, A^4, A^3, A^3.

2.4.8 Моделирование непрерывной случайной величины. Метод обратных функций

Необходимо найти последовательность значений случайной величины X, имеющей монотонно возрастающую функцию распределения *F(x)* (рисунок 2.12).

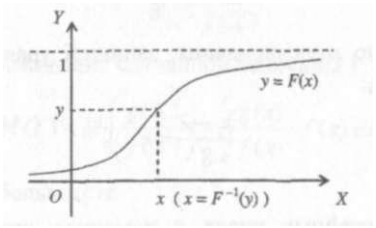


Рисунок 2.12 - График функции распределения *F(x)*

Введем случайную величину $Y=F(X)$, $0 < У < 1$. Функция распределения случайной величины *Y* имеет вид:

$$F(y)=\begin{cases}0, y<0;\\ P(Y<y), 0\le y<1;\\ 1, y\ge 1.\end{cases} \qquad (2.9)$$

Функция распределения *F(x)* монотонная, поэтому $Y \in [0, y)$ тогда и только тогда, когда $X \in [0, x)$, т. е.

$$P(Y<y) = P(X<x). \qquad (2.10)$$

Поскольку $P(X<x) = P(X < F^{-1}(y)) = F(F^{-1}(y)) = y$, *то* $F(y)$ примет вид:

$$F(y)=\begin{cases}0, y<0;\\ y, 0\le y<1;\\ 1, y\ge 1,\end{cases} \qquad (2.11)$$

т. е. *Y* - случайная величина, имеющая равномерное распределение на промежутке [0, 1). Это означает, что Y является базовой случайной величиной. Таким образом, значение *x* случайной величины *X* можно найти из уравнения $F(x) = r$, где r- значение БСВ *R*. Отсюда $x = F^{-1}(r)$.

Правило моделирования непрерывной случайной величины. Для того чтобы найти возможное значение x^i непрерывной случайной величины *X,* зная ее функцию распределения F(x), надо выбрать случайное число r^i с помощью датчика БСВ, приравнять его к функции распределения и решить относительно x^i полученное уравнение $F(x) = r^i$.

Пример моделирования непрерывной случайной величины с равномерным распределением. Промоделировать 3 возможных значения непрерывной случайной величины *X,* распределенной равномерно в интервале (2, 10).

Решение. Функция распределения случайной величины *X,* распределенной по равномерному закону в интервале

(2.12) $$F(x)=\frac{x-a}{b-a}.$$

По условию $a = 2$, $b = 10$, следовательно,

(2.13) $$F(x)=\frac{x-2}{8}.$$

Используя правило моделирования, запишем уравнения для вычисления возможных значений x^i:

$$\frac{x_i-2}{8}=r_i.$$

Отсюда $x^i = 8r^i + 2$.

Выберем три случайных числа с помощью датчика БСВ, например, $r^1 = 0{,}11$; $r^2 = 0{,}17$; $r^3 = 0{,}66$. Подставив их в последнее уравнение, получим соответствующие значения случайной величины *X:* $x^1 = 8*0{,}11+2 = 2{,}88$; $x^2 = 3{,}36$; $x^3 = 7{,}28$.

Пример моделирования непрерывной случайной величины с показательным распределением. Непрерывная случайная величина *X* распределена по показательному закону, заданному функцией распределения:

$$F(x) = 1-e^{-\lambda x} \quad (x>0),$$

(параметр λ известен). Требуется найти явную формулу для моделирования возможных значений *X.*

Решение. Используя правило моделирования, напишем уравнение:

(2.14) $$1-e^{-\lambda x_i}=r_i.$$

Решим его относительно x^i:

(2.15) $$e^{-\lambda x_i}=1-r_i,$$
$$-\lambda x_i=\ln(1-r_i).$$

Отсюда:

(2.16) $$x_i=-\frac{1}{\lambda}\ln(1-r_i).$$

Случайное число r^i заключено в промежутке [0, 1), следовательно, число $1-r^i$ также случайное и принадлежит этому

(2.17) $$x_i=-\frac{1}{\lambda}\ln r_i.$$ (2.17)

2.**4**.9 Вычисление определенного интеграла методом Монте-Карло

Рассмотрим функцию *g(x)*, определенную на интервале *(a, b)*. Требуется приближенно вычислить интеграл:

$$I = \int_a^b g(x)\,dx. \tag{2.18}$$

Выберем произвольную случайную величину *X* с плотностью распределения *fix)*, определенной на интервале *(a, b)*. Наряду с *X* рассмотрим случайную величину:

$$Y = \frac{g(X)}{f(X)}. \tag{2.19}$$

Математическое ожидание случайной величины У:

$$M(Y) = M\left(\frac{g(X)}{f(X)}\right) = \int_a^b \frac{g(x)}{f(x)} \cdot f(x)\,dx = I. \tag{2.20}$$

При достаточно больших и:

$$M(Y) \approx \frac{1}{n}\sum_{i=1}^{n} \frac{g(x_i)}{f(x_i)} \approx I. \tag{2.21}$$

Пример вычисления определенного интеграла методом Монте-Карло. Вычислим приближенно интеграл:

(2.22) $$I = \int_0^{\pi/2} \sin x\,dx.$$

Точное значение этого интеграла известно:

(2.23) $$\int_0^{\pi/2} \sin x\,dx = [-\cos x]_0^{\pi/2} = -(\cos\frac{\pi}{2} - \cos 0) = -(0-1) = 1.$$

Решение. Используем для вычисления интеграла случайную величину *X* с постоянной плотностью вероятностей f(

$$\frac{x_i - a}{b - a} = r_i \tag{2.24}$$

при $a = 0$ и $b = \pi/2$:

(2.25) $$x_i = \frac{\pi r_i}{2}.$$

Тогда формула $M(Y) \approx \frac{1}{n}\sum_{i=1}^{n} \frac{g(x_i)}{f(x_i)} \approx I$ для вычисления интеграла I примет вид:

(2.26) $$I \approx \frac{\pi}{2n}\sum_{i=1}^{n} \sin x_i.$$

Пусть n = 10. В качестве значений БСВ r_i используем следующие числа:

i	**1**	2	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**
r^i	0,865	0,159	0,079	0,566	0,155	0,664	0,345	0,655	0,812	0,332
x^i	1,389	0,250	0,124	0,889	0,243	1,043	0,542	1,029	1,275	0,521
sin x_i	0,978	0,247	0,124	0,776	0,241	0,864	0,516	0,857	0,957	0,498

Результат вычисления: $I \approx 0{,}952$.

2.4.10 Особенности метода Монте-Карло

Метод статистических испытаний Монте-Карло как численный метод решения задач характеризуют следующие особенности:

- простая структура вычислительного алгоритма. Программа составляется для осуществления одного случайного испытания, которое затем повторяется, причем каждый следующий опыт не зависит от предыдущих;
- метод особенно эффективен при решении задач с небольшой точностью результата;
- позволяет моделировать любой процесс, подверженный влиянию случайных факторов;
- для многих математических задач, не связанных со случайностями, можно искусственно создать вероятностную модель, обеспечивающую решение этих задач.

Таким образом, метод Монте-Карло можно отнести к универсальным способам решения математических задач.

Вопросы для самоконтроля

1 Перечислите основные этапы построения математической модели.
2 Что такое нейронная сеть?
3 Из чего состоит биологический нейрон?
4 Перечислите основные свойства биологических нейронных сетей.
5 Приведите пример задачи, решаемой с помощью нейронной сети.
6 Что является основным элементом искусственной нейронной сети?
7 Приведите примеры функций активации.
8 Что такое слой нейронных элементов?
9 Сколько слоев содержит однослойная нейронная сеть?
10 Перечислите существующие методы обучения нейронной сети.
11 Сформулируйте правило обучения Хебба.
12 Что такое персептрон Розенблатта?
13 Какие типы нейрокомпьютеров известны?
14 В чем состоит сущность метода Монте-Карло?
15 Приведите примеры использования метода Монте-Карло.
16 Что такое базовая случайная величина?
17 Охарактеризуйте типы датчиков базовой случайной величины.
18 К чему сводится моделирование противоположных событий?
19 Для чего используется метод обратных функций?
20 Как вычислить определенный интеграл методом Монте-Карло?

ГЛАВА 3

ПРИМЕНЕНИЕ ТАБЛИЧНОГО ПРОЦЕССОРА MICROSOFT EXCEL ДЛЯ РЕШЕНИЯ МАТЕМАТИЧЕСКИХ ЗАДАЧ

3.1 Основные возможности Excel для решения математических задач

Табличный процессор Microsoft Excel наряду с широко известными возможностями обработки данных, построения диаграмм и т.д. обладает достаточно широкими возможностями для решения сложных математических задач. Среди этих возможностей можно выделить следующие:

- решение задач статистического анализа данных;
- решение алгебраических уравнений и систем уравнений;
- поиск экстремумов функций;
- поиск минимумов и максимумов функций при наличии ограничений (задачи линейного и нелинейного программирования);
- решение задач имитационного моделирования.

3.2 Основные возможности статистического анализа данных в Excel

Для решения задач статистического анализа данных в Excel применяются инструменты, входящие в состав пакета **Анализ данных** (элемент меню **Сервис - Анализ данных),** а также набор статистических функций.

Примечание - При изложении материала данного раздела предполагается, что пользователь знаком с основными понятиями математической статистики.

3.2.1 Построение таблиц частот и гистограмм

Для построения таблиц частот и гистограмм на основе выборки в Excel применяется инструмент **Гистограмма** из пакета **Анализ данных.**

Пример 3.1 - Имеются значения показателя прочности 25 образцов некоторого материала:

32 48 62 54 59 31 29 47 42 37 51 52 43
47 61 54 53 41 42 36 48 41 49 57 46

Построить таблицу частот и гистограмму.

1 В ячейку A1 ввести заголовок "Материал А". В ячейки A2:A26 ввести данные (значения прочности образцов материала).

2 В ячейку B1 ввести заголовок "Прочность"; в ячейки B2:B7 ввести границы интервалов гистограммы (20, 30, 40, 50, 60, 70).

3 Проверить, установлен ли пакет **Анализ данных.** Для этого выбрать элемент меню **Сервис - Надстройки** и убедиться, что установлен флажок **Пакет анализа** (при необходимости установить его).

4 Получить таблицу частот и гистограмму. Для этого выбрать элемент меню **Сервис - Анализ данных.** Из полученного списка выбрать инструмент **Гистограмма.** Установить следующие параметры: **Входной интервал - A1:A26, Интервал карманов - B1:B7.** Установить флажки **Метки** (так как в ячейках A1 и B1 указаны не данные, а заголовки), **Интегральный процент** (для получения накопленных относительных частот) и **Вывод графика** (для построения гистограммы). Чтобы результаты были выведены на отдельный лист, установить переключатель **Новый рабочий лист.** Для получения результатов нажать кнопку **ОК.**

3.2.2 Вычисление статистических показателей

Для вычисления отдельных статистических показателей по выборке применяются имеющиеся в Excel статистические функции, а для вычисления всех статистических показателей сразу - инструмент **Описательная статистика** из пакета **Анализ данных.**

Пример 3.2 - По данным примера 3.1 найти основные статистические показатели прочности испытываемого материала: среднее, стандартное отклонение, дисперсию, а также статистическую ошибку и 95-процентный доверительный интервал для среднего.

1 В ячейки C1, C2, C3 ввести обозначения "Среднее", "Станд. отклон.", "Дисперсия". В ячейках D1, D2, D3 найти среднее, стандартное отклонение и дисперсию, используя функции **СРЗНАЧ, СТАНДОТКЛОН** и **ДИСП** соответственно.

2 В ячейку C5 ввести обозначение "Стат. ошибка". В ячейке D5 найти статистическую ошибку для среднего по формуле: $S_{\bar{x}} = \sigma / \sqrt{n}$, где σ - стандартное отклонение, n - объем выборки (в данном примере n*=25).* Для этого в ячейку D5 ввести формулу: **=D2/КОРЕНЬ(25).**

3 Найти границы 95-процентного доверительного интервала по формуле: $\bar{X} \pm t_{\alpha;n-1} \cdot S_{\bar{x}}$, где $\bar{X}$ - среднее; $t_{\alpha;n-1}$ - квантиль распределения Стьюдента с уровнем значимости a (в данном случае α = 0,05, так как определяется 95-процентный доверительный интервал) и π-1 степенью свободы; $S_{\bar{x}}$ - статистическая ошибка. Для этого выполнить следующее:

- в ячейке F6 найти квантиль распределения Стьюдента $t_{\alpha;n-1}$, используя функцию **СТЬЮДРАСПОБР** с параметрами **Вероятность - 0,05, Степени свободы - 24;**
- в ячейку C6 ввести обозначение "Нижняя граница", в ячейку C7 - "Верхняя граница";
- в ячейке D6 найти нижнюю границу доверительного интервала, используя формулу: **=D1-F6*D5.** Аналогично в ячейке D7 найти верхнюю границу доверительного интервала: **=D1+F6*D5.**

Пример 3.3 - Найти статистические показатели по данным из примера 3.1, используя инструмент **Описательная статистика.**

1 Выбрать элемент меню **Сервис - Анализ данных.** Из списка инструментов выбрать инструмент **Описательная статистика.** Установить следующие параметры: **Входной интервал** - A1:A26, **Группирование - По столбцам.** Установить флажки **Метки в первой строке** (так как в ячейке A1 указан заголовок), **Итоговая статистика** (для получения всех статистических показателей) и **Уровень надежности** (для получения доверительного интервала). В поле **Уровень надежности** указать доверительную вероятность в процентах: 95. Установить переключатель **Новый рабочий лист.** Для получения результатов нажать кнопку **ОК.** На новый рабочий лист выводятся значения статистических показателей, вычисленные по выборке значений прочности материала.

2 Определить границы доверительного интервала. Пусть, например, среднее находится в ячейке B3, а величина, обозначенная как **Уровень надежности** - в ячейке B16. "Уровень надежности" - это величина $t_{\alpha;n-1} \cdot S_{\bar{x}}$ (см. вычисление доверительного интервала в примере 3.2). Для определения верхней границы доверительного интервала следует в любую свободную ячейку ввести формулу **=B3+B16.** Для определения нижней границы требуется ввести формулу **=B3-B16.**

3.2.3 Оценка значимости различий между выборками

Для решения задач, связанных с оценкой статистической значимости различий между выборками, применяются инструменты, входящие в состав пакета **Анализ данных: Двухвыборочный F-тест для дисперсии, Двухвыборочный t-тест с одинаковыми дисперсиями, Двухвыборочный t-тест с различными дисперсиями.**

Пример 3.4 - Выполняется сравнение показателей прочности двух материалов (материалы A и B). Имеются значения показателя прочности 25 образцов материала A (см. пример 3.1) и 19 образцов материала B:

48	29	37	32	47	52	38	34	41	32
47	53	42	35	37	29	51	42	53	

Требуется определить, является ли статистически значимым различие между материалами по среднему значению показателя прочности.

1 Перейти на новый рабочий лист Excel. Скопировать на него столбец с данными о прочности образцов материала A (см. пример 3.1).

2 В ячейку B1 ввести заголовок "Материал B". В столбец B (ячейки B2-B20) ввести значения показателя прочности образцов материала B.

3 Выяснить, является ли статистически значимым различие между значениями дисперсии исследуемой величины (в данном примере - прочности) в двух анализируемых выборках. Для этого использовать инструмент **Двухвыборочный F-тест для дисперсии.**

Из меню **Сервис - Анализ данных** выбрать инструмент **Двухвыборочный F-тест для дисперсии.** Установить параметры: **Интервал переменной 1 - A1:A26, Интервал переменной 2 - B1:B20, Альфа - 0,05.** Установить флажок **Метки.** Выбрать переключатель **Новый рабочий лист.** *Дня* получения результатов нажать кнопку **OK.** Значимость различия по изменчивости **(т**.е. значимость различия между величинами дисперсии) оценивается по следующим величинам:

F - F-критерий Фишера для оценки значимости различия между дисперсиями;

P(F<=f) одностороннее - расчетный уровень значимости: вероятность того, что различие между величинами дисперсии двух анализируемых выборок *статистически незначимо;*

F критическое одностороннее - квантиль распределения Фишера $F_{\alpha;n1-1;n2-1}$

Если величина **F** больше, чем **F критическое одностороннее** (и **P(F<=f) одностороннее** меньше заданного уровня значимости а), то можно считать, что выборки значимо различаются по дисперсии прочности. Обычно применяется значение α=0,05.

4 Определить, является ли статистически значимым различие между выборками по среднему значению исследуемой величины (в данном примере - прочности). Для этого использовать инструмент **Двухвыборочный t-тест с одинаковыми дисперсиями** (если по результатам анализа, выполненного на предыдущем шаге, различие между дисперсиями оказалось статистически незначимым) или **Двухвыборочный t-тест с различными дисперсиями** (если различие между дисперсиями статистически значимо).

Из меню **Сервис - Анализ данных** выбрать инструмент **Двухвыборочный t-тест с одинаковыми дисперсиями** или **Двухвыборочный t-тест с различными дисперсиями.** Установить следующие параметры: **Интервал переменной 1** - A1:A26, **Интервал переменной 2** - B1:B20, **Гипотетическая разность** - 0, **Альфа** - 0,05. Установить флажок **Метки** (так как в первых ячейках каждого столбца указаны не данные, а заголовки). Установить переключатель **Новый рабочий лист.** Нажать **OK.**

Для оценки значимости различия между средними используются величины: **t-статистика** (t-критерий Стьюдента), **t критическое двухстороннее** (квантиль распределения Стьюдента $t_{\alpha;n_1+n_2-2}$) и **P(T<=t) двухстороннее** (расчетный уровень значимости *P,* представляющий собой вероятность того, что различие между средними значениями выборок статистически незначимо). Если величина **t-статистика** больше, чем **t критическое двухстороннее** (и **P(T<=t) двухстороннее** меньше заданного уровня значимости а, обычно а=0,05), то можно считать, что выборки значимо отличаются по среднему значению прочности образцов материала.

3.2.4 Корреляционный анализ в Excel

Корреляционный анализ позволяет выяснить, имеется ли значимая связь между двумя анализируемыми величинами. Для решения таких задач вычисляется коэффициент парной корреляции между анализируемыми величинами. Для этого применяется функция **КОРРЕЛ** или инструмент **Корреляция** из пакета **Анализ данных.**

Пример 3.5 - Анализируется зависимость прочности некоторого материала от содержания примесей и температуры обработки. Получены значения исследуемых величин для девяти образцов материала:

Содержание примесей, %	13,5	17,0	11,5	16,0	23,0	18,0	9,0	21,0	15,0
Температура обработки, °С	200	320	290	200	170	230	280	300	260
Удлинение образца при разрыве, мм	6,2	8,7	11,2	4,9	3,1	4,5	9,7	4,2	5,1

Найти коэффициенты парной корреляции между удлинением при разрыве и содержанием примесей, между удлинением при разрыве и температурой обработки.

1 Перейти на свободный рабочий лист. В ячейки **Al,** Bl, C1 ввести заголовки "Содержание примесей", "Температура обработки", "Удлинение при разрыве". В столбцы А, В, С ввести исходные данные.

2 В ячейку Е2 ввести обозначение "R". В ячейке F2 найти коэффициент парной корреляции (r) между удлинением при разрыве и содержанием примесей, используя функцию **КОРРЕЛ** со следующими аргументами: **Массив1 - A2:A10, Массив2 - C2:C10.**

3 Выполнить проверку статистической значимости коэффициента парной корреляции между удлинением при разрыве и содержанием примесей. По ее результатам сделать выводы о связи между исследуемыми величинами. Для этого выполнить следующее:

- в ячейке Е3 ввести обозначение "t". В ячейке F3 рассчитать значение t-критерия Стьюдента по формуле:

$$t = r\frac{\sqrt{n-2}}{\sqrt{1-r^2}},$$

где r - коэффициент парной корреляции, найденный на шаге 1; n - объем анализируемой выборки (в данном примере n=9). Для этого ввести в ячейку F3 формулу: **=F2*КОРЕНЬ(7)/КОРЕНЬ(l- F2^2);**

- в ячейку Е4 ввести обозначение "Квантиль". В ячейке F4 найти квантиль распределения Стьюдента $t_{\alpha;n-2}$. Для этого используется функция **СТЬЮДРАСПОБР** со следующими аргументами: **Вероятность - 0,05, Степени свободы - 7;**
- в ячейку Е5 ввести обозначение "Р". В ячейке F5 найти расчетный уровень значимости *P* - вероятность незначимости корреляции между

исследуемыми величинами. Для этого используется функция **СТЬЮДРАСП** со следующими аргументами: **X - ABS(F3), Степени свободы - 7, Хвосты - 2;**

Примечание - Использование функции ABS (модуль) необходимо из-за того, что аргумент X функции СТЬЮДРАСП должен быть положительным.

- по найденным значениям t-критерия Стьюдента и квантиля распределения Стьюдента, а также по расчетному уровню значимости сделать выводы о связи между исследуемыми величинами (удлинением при разрыве и содержанием примесей). Если $t > t_{\alpha;n-2}$, и $P<\alpha$ (в данном примере используется $\alpha=0,05$), то коэффициент парной корреляции между исследуемыми величинами статистически значим. Это означает, что имеется достаточно существенная линейная связь между исследуемыми величинами. При этом если $r>0$, то связь между исследуемыми величинами положительная: с ростом одной величины увеличивается и другая. Значение $r<0$ указывает на отрицательную связь между величинами: с ростом одной величины другая уменьшается.
  Если коэффициент парной корреляции статистически незначим, это означает, что линейная связь между исследуемыми величинами выражена слабо. Это может означать, что связь между величинами является более сложной (нелинейной) или вообще отсутствует.

4 Найти коэффициент парной корреляции между удлинением при разрыве и температурой обработки материала (аналогично тому, как показано выше). Проверить его статистическую значимость. Сделать выводы о связи между исследуемыми величинами.

3.2.5 Регрессионный анализ в Excel

Методы регрессионного анализа позволяют по выборкам значений двух величин **(X** и *Y)* или нескольких величин (X1 *X2, ...Xm, Y)* построить модель (уравнение), описывающую связь между этими величинами. Основной математический метод, применяемый для построения таких моделей - метод наименьших квадратов.

В Excel основным средством решения задач регрессионного анализа является инструмент **Регрессия** из пакета **Анализ данных.** Кроме того, модели связи между двумя величинами могут строиться на основе диаграмм.

Пример 3.6 - Исследуется зависимость между некоторыми двумя величинами **(X** и У). Имеются значения этих величин, полученные в десяти наблюдениях:

X	2,2	2,8	3,5	2,6	2	2,9	3,1	3,4	1,9	3,5
У	21,4	28,6	35,7	21,7	18,5	27,1	34,1	31,4	18,1	21,2

Построить линейную модель связи между исследуемыми величинами: $Y = A^0 + A^1X$, где A^0, A^1 - коэффициенты модели, которые требуется опреде-

лить, используя метод наименьших квадратов. Выполнить проверку адекватности модели и статистической значимости ее коэффициентов.

Решение задачи с помощью инструмента Регрессия

1 Перейти на свободный рабочий лист. В ячейку A1 ввести заголовок "X", в ячейку B1 - заголовок "Y". В ячейки A2:A11 и B2:B11 ввести исходные данные.

2 Из меню **Сервис - Анализ данных** выбрать инструмент **Регрессия.** Установить параметры: **Входной интервал Y** - B1:B11, **Входной интервал** X - A1:A11. Установить флажок **Метки.** В области **Параметры вывода** выбрать переключатель **Выходной интервал** и указать ячейку, с которой будет начинаться вывод результатов на текущий рабочий лист (например, ячейку A14). Для получения результатов нажать кнопку **OK.** Результаты будут иметь примерно такой вид, как показано на рисунке 3.1.

Примечание - Результаты применения инструмента **Регрессия** занимают достаточно много места. Поэтому в поле **Выходной интервал** необходимо указывать такую ячейку, чтобы снизу и справа **от** нее имелось достаточное количество свободных ячеек.

ВЫВОД ИТОГОВ

Регрессионная статистика	
Множественный R	0,735357212
R-квадрат	0,540750229
Нормированный R-квадр	0,483344008
Стандартная ошибка	4,556404212
Наблюдения	10

Дисперсионный анализ

	df	*SS*	*MS*	*F*	*Значимость F*
Регрессия	1	204,2391985	204,2391985	9,419714733	**0,015367589**
Остаток	8	173,4568015	21,68210018		
Итого	9	377,696			

	Коэффициенты	*Стандартная ошибка*	*t-статистика*	*P-Значение*	*Нижние 95%*	*Верхние 95%*
Y-пересечение	**3,794223168**	7,313234083	0,518816043	**0,617931139**	-13,07012486	20,65857119
X	**7,88020675**	2,567548977	3,069155378	**0,015367589**	1,959428197	13,8009853

Рисунок 3.1 - **Результаты решения задачи регрессионного анализа из примера 3.6**

Искомые коэффициенты модели, вычисленные по методу наименьших квадратов, находятся в столбце **Коэффициенты.** Сначала указывается коэффициент A^0, затем - A^1. В данном примере A^0=3,79, A^1=7,88 (значения коэффициентов указаны с округлениями). Таким образом, связь между величинами X и Y может быть описана следующим уравнением: $Y = 3,79 + 7,88\ X$.

Проверка адекватности модели выполняется по расчетному уровню значимости, указанному в столбце **Значимость F.** Эта величина представляет собой вероятность того, что построенная модель является неадекватной (недостаточно точной). Если расчетный уровень значимости *меньше* заданного

уровня значимости *а* (обычно используется α=0,05), то модель адекватна исходным данным (т.е. достаточно точная). В данном примере величина **Значимость F** равна (округленно) 0,015, что меньше, чем 0,05. Таким образом, построенная модель связи между величинами *X* и *Y* является адекватной. Упрощенно говоря, это означает следующее: если в построенную модель (Y = 3,79 + 7,88 *X)* подставить известные значения *X* (т.е. значения 2,2; 2,8; ...;3,5), то будут получены модельные значения У, достаточно близкие к фактическим (21,4; 28,6;...; 21,2).

Примечание - Если построенная линейная модель оказывается неадекватной (величина Значимость F превышает заданный уровень значимости), это означает, что связь между исследуемыми величинами X и Y не может быть достаточно точно описана линейным уравнением $Y = A^0 + A^1X$, т.е. является более сложной (нелинейной) или вообще отсутствует.

Проверка статистической значимости коэффициентов модели выполняется по расчетным уровням значимости, указанным в столбце **Р-Значение.** Если расчетный уровень значимости *меньше* заданного уровня значимости а, то соответствующий коэффициент модели статистически значим. В данном примере **Р-значение** для коэффициента A^0 составляет 0,618 > 0,05, для коэффициента A^1 - 0,015<0,05. Таким образом, коэффициент A^0 статистически незначим, коэффициент A^1 - статистически значим. Значимость коэффициента модели при входной переменной (в данном примере - значимость коэффициента A^1 при переменной *X)* означает, что имеется статистически значимая (т.е. достаточно четко выраженная) линейная связь величин *Y* и *X.*

Примечание - Объяснение смысла статистической значимости (или незначимости) коэффициента A^0, а также других величин, вычисляемых при использовании инструмента Регрессия (см. рисунок 3.1), требует глубокого изучения математической статистики (в частности, регрессионного анализа). В данном пособии эти величины не рассматриваются.

Решение задачи с использованием диаграммы

Требуется построить точечную диаграмму, отражающую значения исследуемых величин, и нанести на нее график линейного уравнения связи между этими величинами $Y = A^0 + A^1X$. Для этого выполнить следующее.

1 Построить точечную диаграмму, отражающую связь между исследуемыми величинами. Диаграмма строится в следующем порядке:

- выделить ячейки A1: B11;
- из меню **Вставка** выбрать команду **Диаграмма;**
- выбрать **Тип диаграммы - Точечная, Вид** - диаграмма из отдельных точек (этот вид предлагается автоматически). Нажать кнопку **Далее;**
- в окне **Источник данных диаграммы,** в поле **Диапазон данных,** должен быть указан диапазон **A1:B11,** выбран переключатель **Ряды - в столбцах.** Нажать кнопку **Далее;**
- в окне **Параметры диаграммы** настроить внешний вид диаграммы. Рекомендуется на вкладке **Заголовки** удалить содержимое поля **Название диаграммы,** в поле **Ось X (категорий)** указать "X", в поле **Ось Y (значений)** указать "Y". Перейти на вкладку **Легенда** и снять

флажок **Добавить легенду.** По окончании настройки нажать кнопку **Далее;**

- в окне **Размещение диаграммы** выбрать кнопку **Поместить диаграмму на листе - имеющемся.** Нажать кнопку **Готово.** На текущем листе будет построена диаграмма.

2 Нанести на диаграмму линейную модель связи между исследуемыми величинами (линейный тренд). Тренд строится в следующем порядке:

- выделить диаграмму щелчком мыши;
- из меню **Диаграмма** выбрать команду **Добавить линию тренда;**
- в окне **Линия тренда,** на вкладке **Тип,** выбрать вид линии тренда - **Линейная.** На вкладке **Параметры** выбрать переключатель **Название аппроксимирующей сглаживающей кривой - другое;** установить флажки **Показывать уравнение на диаграмме** и **Поместить на диаграмму величину достоверности аппроксимации (R^2).** Нажать кнопку **ОК.** На диаграмме будет построен линейный тренд, указано уравнение модели, а также коэффициент детерминации, отражающий точность построенной модели (чем ближе эта величина к единице, тем точнее построенная модель).

Пример 3.7 - По данным примера 3.5 построить линейную модель связи удлинения образца при разрыве с содержанием примесей и температурой обработки материала: $Y = A^0 + A^1X^1 + A^2X^2$ где Y - удлинение при разрыве (выходная переменная); X^1 - содержание примесей, X^2 - температура обработки материала (X^1, X^2 - входные переменные); A^0, A^1, A^2 - коэффициенты модели, которые требуется определить, используя метод наименьших квадратов. Выполнить проверку адекватности модели и статистической значимости ее коэффициентов.

Для решения задачи используется инструмент **Регрессия** точно так же, как в примере 3.6. При использовании инструмента **Регрессия** в параметре **Входной интервал X** указывается диапазон ячеек, включающий *все* значения входных величин (т.е. значения содержания примесей и температуры обработки материала).

Результаты решения данного примера приведены на рисунке 3.2. Из полученных результатов видно, что модель связи удлинения образца при разрыве *(Y)* с содержанием примесей (X^1) и температурой обработки материала (X^2) имеет следующий вид: $Y = 13{,}543 - 0{,}604X^1 + 0{,}008X^2$. Модель является адекватной: величина **Значимость F** равна 0,034, т.е. меньше, чем 0,05. Из величин, указанных в столбце **Р-значение,** видно, что коэффициенты A^0 и A^2 статистически незначимы, коэффициент A^1 - статистически значим. Из того, что коэффициент A^1 статистически значим, следует, что имеется значимая линейная связь между содержанием примеси и прочностью образца при разрыве. Следует также отметить, что эта связь - обратная (чем больше содержание примеси, тем меньше удлинение образца при разрыве), так как коэффициент A^1 - отрицательный. Статистическая незначимость коэффициента A^2 указывает, что линейная связь между температурой обработ-

ки материала и удлинением при разрыве выражена слабо. Возможно, эти величины вообще не связаны, или связь между ними более сложная (нелинейная).

ВЫВОД ИТОГОВ

Регрессионная статистика	
Множественный R	0,82199497
R-квадрат	0,67567573
Нормированный R-квадрат	0,56756764
Стандартная ошибка	2,258464037
Наблюдения	9

Дисперсионный анализ

	df	*SS*	*MS*	*F*	*Значимость F*
Регрессия	2	63,75826339	31,8791317	6,250001553	0,034114448
Остаток	6	30,60395883	5,100659805		
Итого	8	94,36222222			

	Коэффициенты	*Стандартная ошибка*	*t-статистика*	*P-Значение*	*Нижние 95%*	*Верхние 95%*
Y-пересечение	13,54252615	5,765127819	2,349041786	0,057133225	-0,564233406	27,84928571
Примесь	-0,603827216	0,189947646	-3,178913915	0,019102094	-1,068612361	-0,13904207
Температура	0,008297059	0,016042456	0,517193629	0,623526324	-0,030957417	0,047551536

Рисунок **3.2** - Результаты решения задачи регрессионного анализа из примера **3.7**

Пример **3.8** - Исследуется зависимость между некоторыми двумя величинами *(X* и У). Имеются значения этих величин, полученные в семи наблюдениях:

X	5	10	20	30	40	50	60
Y	48	32	17	14	9	5	2

Построить модели связи между величинами У и *X:* логарифмическую, степенную, экспоненциальную, линейную. Определить наиболее точную модель.

1 Для ввода исходных данных перейти на свободный рабочий лист. В ячейку A1 ввести заголовок "X", в ячейку Bl - "Y". В ячейки A2:A8 и B2:B8 ввести исходные данные.

2 Построить точечную диаграмму, отражающую связь между исследуемыми величинами (см. пример 3.6).

3 Сделать три копии построенной диаграммы. В результате на рабочем листе должны находиться четыре диаграммы (так как будут строиться четыре модели).

4 Построить на диаграммах линейный, логарифмический, степенной и экспоненциальный тренды (по одному тренду на каждой диаграмме), как показано в примере 3.6.

По максимальному значению коэффициента детерминации (R^2) определяется наиболее точная модель.

3.3 Решение уравнений и систем уравнений в Excel

Для решения уравнений в Excel используется инструмент Сервис - Подбор параметра, для решения систем уравнений - Сервис - Поиск решения.

Пример 3.9 - Решить уравнение: $60\text{-}2^x = 0{,}1$.

1 Перейти на новый рабочий лист. Выбрать любую свободную ячейку для получения решения, т.е значения переменной *x*. Пусть для этого выбрана, например, ячейка C1. В соседнюю ячейку B1 ввести подпись "X".

2 В ячейку B2 ввести подпись "Левая часть". В ячейку C2 ввести формулу, задающую левую часть уравнения: =60*2^C1.

3 Выбрать элемент меню Сервис - Подбор параметра. В появившемся окне ввести следующее:

- в поле Установить в ячейке указать ячейку, в которой задана левая часть уравнения, в данном примере - C2;
- в поле Значение ввести правую часть уравнения. Она должна представлять собой число. В данном примере требуется ввести 0,1;
- в поле Изменяя значение ячейки указать ячейку, в которой должно быть получено решение уравнения, в данном примере - C1;
- для решения уравнения нажать **OK.**

Выполняется поиск решения заданного уравнения. По окончании решения на экран выводится окно Результат подбора параметра, в котором сообщается, найдено ли решение. В этом окне следует нажать **OK.** Найденное решение выводится в ячейку C1. В данном примере *x=-9,*23.

Пример 3.10 - Решить систему уравнений:

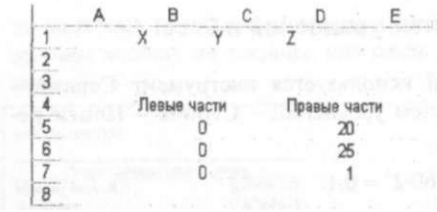
$$\begin{cases} 7x+48y-2z=20 \\ 17x+10y-8z=25 \\ xyz=1 \end{cases}$$

1 Перейти на новый рабочий лист. Выбрать любые свободные ячейки для получения решения, т.е значений переменных *x, y, z*. Пусть для этого выбраны, например, ячейки B2, C2, D2. В ячейки B1, C1, D1 ввести подписи "X", "Y", "Z".

2 В ячейку B4 ввести подпись "Левые части". В ячейки B5, B6, B7 ввести формулы, задающие левые части уравнений: в ячейку B5 - формулу =7*B2+48*C2-2*D2, в ячейку B6 - формулу =17*B2+10*C2-8*D2, в ячейку B7 - формулу =B2*C2*D2.

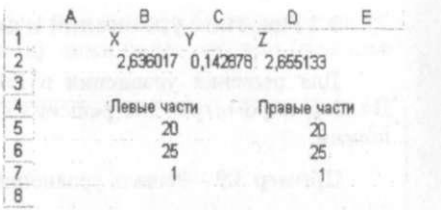
3 В ячейку D4 ввести подпись "Правые части". В ячейки D5, D6, D7 ввести правые части уравнений (20, 25 и 1). Рабочий лист с исходными данными для решения задачи будет выглядеть, как показано на рисунке 3.3.

Примечание - Подписи и обозначения на рабочем листе (X, Y, Z, "Левые части" и т.д.), показанные на рисунке 3.3, необязательны. Значения 0 в ячейках B5-B7 получены автоматически для начальных значений переменных, равных нулю.



	A	B	C	D	E
1		X	Y	Z	
2					
3					
4		Левые части		Правые части	
5		0		20	
6		0		25	
7		0		1	
8					

Рисунок **3.3 - Рабочий лист с исходными данными для примера 3.10**



	A	B	C	D	E
1		X	Y	Z	
2		2,636017	0,142878	2,655133	
3					
4		Левые части		Правые части	
5		20		20	
6		25		25	
7		1		1	
8					

Рисунок **3.4 - Рабочий лист с результатами решения примера 3.10**

4 Для решения задачи из меню **Сервис** выбрать элемент **Поиск решения.** В появившемся окне указать следующее:

- очистить поле **Установить целевую ячейку;**
  Примечание - Назначение поля Установить целевую ячейку, а также переключателя Равной максимальному/минимальному значению будет показано в подразделе 3.4.
- в поле **Изменяя ячейки** указать ячейки, в которых должны быть получены значения переменных: **B2:D2;**
- в области **Ограничения** ввести уравнения, составляющие систему. Для начала их ввода нажать кнопку **Добавить.** На экран выводится окно **Добавление ограничения. В** этом окне в поле **Ссылка на ячейку** указывается ячейка, в которой находится левая часть уравнения, а в поле **Ограничение** - правая часть уравнения (число или ссылка на ячейку, где находится правая часть уравнения). Чтобы задать первое уравнение, требуется в поле **Ссылка на ячейку** указать ячейку **B5. B** среднем поле выбрать знак равенства (=). **В** поле **Ограничение** указать ячейку **D5.** Для ввода уравнения нажать кнопку **Добавить.** Аналогично вводятся остальные уравнения. Для ввода второго уравнения требуется в поле **Ссылка на ячейку** ввести **B6,** в поле знака ввести знак =, в поле **Ограничение** ввести **D6.** Для ввода третьего уравнения требуется в поле **Ссылка на ячейку** ввести **B7,** в поле знака ввести знак =, в поле **Ограничение** ввести **D7.** По окончании ввода всех уравнений нажать **OK;**
- для решения задачи нажать кнопку **Выполнить.**

5 После появления окна с сообщением о том, что решение найдено, установить переключатель **Сохранить найденное решение** и нажать **OK.** Рабочий лист с результатами будет иметь примерно такой вид, как показано на рисунке 3.4. Решение системы уравнений находится в ячейках B2:D2.

Примечание - В некоторых случаях табличный процессор Excel не находит решения задачи при нулевых начальных значениях переменных. Обычно это происходит тогда, когда в постановке задачи где-либо используется операция деления. При нулевых значениях переменных происходит деление на ноль и выводится сообщите об ошибке. В таких случаях в ячейках, в которых определяются значения переменных, перед началом решения задачи следует указать произвольные начальные значения (например, единицы).

3.4 Поиск экстремумов функций в Excel

Для поиска экстремумов функций в Excel используется инструмент **Сервис - Поиск решения.**

Пример 3.**11** - Найти экстремум функции $y = 5x^2-8x+3$.

1 Перейти на новый рабочий лист. Выбрать любую свободную ячейку для получения решения, т.е значения переменной ***x.*** Пусть для этого выбрана, например, ячейка С1. В соседнюю ячейку В1 ввести подпись "X".

2 В ячейку В2 ввести подпись "Функция". В ячейку С2 ввести формулу, задающую функцию: **=5*C1^2-8*C1**+3.

3 Выбрать элемент меню **Сервис - Поиск решения.** В появившемся окне указать следующее:

- в поле **Установить целевую ячейку** указать ячейку с формулой функции, для которой определяется экстремум: **С2;**
- установить переключатель **Равной минимальному значению,** так как в данной задаче требуется определить минимум функции;
- в поле **Изменяя ячейки** указать ячейку, в которой должна быть получена точка экстремума: **С1;**
- для решения задачи нажать кнопку **Выполнить.**

4 После появления окна с сообщением о том, что решение найдено, установить переключатель **Сохранить найденное решение** и нажать **OK.** В ячейку С1 выводится найденное значение ***x,*** а в ячейку С2 - соответствующее ему значение ***y.*** В данном случае должно быть получено следующее решение: x=0,8, y=-0,2.

3.5 Решение задач линейного и нелинейного программирования

Рассмотрим решение задач, в которых требуется найти экстремум некоторой функции нескольких переменных при условии, что переменные, от которых она зависит, должны удовлетворять некоторым ограничениям. Ограничения на значения переменных обычно задаются в виде равенств или неравенств. Функция, для которой требуется найти экстремум, называется *целевой функцией.* Если и целевая функция, и все ограничения задачи линейны, то такая задача называется задачей *линейного программирования.* Если в постановке задачи имеется хотя бы одно нелинейное выражение, то речь идет о задаче *нелинейного программирования.*

Для решения таких задач в Excel используется инструмент **Сервис - Поиск решения.**

Пример 3.**12** - Найти значения переменных x^1, x^2, x^3, при которых достигается максимальное значение следующей функции:

$$E = 3x^1 + 5x^2 + 2x^3$$

при следующих ограничениях:

$$8x_1 + 7x_2 + 5x_3 \le 1000$$
$$2x_1 + 5x_3 \le 200$$
$$x_1 + x_2 + x_3 = 100$$
$$x_1 + x_2 \ge 20$$
$$x_1x_2 + 2x_1x_3 + x_2x_3 \ge 2500$$
$$x_i \ge 0,\ i=1,\ldots,3.$$

Здесь E - целевая функция, подлежащая максимизации. Это обычно записывается следующим образом: $E = 3x^1 + 5x^2 + 2x^3 \rightarrow max$. Рассматриваемый пример представляет собой задачу нелинейного программирования, так как содержит нелинейное ограничение $x^1x^2 + 2x^1x^3 + x^2x^3 >= 2500$.

1 Перейти на новый рабочий лист. Выбрать любые свободные ячейки для получения значений переменных x^1, x^2, x^3. Пусть для этого выбраны, например, ячейки B2, C2, D2. В ячейки B1, C1, D1 желательно ввести подписи "X1", "X2", "X3".

2 В ячейку A4 ввести подпись "Целевая функция". В ячейку B4 ввести формулу целевой функции: **=3*B2+5*C2+2*D2.** Для наглядности ввести в ячейку D4 подпись "max".

3 В ячейку A5 ввести подпись "Ограничения". Ввести формулы и правые части ограничений. Например, в ячейку B5 следует ввести формулу левой части первого ограничения: **=8*B2+7*C2+2*D2.** В ячейку C5 ввести (для наглядности) знак этого ограничения, т.е. обозначение "<=". В ячейку D5 ввести правую часть ограничения: число **1000.** В ячейку B6 ввести формулу второго ограничения **(=2*B2+5*D2),** в ячейку C6 - знак ограничения (<=), в ячейку D6 - правую часть ограничения **(200).** Аналогично ввести формулы и правые части остальных ограничений. Вид рабочего листа с исходными данными для решения задачи (с указанием всех введенных формул) показан на рисунке 3.5.

Примечание - При вводе знака равенства, указывающего вид ограничения (в данном примере - в ячейке C7), необходимо перед знаком равенства ввести в ячейку знак "Пробел", чтобы знак равенства распознавался табличным процессором Excel именно как поясняющий текст, а не как начало математической формулы.

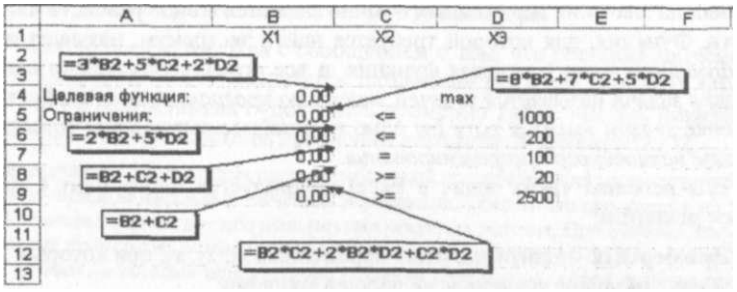


Рисунок 3.5 - Рабочий лист с исходными данными для примера 3.12

4 Для решения задачи из меню **Сервис** выбрать элемент **Поиск решения.** В появившемся окне указать следующее:

- в поле **Установить целевую ячейку** указать ячейку с формулой целевой функции: **B2;**
- установить переключатель **Равной максимальному значению,** так как требуется определить максимум целевой функции;
- в поле **Изменяя ячейки** указать ячейки, в которых должны быть получены значения переменных: B**2:D2:**
- в области **Ограничения** ввести ограничения задачи. Для начала их ввода нажать кнопку **Добавить.** На экран выводится окно **Добавление ограничения.** В этом окне в поле **Ссылка на ячейку** указывается ячейка, в которой находится формула левой части ограничения, а в поле **Ограничение** - его правая часть (число или ссылка на ячейку, где находится правая часть ограничения). Например, чтобы ввести первое ограничение, требуется в поле **Ссылка на ячейку** указать ячейку **B5,** в среднем поле выбрать знак ограничения (<=), а в поле **Ограничение** указать ячейку **D5.** Для ввода ограничения нажать кнопку **Добавить.** Аналогично ввести остальные ограничения. Чтобы указать, что все переменные должны быть неотрицательными, необходимо в поле **Ссылка на ячейку** ввести B2:D2, в поле знака ограничения выбрать ">=", в поле **Ограничение** ввести **0.** По окончании ввода всех уравнений нажать **OK;**
- для решения задачи нажать кнопку **Выполнить.**

5 В окне с сообщением о том, что решение найдено, установить переключатель **Сохранить найденное решение** и нажать **OK.** Рабочий лист с результатами решения будет иметь примерно такой вид, как показано на рисунке 3.6. Таким образом, максимальное значение целевой функции E=430,28 достигается при x^1=22,26, x^2=69,34, x^3=8,4.

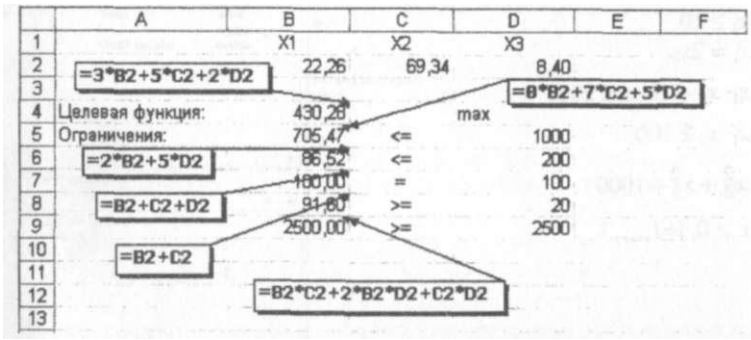


Рисунок **3.6 - Рабочий лист с результатами решения примера 3.12**

Пример 3.13 - Решить задачу из примера 3.12 со следующим дополнительным условием: значения переменных x^1, x^2, x^3 должны быть целочисленными.

Выбрать элемент меню **Сервис - Поиск решения.** Чтобы ввести дополнительное ограничение, в появившемся окне в области **Ограничения** нажать кнопку **Добавить.** В окне **Добавление ограничения** в поле **Ссылка на ячейку** ввести **B2:D2.** В поле знака ограничения выбрать отметку **"цел".** Для ввода ограничения нажать кнопку **ОК.** Дальнейшие действия выполняются так же, как показано в примере 3.12.

адачи должен быть следующим: x^1=20, x^2=70, x^3=10,

E=

$$E = \frac{e^{x_1}}{x_2 \cdot e^{x_3}} \to min$$

ить следующую задачу линейного программирова-

ния

$$\begin{cases} x_1 + x_2 + x_3 \geq 50 \\ 3x_1 + 2x_2 \leq 600 \\ x_2 \geq 10 \\ x_1 = 2x_2 \\ x_1 \cdot x_2 + x_2 \cdot x_3 \leq 2000 \\ x_1 \cdot x_2 \geq 100 \\ x_2^2 + x_3^2 = 1000 \\ x_i \geq 0,\ i=1,\ldots,3. \end{cases} \quad max$$

ть следующую задачу нелинейного программирова-

ния

ГЛАВА 4

СИСТЕМА КОМПЬЮТЕРНОЙ МАТЕМАТИКИ MATLAB

4.1 Начало работы с Matlab. Элементарные вычисления в Matlab

В данной главе приводятся некоторые основные сведения, необходимые для начала работы с системой компьютерной математики Matlab.

Стандартный вид окна Matlab показан на рисунке 4.1. Видно, что окно Matlab состоит из трех основных частей:

- командное окно **(Command Window)** - область ввода команд и вывода получаемых результатов. На рисунке 4.1 в командном окне показаны две команды: присваивание значений переменной *a* и матрице b*;*
- рабочая область **(Workspace)** - область отображения всех переменных, используемых Matlab в текущем сеансе работы. На рисунке 4.1 это переменная **a,** равная пяти, и матрица b**,** содержащая одну строку и 91 столбец;
- окно истории команд **(Command History)** - перечень команд, выполнявшихся ранее.

Одним из элементов панели инструментов (в верхней части экрана), на который следует обратить внимание, является поле **Current Directory:** в нем указывается папка, которая будет по умолчанию использоваться для всех операций с файлами.

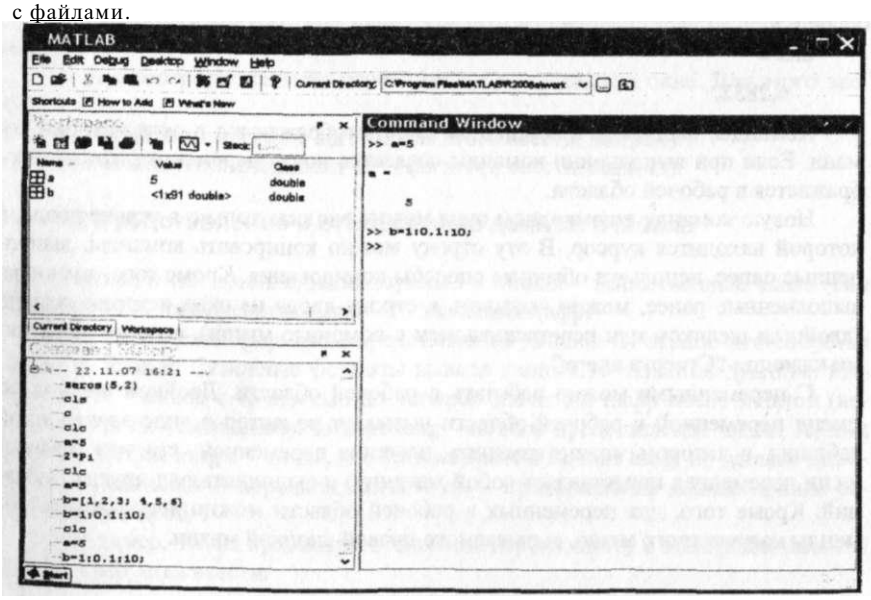


Рисунок 4.1 - Стандартный вид окна Matlab

Вид окна можно изменять с помощью команд из меню **Desktop,** а также используя стандартные возможности управления окнами.

Вид команд, вводимых в командном окне, достаточно несложен. Например, если ввести

```
x=2*pi
```

то на экран выводится следующий результат:

```
x =

   6.2832.
```

При этом выполненная команда отображается в окне истории команд, а в рабочей области создается переменная **x.**

Если команда завершается точкой с запятой, то ее результат не выводится на экран. Например, если ввести

```
x=2*pi;
```

то переменная **x** получит такое же значение, что и при выполнении команды **x=2*pi** (без точки с запятой), но результат не будет выведен на экран.

Если при выполнении команды не указана переменная, которой должен быть присвоен результат, то он по умолчанию присваивается переменной ans. Например, если ввести

```
2*pi
```

то на экран выводится следующий результат:

```
ans =

   6.2832.
```

Команды, вводимые в командном окне, отображаются в окне истории команд. Если при выполнении команды создается новая переменная, то она отображается в рабочей области.

Новую команду в командном окне можно вводить только в строке ввода, в которой находится курсор. В эту строку можно копировать команды, выполненные ранее, используя обычные способы копирования. Кроме того, команды, выполненные ранее, можно вызывать в строку ввода из окна истории команд (двойным щелчком или перетаскиванием с помощью мыши), а также нажатиями клавиши "Стрелка вверх".

С переменными можно работать в рабочей области. Двойной щелчок по имени переменной в рабочей области вызывает редактор в виде электронной таблицы, в котором можно изменять значение переменной, строить графики (если переменная представляет собой матрицу) и выполнять ряд других операций. Кроме того, для переменных в рабочей области можно использовать команды контекстного меню, вызываемого правой кнопкой мыши.

Имена переменных могут начинаться как с заглавной, так и со строчной буквы. Следует обратить внимание, что заглавные и строчные буквы в именах переменных *различаются.* Например, переменные с именами **x** и **X** -*разные.*

Имена функций (например, sin), стандартных констант (например, **pi,** inf) и другие зарезервированные слова указываются строчными буквами.

Имеется возможность просмотра краткой подсказки но любой функции. Для этого используется команда **help.** Например, для получения подсказки о функции **sin** следует в окне команд ввести: **help sin.**

Для очистки командного окна используется команда clc.

Пример 4.1 - Выполнить приведенные ниже элементарные расчеты.

1 Вычислить длину окружности радиусом 10 см. Для этого в окне команд ввести: **2*pi*10.** Результат присваивается переменной **ans** и выводится на экран.

2 Нажатием клавиши "Вверх" вызвать на экран предыдущую команду. Исправить ее, чтобы вычислить длину окружности радиусом 12 см.

3 Присвоить переменной **x** значение **8,3.** Для этого ввести: **x=8.3.**

4 Присвоить переменной **x** новое значение 20, не выводя результат на экран. Для этого ввести: **x=20;** (точка с запятой необходима, чтобы результат не выводился на экран).

5 Присвоить переменной **y** значение кубического корня из **x.** Для этого ввести: **y=x^(1/3).**

6 В рабочей области выбрать двойным щелчком мыши переменную **x** и изменить ее значение на 30.

7 Используя контекстное меню, удалить из рабочей области переменную ans.

8 Просмотреть значение переменной **x** в командном окне. Для этого ввести: **x.**

9 Вычислить значение sin π/4. Для этого ввести: sin**(pi/4).**

10 Вычислить значение sin 45°. Для этого ввести: **sind(45).**

4.2 Представление и отображение данных в Matlab

Основной тип данных, используемый в Matlab - вещественные числа (тип **double).** Точность представления - 15 значащих цифр.

Для управления форматом представления данных на экране используется команда **format.** Основные форматы вывода данных на экран следующие: **format short** - вывод с точностью до четырех значащих цифр после запятой (используется по умолчанию); **format long** - полное представление числа; **format short e, format long e** - то же, что **format short** и **format long,** но данные выводятся с мантиссой и порядком; **format rat** - представление данных в виде рациональной дроби.

Например, чтобы просмотреть значение переменной **y** в виде рациональной дроби, требуется ввести:

format rat

у

Выбранный формат применяется до тех пор, пока не будет введена команда **format** с другим форматом.

Для вывода данных в произвольном формате используется функция **fprintf,** аналогичная одноименной функции языка С. Например, функция **fprintf('%6.2f\n',y)** выводит на экран значение переменной у, выделяя для нее шесть позиций, в том числе две - для дробной части. Обозначение **\n** указывает, что после вывода переменной требуется переход в следующую строку.

Пример 4.2 - В командном окне просмотреть значение переменной у во всех вышеуказанных форматах.

4.3 Сохранение значений переменных и информации о ходе работы в Matlab

4.3.1 Сохранение значений переменных

Содержимое рабочей области (т.е. значения всех переменных) можно сохранить на диске, используя команду меню **File - Save Workspace As...,** или команду **save имя файла** в командном окне. Файл с содержимым рабочей области сохраняется с расширением **.mat.** При выходе из Matlab содержимое рабочей области теряется, поэтому, прежде чем завершать сеанс работы в Matlab, необходимо сохранить рабочую область.

Для загрузки сохраненного содержимого рабочей области используется команда меню **File - Open** или команда **load имя файла** в командном окне.

Имя файла в командах **save** и **load** задается по обычным правилам, например: **save d:\lab\ivanov\rab_oblast.mat.**

Если при сохранении или загрузке файла не указывается путь, то используется папка, указанная в поле **Current Directory** (на панели инструментов).

4.3.2 Сохранение информации о ходе работы в Matlab

Все команды, вводимые в Matlab (по мере их ввода), а также результаты их выполнения, выводимые в командном окне, можно сохранять в текстовом файле. Для этого требуется ввести команду **diary имя_файла** (например, **diary d:\lab\ivanov\seansl.txt).**

Для прекращения вывода команд и результатов в файл используется команда **diary off.**

Пример 4.3 - Выполнить приведенные ниже операции по сохранению информации о ходе вычислений и получаемых результатов.

1 Используя команду **diary имя файла,** предусмотреть сохранение всех последующих команд и результатов в файле.

2 Выполнить следующие расчеты:

- присвоить переменной kor значение квадратного корня из 5;
- присвоить переменной **tg** значение тангенса 45°.

3 Прекратить сохранение команд и результатов в файле **(diary off).**

4 Выполнить следующий расчет: увеличить значение переменной kor в два раза **(**kor=kor***2).**

5 Сохранить в файле содержимое рабочей области.

6 Выйти из Matlab.

7 Просмотреть файл, созданный командой **diary.** Убедиться, что в нем сохранены все команды и их результаты до команды **diary off.**

8 Загрузить Matlab.

9 Открыть файл рабочей области.

4.4 Операции с матрицами

Система Matlab обладает широкими возможностями для операций с матрицами. Само название этой системы обозначает "Matrix Laboratory" (буквально - "матричная лаборатория"). Строго говоря, все данные в Matlab рассматриваются как матрицы. Даже обычная переменная представляет собой матрицу размером 1 x 1 .

Примечание - Термин "матрица" в Matlab применяется к матрице любой размерности. Для одномерных матриц (строк и столбцов) применяется также термин "массив".

4.4.1 Способы задания матриц

Применяются следующие простейшие способы задания матриц:

- перечислением;
- в виде диапазона значений (с помощью операции "двоеточие");
- с помощью специальных функций.

Пример **4.4** - Задать следующие матрицы:

$$a=(3\ \ 6\ \ 4),\ b=\begin{pmatrix}5 & 7 & 9\\ 6 & 2 & 1\end{pmatrix}.$$

Для этого ввести:

a=[3 6 4]

b=[5 7 9; 6 2 1]

Как видно из данного примера, матрицы вводятся *в квадратных скобках.* Если матрица двумерная, то она вводится *по строкам;* конец *строки* обозначается *точкой с запятой.* Элементы матрицы разделяются *пробелами.* Вместо пробела в качестве разделителя элементов матриц можно использовать *запятую,* например:

a=[3,6,4]

b=[5,7,9; 6,2,1]

Пример 4.5 - Задать одномерную матрицу (строку), содержащую числа от 0 до **10** с шагом **0,1.**

Для этого ввести: **x=0:0.1:10;**

Точка с запятой в конце команды требуется, чтобы созданная матрица (из **101** элемента) не выводилась на экран.

Пример 4.6 - Создать три матрицы размером 4 x 2 (четыре строки, два столбца): из нулей, из единиц и из случайных чисел.

Для этого ввести:

u=zeros(4,2)

v=ones(4,2)

w=rand(4,2)

Здесь zeros, ones и rand - функции для создания массивов заданного размера, состоящих из нулей, единиц и случайных чисел соответственно.

4.4.2 Операции с элементами, строками и столбцами матриц

В Matlab имеются удобные возможности для операций с отдельными элементами матриц, а также их строками и столбцами, включая просмотр, добавление, удаление и т.д. Такие операции могут выполняться с отдельными строками и столбцами, их диапазонами и матрицами в целом. Рассмотрим эти операции на следующем примере.

Пример 4.7 - Выполнить следующие операции с элементами, строками и столбцами матриц.

1 Удалить из рабочей области все матрицы, созданные в предыдущих примерах.

2 Задать следующие матрицы, как показано выше:

$$c=\begin{pmatrix}4 & 8 & 5 & 7\\ 3 & 1 & 2 & 5\\ 5 & 9 & 6 & 1\end{pmatrix},\quad d=\begin{pmatrix}5 & 2\\ 1 & 6\\ 4 & 7\end{pmatrix},\quad e=\begin{pmatrix}4 & 2 & 8 & 1\\ 3 & 7 & 5 & 2\end{pmatrix}.$$

3 Просмотреть на экране элемент матрицы c, расположенный во второй строке, третьем столбце. Для этого ввести: **c(2,3).** На экран должно быть выведено число **2.** Из этого примера видно, что при ссылке на конкретный элемент матрицы первым указывается номер *строки,* затем - номер *столбца.* Номера заключаются в *круглые* скобки.

4 Присвоить элементу матрицы новое значение: **c(2,3)=8.**

5 Просмотреть на экране вторую строку матрицы **c: c:(2,:).**

6 Просмотреть на экране третий столбец матрицы **c: c(:,3).**

7 Получить вектор-столбец **v,** состоящий из элементов третьего столбца матрицы **c: v=c(:,3).**

8 Выделить строки **2-3** и столбцы **2-4** матрицы **c** в новую матрицу **w: w=c(2:3,2:4).**

9 Получить новую матрицу к, добавив матрицу d к матрице с справа: k=[c,d].

10 Получить новую матрицу z, добавив матрицу с к матрице **e** снизу: z=[e;c].

Как видно из данного примера, обозначение : (двоеточие) указывает на диапазон элементов матрицы (или на всю строку, или весь столбец). Запись вида **[матрица_1, матрица_2]** означает, что **матрица_2** добавляется к **матрице_1** справа; запись **[матрица_1; матрица_2]** - добавление **матрицы_2** к **матрице_1** снизу. При этом матрицы должны соответствовать друг другу по размерам. Например, операции [c, **e]** или [c; d] невозможны из-за несоответствия размеров матриц. Попытка выполнить их привела бы к ошибке.

4.4.3 Вычисления с матрицами

В Matlab имеются возможности для самых разнообразных операций с матрицами: от поэлементного сложения до наиболее сложных операций матричной алгебры. Рассмотрим эти операции на следующем примере.

Пример 4.8 - Выполнить следующие вычисления с матрицами.

1 Удалить из рабочей области все матрицы, созданные в предыдущих примерах.

2 Задать следующие матрицы, как показано выше:

$$a=\begin{pmatrix}3 & 2 & 8\\ 4 & 6 & 5\end{pmatrix},\quad b=\begin{pmatrix}4 & 1 & 7\\ 9 & 3 & 8\end{pmatrix},\quad c=\begin{pmatrix}4 & 8 & 5 & 7\\ 3 & 1 & 9 & 5\\ 5 & 9 & 6 & 1\end{pmatrix},\quad x=\begin{pmatrix}3 & 7 & 5\\ 4 & 1 & 9\\ 5 & 3 & 6\end{pmatrix}.$$

3 Увеличить все элементы матрицы **b** на единицу: **b=b+1.**

4 Сложить матрицы **a** и b, присвоив результат матрице **u: u=a+b.**

5 Перемножить матрицы **a** и **b** *поэлементно,* присвоив результат матрице **v: v=a.*b.** Точка перед знаком умножения обозначает поэлементную операцию.

6 Извлечь квадратный корень из всех элементов матрицы **b: sqrt(b).**

7 Перемножить матрицы **a** и с *по правилам матричной алгебры:* **z=a*c.**

Примечание - Для умножения по правилам матричной алгебры матрицы должны соответствовать друг другу по размерам: количество столбцов первой из умножаемых матриц должно быть равно количеству строк второй.

8 Транспонировать матрицу b, т.е. поменять местами строки и столбцы: **b=b'.**

9 Вычислить определитель матрицы **x:** d=det(x).

10 Вычислить матрицу, обратную к **x: xobr=inv(x).**

11 Чтобы убедиться, что обратная матрица найдена правильно, перемножить матрицы **x** и **xobr** по правилам матричной алгебры, как показано выше. Результатом должна быть единичная матрица, т.е. матрица, в которой диагональ состоит из единиц, остальные элементы - нули.

12 Вычислить собственные значения матрицы **x: eig(x).**

13 Вычислить собственные векторы и собственные значения матрицы **x.** Для этого ввести: **[k,s]=eig(x).** Вычисляются две матрицы: **к** - матрица, столбцы которой представляют собой собственные векторы матрицы **x;** s - матрица, диагональ которой состоит из собственных значений матрицы **x.**

Пример 4.9 - Решить систему линейных уравнений:

$6x^1 + 14x^2 + 7x^3 = 120$

$3x^1 + 5x^2 + 9x^3 = 175$

$8x^1 + 3x^2 + 5x^3 = 100.$

Для решения системы уравнений вычислить столбец значений переменных **x** следующим образом: $x=a^{-1}b$, где **a** - матрица коэффициентов уравнений (a^{-1} - обратная матрица), **b** - столбец правых частей уравнений. Для проверки полученного решения перемножить матрицы **a** и **x;** должны быть получены правые части уравнений.

4.5 Построение графиков

Возможности Matlab для построения и оформления графиков достаточно многообразны. В данном разделе на примерах рассматриваются основные из них.

Основным средством построения несложных графиков вида *y=f(x)* является функция **plot(x**,y,**'строка'),** где **x** и y - матрицы (обычно- одномерные), задающие координаты точек, по которым строится график; **'строка'** - набор управляющих символов, задающих вид линии графика (необязателен). Кроме того, в окне графика имеется собственная система меню для настройки его внешнего вида.

Пример 4.10 - Построить график функции y=0,25*+sin(x)-1 для значений * от 0 до 10.

1 Получить массив значений переменной * от 0 до 10 с шагом 0,1. Для этого ввести: **x=0:0.1:10;** (точка с запятой требуется, чтобы на экран не выводились все полученные величины).

2 Получить массив соответствующих значений переменной ***y:*** **y=0.25*x+sin(x)-l;.**

3 Для построения графика ввести: **plot(x,y).**

Пример 4.11 - Настроить внешний вид графика, полученного в примере **4.10.**

1 Перейти в окно графика.

2 Выбрать команду **Edit - Axes Properties.** Вызывается окно редактора свойств осей **(Property Editor - Axes).** Для осей X и Y ввести буквенные обозначения X и Y (поле **Label),** а также установить по обеим осям сетку (флажок **Grid).**

3 Закрыть окно **Property Editor.**

4 Используя кнопку **Edit Plot** (), перейти в режим редактирования графика.

Примечание - Если кнопка **Edit Plot** отсутствует в окне графика, следует выбрать команду **View** и установить панель инструментов **Figure Toolbar.**

5 С помощью мыши переместить обозначения **X** и **Y** на концы осей.

6 Повернуть обозначение оси **Y** в вертикальное положение. Для этого в режиме редактирования графика выбрать обозначение **Y.** Выбрать команду **Edit - Current Object Properties,** или щелкнуть правой кнопкой мыши и выбрать из контекстного меню команду **Show Property Editor.** Появляется окно **Property Editor - Text.** В этом окне нажать кнопку **More Properties.** В появившемся окне **Property Inspector** установить для параметра **Rotation** значение 0. Закрыть окна **Property Inspector** и **Property Editor.**

7 Используя команду **Insert - Text Box,** нанести на график подпись: **Y**=0**,25X+sin(X)-l.** В результате график должен иметь такой вид, как показано на рисунке **4**.2.

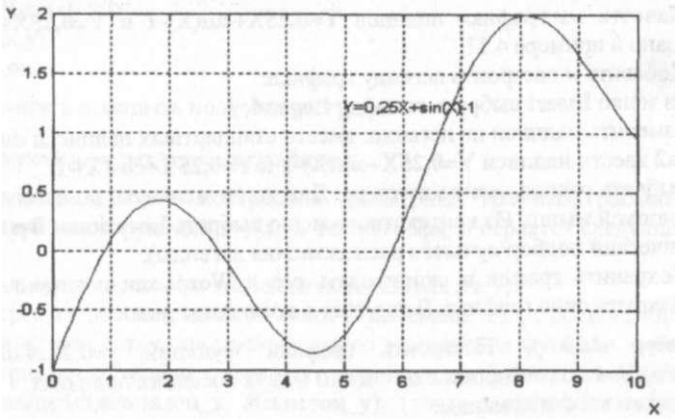


Рисунок **4.2 - Окончательный вид графика для примера** 4.10

8 Используя команду **Edit - Copy Figure,** скопировать график в буфер обмена. Загрузить текстовый редактор Word и вставить график в текстовый файл. Сохранить файл в Word.

9 Сохранить график, используя команду **File - Save.** График сохраняется в файле с расширением **.fig.**

10 Закрыть окно графика. Вернуться в командное окно.

Пример 4.12 - Построить в одном окне графики функций y**=0**,25x+sin(x)-1 и y**=0**,25x+cos(x)-1 для значений x: от **0** до **10.**

1 Получить массив значений переменной **x** от **0** до **10** с шагом **0,1,** а также массивы **yl** и y2 с соответствующими значениями функций, как показано в примере **4.10.**

2 Для построения графиков ввести последовательность команд:

```
plot(x,yl,'b')
hold on
plot(x,y2,'r')
grid on
```

Здесь **'b'** и **'r'** - цвета графиков (синий и красный). Команда **hold on** требуется для того, чтобы очередной график строился в том же окне, что и предыдущий. Команда **grid on** устанавливает на графике сетку.

3 Перейти в окно построенного графика, чтобы настроить его внешний вид.

4 Изменить линию одного из графиков на пунктирную. Для этого перейти в режим редактирования графика (кнопка **Edit Plot).** Щелкнуть правой кнопкой мыши на одном из графиков. Из контекстного меню выбрать команду **Line Style.** Выбрать тип линии **Dash.**

5 Нанести на графики подписи **Y=0,25X+sin(X)-l** и **Y=0,25X+cos(X)-l,** как показано в примере 4.11.

6 Добавить и настроить легенду графика:

- из меню **Insert** выбрать команду **Legend;**
- изменить надписи на легенде: вместо стандартных надписей **data1** и **data2** ввести надписи **Y=0,25X+sin(X)-l** и **Y=0,25X+cos(X)-l;**
- выбрать расположение легенды. Для этого щелкнуть по легенде правой кнопкой мыши. Из контекстного меню выбрать **Location - Best** (автоматический подбор лучшего расположения легенды).

7 Сохранить график и скопировать его в Word, как показано в примере 4.11. Закрыть окно графика. Вернуться в командное окно.

Пример 4.13 - Построить графики функций y=0,25x+sin(x)-1 и y=0,25x+cos(x)-l для значений *x* от 0 до 10 в двух соседних подокнах.

Для этого ввести команды:

```
subplot (1,2,1); plot(x,yl,'b'); text(0,0,'y=0,25x+sin(x)-l')
grid on
subplot (1,2,2); plot(x,y2,'r'); text(0,0,'y=0,25x+cos(x)-l')
grid on
```

Через точку с запятой перечисляются несколько команд, вводимых в одной строке.

Команда **subplot(a,b,n)** разбивает окно графиков на подокна. Здесь **a** - количество подокон по горизонтали, **b** - количество подокон по вертикали, n - номер подокна, в котором будет строиться очередной график. Команда **text(x,y,'строка')** выводит указанную строку в точке с координатами **x** и y.

Сохранить и закрыть построенный график.

Пример 4.14 - Построить график функции, заданной в полярных координатах: r=6cos(3t), для значений ***t*** от 0 до 2π.

Для этого ввести:

```
t=0:0.1:2*pi;
```

```
r=6*cos(3*t);
poIar(t,r)
```

Здесь **polar** - функция для построения графиков в полярных координатах. Сохранить и закрыть полученный график.

Пример 4.15 - Построить график параметрической функции:

$x=6\cos^3 t$
$y=6\sin^3 t$

для значений *t* от 0 до 2π.

Для этого ввести:

```
t=0:0.1:2*pi;
x=6*(cos(t)).^3;
y=6*(sin(t)).^3;
pIot(x,y)
grid on
```

Сохранить и закрыть полученный график.

Построение трехмерных графиков

В качестве примера построения трехмерных графиков рассмотрим построение графиков функций *z=f(x,y).* Такие графики строятся следующим образом:

- задаются диапазоны значений переменных *x* и y;
- строятся две матрицы значений переменных *x* и *y,* составляющие координатную сетку для последующего вычисления функции *z=f(x,y)* и построения ее графика. Для этого используется функция **meshgrid: [x,y] = meshgrid(диапазон_x, диапазон_y);** (точка с запятой в конце строки желательна, так как матрицы координатной сетки **x** и **y,** получаемые в результате применения функции **meshgrid,** обычно достаточно велики);
- вычисляются значения функции *z=f(x,y)* (матрица **z);**
- строится график функции *z=f(x,y):* **plot3(x,y,z).**

Пример 4.16 - Построить график функции $z=x^2+y^2$ для $0<=x<=10,5<=y<=7$.

1 Построить матрицы координатной сетки:

```
[x,y]=meshgrid(0:0.1:10, 5:0.1:7);
```

2 Вычислить матрицу значений функции г:

```
z=x.^2+y.^2;
```

3 Построить график функции z*:*

```
plot3(x,y,z)
```

4 Используя возможности настройки графиков, рассмотренные в примере 4.11, получить график такого вида, как показано на рисунке 4.3. Для вращения графика в пространстве использовать кнопку **R o t a t e 3 D .**

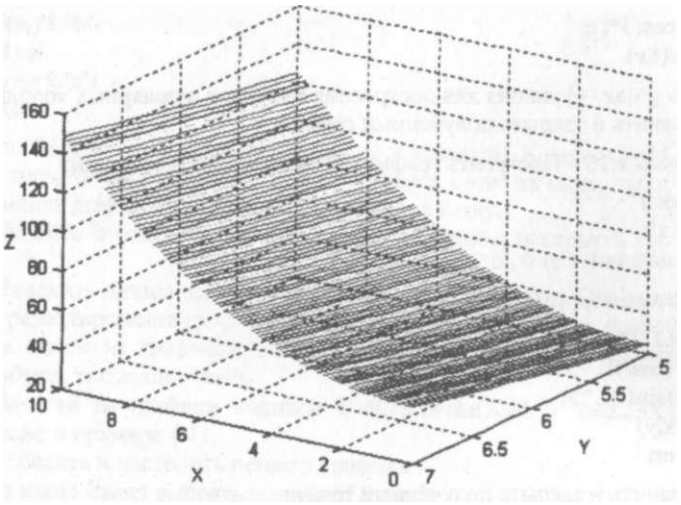


Рисунок 4.3 - Окончательный вид графика для примера 4.16

4.6 Основы программирования в Matlab

Система компьютерной математики Matlab имеет собственный язык программирования высокого уровня, включающий как все обычные возможности традиционных языков программирования, так и все математические возможности Matlab. Подробное рассмотрение вопросов программирования вообще и программирования в Matlab в частности не входит в задачу данного пособия. В данном подразделе приводятся лишь основные сведения о программировании в Matlab.

Вместо термина "программа", в Matlab чаще применяется термин "М-файл", так как файлы с текстами программ сохраняются в Matlab с расширением **.m.** Для подготовки текста нового М-файла следует использовать команду **File - New** - М**-file,** для загрузки существующего М-файла и его последующего редактирования - команду **File - Open.**

В Matlab имеются два основных вида М-файлов: файлы-сценарии и файлы-функции.

4.6.1 Файлы-сценарии

Файл-сценарий представляет собой набор команд Matlab, сохраненный в файле.

После того, как файл-сценарий подготовлен и сохранен, для его выполнения требуется указать имя данного файла-сценария в командном окне или в другом М-файле. Файл-сценарий не имеет входных или выходных параметров. При выполнении файла-сценария используются и изменяются переменные ра-

бочей области, как если бы команды, составляющие файл-сценарий, просто вводились в командном окне.

Пример 4.17 - Разработать файл-сценарий для выделения последнего столбца произвольной матрицы в отдельную матрицу.

Подготовка текста файла-сценария

Для подготовки текста М-файла требуется выбрать команду File - New - M-file. Вызывается редактор М-файлов. В данном примере текст М-файла может быть следующим:

```
% Выделение последней строки в отдельную матрицу
[m,n]=size(a);
b=a(:,n);
```

Символ "%" в М-файлах является признаком комментария.

Функция size(имя_матрицы) определяет размеры заданной матрицы. Точнее, результатом выполнения функции size является матрица (строка) из двух элементов, первый из которых - количество строк заданной матрицы, второй - количество столбцов. В результате переменная m получит значение, равное количеству строк матрицы a, а переменная n - количеству ее столбцов.

Последняя команда в файле-сценарии выделяет из матрицы с именем a последний столбец. Подробнее о таких операциях см. в п. 4.4.2.

Для сохранения М-файла выбирается команда File - Save. Сохраним этот М-файл, например, под именем stolbec.m.

После сохранения М-файла можно закрыть окно редактора М-файлов и вернуться в командное окно.

Использование файла-сценария

Прежде чем использовать созданный файл-сценарий, необходимо создать матрицу с именем a, которая будет обрабатываться с помощью этого сценария. Например, введем в командном окне следующую матрицу из трех строк и четырех столбцов: a=[4,8,5,7;3,1,9,5;5,9,6,1].

Следует обратить внимание, что матрица, которую предполагается обрабатывать с помощью созданного файла-сценария, должна иметь имя **a,** так как это имя указано в файле-сценарии.

Чтобы выделить из введенной матрицы последний столбец в отдельную матрицу, требуется ввести в командном окне имя файла-сценария, т.е. слово stolbec.

В результате выполнения файла-сценария в рабочей области создаются три новые переменные: m=3, n=4, b=[7; 5; 1]. Если до выполнения файла-сценария переменные с такими именами уже имелись в рабочей области, то их прежние значения теряются.

Пример 4.18 - Разработать файл-сценарий для построения графика функции y=0,25x+sin(x)-l. Границы диапазона значений переменной *x* должны задаваться переменными, вводимыми в командном окне.

4.6.2 Файлы-функции

Файл-функция представляет собой программу, обычно имеющую входные и выходные параметры. Файл-функция обрабатывает величины (переменные или константы), переданные ему в качестве входных параметров, и возвращает переменные, указанные как выходные параметры.

При выполнении файла-функции переменные рабочей области не изменяются и не используются, если это не предусмотрено в самом файле-функции. Другими словами, все переменные файла-функции локальны. Например, если в рабочей области имеется переменная с именем **x,** и в файле-функции имеется переменная с тем же именем, то любые операции с переменной **x** в файле-функции никак не влияют на ее значение в рабочей области (конечно, если при вызове файла-функции переменная **x** не была указана в качестве выходного параметра).

Первая строка (заголовок) файла-функции имеет вид:

function [выходные_параметры] = имя файла (входные_параметры)

Здесь function - зарезервированное слово. Имя_файла - это имя М-файла, в котором сохраняется функция.

Пример **4**.19 - Разработать файл-функцию для решения квадратного уравнения. Функция должна вычислять как вещественные, так и комплексные корни. Если корни комплексные, то выводится сообщение. Кроме корней уравнения, функция должна возвращать код результата: 1 - уравнение имеет два вещественных корня, 0 - один вещественный корень, -1 - корни комплексные.

Подготовка текста файла-функции

Для подготовки текста файла-функции, как и файла-сценария, используется редактор М-файлов. В данном примере текст файла-функции может быть следующим:

```
function [x,kod]=quadur(a,b,c)
% Решение квадратного уравнения
% Вызов: quadur(a,b,c), или x=quadur(a,b,c), или |x,k]=quadur(a,b,c)
% a,b,c - коэффициенты уравнения
% x- корни уравнения
% к - код результата (1 - уравнение имеет два вещественных корня,
% 0 - один вещественный корень, -1 - корни комплексные)
n=nargin;
if n~=3
   errtor('Неверное количество аргументов')
end
d=b^2-4*a*c;
if d~=0
   x(1)=(-b+sqrt(d)/(2*a);
   x(2)=(-b-sqrt(d)/(2*a);
else
   x=-b/(2*a);
end
```

```
if d<0
    kod=-l;
elseif d = 0
    kod=0;
else
    kod=l;
end
```

Сохраним этот файл-функцию под именем **quadur.m.**

Комментарий, указанный в начале файла-функции (сразу же после строки **function),** включается в систему подсказок Matlab. Это значит, что если в командном окне ввести команду **help quadur** (конечно, после того, как приведенный выше файл **quadur.m** сохранен), то комментарий, приведенный в файле **quadur.m,** выводится на экран.

Использованная в М-файле функция **nargin** - стандартная функция, возвращающая количество аргументов выполняемой функции (в данном примере - функции **quadur).** В данной функции предусмотрена проверка: если количество аргументов отличается от трех, значит, при вызове функции допущена ошибка. **Error** - команда прерывания функции с выводом заданного сообщения.

Из текста М-файла видно, что для проверки на неравенство в Matlab используется обозначение "~=" (условие **if d~=0** означает: "если переменная **d** не равна нулю"). Проверка на равенство обозначается двумя знаками "равно" (условие **elseif d = 0)** в отличие от операции присваивания, обозначаемой одним знаком "равно".

Использование файла-функции

Пусть требуется решить уравнение $-5x^2+2x+1=0$. Приведем возможные способы вызова функции **quadur** для его решения.

Если в командном окне ввести

```
quadur(-5,2,l)
```

то переменная **ans** получает значение матрицы из двух элементов - корней уравнения (0,6899 и -0,2899). Код результата (второй выходной параметр функции **quadur)** не возвращается, так как по умолчанию файл-функция возвращает лишь первый выходной параметр.

Если в командном окне ввести

```
x=quadur(-5,2,1)
```

то матрица из двух корней уравнения присваивается переменной **x.** Код результата также не возвращается, так как при вызове функции указана только одна выходная величина (переменная **x).**

Если же в командном окне ввести

```
[x,k]=quadur(-5,2,l)
```

то переменная **x** будет содержать корни уравнения, а переменная **к** получит значение, равное коду результата (в данном случае - значение 1). Таким образом, файл-функция возвращает столько выходных параметров, сколько указано

в его заголовке. По умолчанию возвращается только первый выходной параметров.

Пример 4.20 - Разработать функцию для решения системы линейных уравнений. В функцию должны передаваться две матрицы: матрица коэффициентов уравнений (двумерная матрица) и матрица правых частей (вектор-столбец). Если количество уравнений *меньше,* чем количество неизвестных, то "лишние" переменные (последние по порядку) принимаются равными нулю. Если количество уравнений *больше,* чем количество неизвестных, то лишние уравнения (последние по порядку) отбрасываются.

Если количество строк в матрице коэффициентов уравнений не равно количеству элементов в матрице правых частей, то выводится сообщение об ошибке, и функция прерывается.

Пример 4.21 - Разработать функцию для вставки строки в матрицу. Входными параметрами являются исходная матрица, вставляемая строка и номер строки, после которой требуется вставить новую строку. Если номер вставляемой строки превышает количество строк в исходной матрице, то строка должна добавляться в конец матрицы. Если длина вставляемой строки не совпадает с количеством столбцов исходной матрицы, то должно выводиться сообщение об ошибке.

Пример 4.22 - Разработать функцию для вставки столбца в матрицу. Входными параметрами являются исходная матрица, вставляемый столбец и номер столбца, после которого требуется вставить новый столбец. Если номер вставляемого столбца превышает количество столбцов в исходной матрице, то столбец должен добавляться в конец матрицы (справа). Если длина вставляемого столбца не совпадает с количеством строк исходной матрицы, то должно выводиться сообщение об ошибке.

4.6.3 Основные управляющие структуры для программ в Matlab

В качестве справочного материала приведем основные конструкции, используемые для управления выполнением программы в Matlab.

Условный оператор:

```
if условие_1
команды_1
elseif условие_2
команды_2
else
команды 3
end
```

Здесь **команды 1, команды 2** и **команды_3** - произвольные наборы команд Matlab, выполняемые при соответствующих условиях.

Цикл "до":
for переменная=начальное**_значение:шаг:конечное_значение**
команды
end

Если шаг равен единице, то его можно не указывать.

Цикл "пока":
while условие
команды
end

Для прерывания цикла используется команда break.

Переключатель:
switch выражение
case значение_1
команды_1
case значение_2
команды_2

otherwise
команды
end

Если выражение равно значению**_1,** то выполняются команды**_1;** если выражение равно значению**_2,** то выполняются команды**__2** и т.д. Если выражение не равно ни одному из указанных значений, то выполняются команды, указанные после слова otherwise.

Команда ввода (ввод значения переменной x):
x=input('Введите переменную:');

Команда вывода (вывод значения переменной *y):*
disp('Значение Y равно '); disp(y)

4.7 Решение алгебраических уравнений

Основная функция для решения алгебраических уравнений вида f(x)=0 - функция fzero. Она может применяться в двух формах:

fzero('уравнение', начальная_точка)

или

fzero('уравнение', [a b]),

где 'уравнение' - левая часть решаемого уравнения *f(x)=0* или имя М-файла, реализующего функцию *f(x);*
начальная_точка - значение переменной, в окрестности которого ищется решение;
a, b - границы отрезка, на котором ищется решение, при этом величины *f(a)u f(b)* должны иметь разные знаки.

Если уравнение имеет несколько решений, то функция **fzero** находит лишь одно из них. Другие решения требуется определять, изменяя значения начальной точки или границы отрезка (a или b**).**

Если в уравнении f(x)=0 функция *f(x)* представляет собой полином, то можно найти все решения уравнения сразу, используя функцию **roots(коэффициенты),** где **коэффициенты** - массив коэффициентов полинома.

Пример 4.23 - Найти все решения уравнения 0,25x+sin(x)-l=0 для значений *x* от 0 до 10.

Предварительно следует построить график функции y=0,25x+sin(x)-l, чтобы примерно определить, где находятся решения уравнения. График будет иметь такой вид, как показано на рисунке 4.2.

Создадим М-файл (функцию), реализующий левую часть уравнения:

```
function y=fun(x)
y=0.25*x+sin(x)-l;
```

Сохраним его под именем fun.m.

Для определения первого решения уравнения можно ввести:

```
x=fzero('fun', 1)
```

На экран выводится результат: x=0.8905.

Конечно, вместо 1 можно указать и другое значение. Можно также вместо имени М-файла указать левую часть уравнения:

```
x=fzero('0.25*x+sin(x)-l',1)
```

Аналогично можно найти остальные решения, например:

```
x=fzero('fun', 3)
```

На экран выводится: x=2.8500.

```
x=fzero('fun', 5)
```

На экран выводится: x=5.8128.

Можно также использовать функцию **fzero** с несколькими выходными параметрами, например:

```
[x,y,k]=fzero('fun', 1)
```

На экран выводятся следующие результаты: x=0,8905, y=0, k=1. Здесь y - значение функции y=0,25x+sin(x)-1 при найденном x (то, что это значение равно нулю, подтверждает, что найдено верное решение); **r** - код результата, возвращаемый функцией **fzero; r**=1 означает, что решение найдено.

Найдем решения уравнения 0,25x+sin(x)-l=0, используя функцию **fzero** в форме fzero('**уравнение',** [a b**]).** Например, чтобы найти первое решение, можно ввести:

```
x=fzero('fun', [0 1])
```

Пример 4.24 - Найти все решения уравнения $5x^3+8x^2+1=0$.

Для этого требуется ввести: **roots([5,8,0,l]).**

4.8 Решение систем алгебраических уравнений

Основная функция для решения систем алгебраических уравнений - функция fsolve**('система_уравнений', начальная_точка),** где **'система уравнений'** - имя М-файла (функции), реализующего левые части уравнений системы; **начальная_точка** - массив, задающий начальную точку для поиска решения.

Для использования функции **fsolve** решаемую систему уравнений необходимо преобразовать к виду, где правые части уравнений представляют собой нули.

Пример 4.25 - Решить систему уравнений:

$$x1^2 - x1x2 = e^{-X1}$$
$$x2^2 - x1x2 = e^{-X2}.$$

Для этого требуется сначала преобразовать систему:

$$x1^2 - x1x2 - e^{-X1} = 0$$
$$x2^2 - x1x2 - e^{-X2} = 0.$$

Создадим функцию (М-файл), реализующую левые части уравнений решаемой системы:

```
function z=sist_ur(x)
z=[x(l)^2-x(1)*x(2)- exp(-x(1)); x(2)^2+x(1)*x(2)- exp(-x(2))];
```

Для решения системы уравнений следует ввести:

```
[x,y,k]=fsolve('sist_ur',[0 0])
```

Здесь **[0 0]** - начальная точка для поиска решения системы уравнений.

На экран выводятся следующие результаты:

```
x =
   0.9066    0.4611
У =
  1.0e-011 *
   0.2307
  -0.0560
к =
     1
```

Это означает, что решение системы уравнений следующее: $x1=0,9066$, $x2=0,4611$. Переменная *y* представляет собой массив правых частей уравнений при найденных значениях переменных *x*. Видно, что в данном случае правые

части уравнений практически равны нулю, что подтверждает правильность найденного решения. Переменная r - код результата, возвращаемый функцией fsolve; r=1 означает, что решение системы уравнений найдено.

Примечание - Функцию fsolve можно использовать и для решения одиночных уравнений (например, для уравнения из примера 4.23).

4.9 Поиск экстремумов функций одной переменной

Для поиска *минимумов* функций одной переменной *y=f(x)* используются следующие функции Matlab:

fminsearch('функция',x^0)

или

fminsearch(функция', a, Ь),

где 'функция' - функция *f(x),* для которой требуется найти минимум, или имя М-файла, реализующего эту функцию;

x^0 - значение переменной, в окрестности которого ищется минимум;

a, b - границы отрезка, на котором ищется минимум.

Если функция *y=f(x)* имеет несколько минимумов, то функции fminsearch и fminbnd находят лишь один из них. Другие минимумы требуется определять, изменяя значения x^0, a или b.

Если требуется найти *максимум* функции *y=f(x),* то необходимо использовать функции Matlab fminsearch или fminbnd, указав в них функцию *f(x),* умноженную на -1.

Примечание - Функция fminsearch может применяться также для поиска экстремумов функций нескольких переменных (см. подраздел 4.10).

Пример **4**.26 - Найти все точки экстремума функции y=0,25x+sin(x)-1 для значений *x* от 0 до 10, используя функцию fminbnd.

Предварительно следует построить график функции y=0,25x+sin(x)-1, чтобы примерно определить, где находятся точки экстремума. График этой функции показан на рисунке 4.2.

Из графика видно, что на заданном отрезке функция имеет один минимум и два максимума. Для поиска точки минимума следует ввести:

[x,y,k]=fminbnd('fun'**,4**,5)

М-файл fun.m подготовлен при решении примера 4.23. Минимум ищется на отрезке от 4 до 5 (конечно, можно задать и другие границы). На экран выводятся следующие результаты: x=4,4597, y=-0,8533, к=1. Это означает, что минимум достигается при x=4,4597, при этом y=-0,8533. Код результата к=1 означает, что минимум найден.

Для поиска *максимумов* следует создать еще один М-файл, где исследуемая функция будет указана *с обратным знаком,* например:

```
function y=fun_minus(x)
y=-(0.25*x+sin(x)-l);
```

Сохраним его под именем **fun_minus.m.**

Например, для поиска максимума на отрезке от 1 до 2 следует ввести:

```
[x,y,k]=fminbnd('fun_minus',l,2)
```

Результат должен быть следующим: x=4,4597, y=0,8533. Аналогично определяется еще одна точка максимума: x=8,1066, *y*= 1,9945.

Пример 4.27 - Найти все точки экстремума функции y=0,25x+sin(x)-1 для значений *x* от 0 до 10, используя функцию **fminsearch.**

Например, *дня* поиска первой точки максимума (в окрестности точки x=4) следует ввести:

```
[x,y,k]=fminsearch('fun_minus',2)
```

Аналогично определяются другие точки экстремума.

Примечание - Вместо имени М-файла в функциях **fminsearch** и **fminbnd** можно указывать саму функцию, для которой требуется определить экстремум, например: **[x,y,k]=fminsearch('-(0.25*x+sin(x)-l)',2).**

Пример 4.28 - Найти все точки экстремума функции $y = 5 - 2/x - x^2 + 10 \sin x$ и все решения уравнения $5 - 2/x - x^2 + 10\sin x = 0$ для значений *x* от -10 до 10.

4.10 Поиск экстремумов функций нескольких переменных

Для поиска минимумов функций нескольких переменных используется функция Matlab **fminsearch('функция*,** x^0), где x^0 - массив, задающий начальную точку для поиска решения.

Пример 4.29 - Найти точку минимума следующей функции двух переменных: $f(x^1,x^2) = (x^1-5)^2(x^2-2)^2 - (x^1-5)(x^2-2)$.

Создадим М-файл, реализующий исследуемую функцию:

```
function f=fun2(x)
f=((x(1)-5)^2)*((x(2)-2)^2)-(x(1)-5)*(x(2)-2);
```

Сохраним этот М-файл под именем **fun2.m.**

Для поиска экстремума следует в командном окне ввести:

```
[x,y,k]=fminsearch('fun2',[0 0])
```

Здесь в качестве начальной точки для поиска экстремума используется начало координат. Результат должен быть следующим: x^1=0,3534, x^2=1,8924, y=-0,2500.

4.11 Решение задач линейного программирования

Понятие задачи линейного и нелинейного программирования рассмотрено в подразделе 3.4. В Matlab процедура решения задач линейного и нелинейного программирования существенно различается. Решение задач линейного программирования рассматривается в данном подразделе, а задач нелинейного программирования - в подразделе 4.12.

Пример **4**.30 - Решить следующую задачу линейного программирования:

$E = 5x^1 + 4x^2 + 12x^3 \rightarrow max$

$2x^1 + 6x^2 + x^3 >= 100$
$3x^1 + 2x^2 + 4x^3 <= 1000$
$3x^1 + 2x^2 <= 600$
$x^1 <= 20$
$x^2 >= 10$
$x^1 = 2x^2$
$x^1 + x^2 + x^3 = 1$

$x^i >= 0, i=1,...,3.$

Здесь требуется найти значения переменных x^1 x^2, x^3, при которых целевая функция E будет иметь максимальное значение при соблюдении всех указанных ограничений.

Чтобы решить такую задачу в Matlab, требуется привести ее к следующей стандартной форме:

- целевая функция должна подлежать минимизации;
- все ограничения-неравенства должны иметь вид <=;
- все ограничения-равенства должны иметь в правой части число (а не переменную и не выражение).

Примечание - Ограничения на минимальные и максимальные значения отдельных переменных могут иметь вид как <=, так и >=

Приведем к стандартной форме решаемую задачу:

$-E = 5x^1 + 4x^2 + 12x^3 \rightarrow min$

$-2x^1 - 6x^2 - x^3 <= -100$
$3x^1 + 2x^2 + 4x^3 <= 1000$
$3x^1 + 2x^2 <= 600$
$x^1 <= 20$
$-x^2 <= -10$
$x^1 - 2x^2 = 0$
$x^1 + x^2 + x^3 = 1$

$x^i >= 0, \quad i=1,...,3.$

Для решения задач линейного программирования в Matlab используется функция

linprog(f, a, b, a^r, b^r x^{min}, x^{max})

где **f** - массив-столбец коэффициентов целевой функции, подлежащей минимизации;

a - матрица коэффициентов ограничений-неравенств, имеющих вид "меньше или равно" (коэффициенты каждого ограничения указываются в отдельной строке матрицы);

b - массив-столбец правых частей ограничений-неравенств;

a^r - матрица коэффициентов ограничений-равенств (коэффициенты каждого ограничения указываются в отдельной строке матрицы);

b^r - массив-столбец правых частей ограничений-равенств;

x_{min} - массив-столбец ограничений на минимальные значения переменных;

x_{max} - массив-столбец ограничений на максимальные значения переменных.

Так как для функции **linprog** требуется задавать много исходных данных, удобно не вводить их в командном окне, а подготовить эти данные в М-файле (сценарии) и вызывать функцию **linprog** из этого же М-файла. Следует вызвать редактор М-файлов (командой **File - New - M-file)** и ввести следующий текст:

```
f=[-5;-4;-12];
a = [-2,-6,-1;3,2,4;3,2,0;1,0,0;0,-1,0];
b = [-100; 1000; 600; 20; -10];
ar = [1, -2,0; 1,1,1];
br = [0; 1];
xmin = [0; 0; 0];
[x, e, k] = linprog(f, a, b, ar, br, xmin)
```

В данном случае не используется массив ограничений на максимальные значения переменных. В последней строке (т.е. в вызове функции **linprog)** не указана точка с запятой, чтобы результаты выводились на экран.

Здесь использованы три выходных параметра функции **linprog: x** - массив значений переменных x, представляющих собой решение задачи; e - значение целевой функции (в данном примере оно будет получено с обратным знаком, так как целевая функция умножалась на -1 для приведения к стандартной форме); **к** - код результата, возвращаемый функцией **linprog (k=l,** если решение успешно найдено).

Необходимо сохранить этот файл (например, под именем lp.m), закрыть редактор М-файлов и вернуться в командное окно. Для решения задачи следует в командном окне ввести lp.

Результат должен быть следующим: x^1=*20,* x^2=10, x^3=70, E=980.

Рассмотренный способ решения задачи - не единственно возможный. Можно, например, указать ограничение x^2 >= 10 как одно из ограничений на минимальные значения переменных, ax^1 <= 20 - как ограничение на максимальное

значение переменной. Тогда М-файл для вызова функции **linprog** будет иметь следующий вид:

```
f=[-5; -4; -12];
a = [-2,-6, -1;3, 2, 4; 3,2,0];
b = [-100; 1000; 600];
ar = [1,-2,0; 1,1,1];
br = [0; 1];
xmin = [0; 10; 0];
xmax = [20; inf; inf];
[x, e, k] = linprog(f, a, b, ar, br, xmin, xmax)
```

Здесь **inf** - стандартное обозначение бесконечности. Оно используется потому, что для переменных x^2 и x^3 ограничения на максимальные значения не заданы.

Примечание - Если в задаче отсутствуют ограничения-равенства, то функция **linprog** вызывается следующим образом:

linprog(f, a, b, [ ], [ ],x^{min}, x^{max})

Другими словами, если требуется пропустить какие-либо входные параметры функции, то вместо них указываются пустые матрицы ([]). Это относится не только к функции **linprog,** но и ко всем функциям Matlab.

Пример 4.31 - Решить следующую задачу линейного программирования:

$E = 0{,}5x^1 + 1{,}2x^2 \rightarrow min$

$x^1 >= 200$

$x^2 >= 100$

$25x^1 + 40x^1 >= 20000$

$2x^1 + 7x^2 <= 2500$

$x^1 >= 0,\ x^2 >= 0.$

Пример 4.32 - Решить следующую задачу линейного программирования:

$E = 20x^1 + 25x^2 + 17x^3 \rightarrow min.$

$4x^1 + 5x^2 + 2x^3 >= 400$

$x^1 + x^2 + 4x^3 >= 250$

$x^1 + x^2 + x^3 = 150$

$2x^1 + x^2 + x^3 <= 300$

$x^i >= 0, i = 1,...,3.$

4.12 Решение задач нелинейного программирования

Решение таких задач рассмотрим на следующем примере.

Пример 4.33 - Решить следующую задачу нелинейного программирования:

$$E = \frac{e^{x_1}}{x_2 \cdot e^{x_3}} \to min$$

$x^1 + x^2 + x^3 >= 50$

$3x^1 + 2x^2 <= 600$

$x^2 >= 10$

$x^1 = 2x^2$

$x^1x^2 + x^2x^3 <= 2000$

$x^1x^2 >= 100$

$x^{2^2}+x^{3^2} = 1000$

$x^i>=0,\ i=1,...,3.$

Чтобы решить задачу нелинейного программирования в Matlab, требуется привести ее к следующей стандартной форме:

- целевая функция должна подлежать минимизации;
- все линейные ограничения (как равенства, так и неравенства) приводятся к стандартной форме, как и для задач линейного программирования (см. подраздел 4.11);
- все нелинейные ограничения должны иметь в правой части ноль, причем все нелинейные ограничения-неравенства должны быть приведены к виду <0.

Приведем к стандартной форме решаемую задачу:

$$E = \frac{e^{x_1}}{x_2 \cdot e^{x_3}} \to min$$

$-x^1 - x^2 - x^3 <= -50$

$3x^1 + 2x^2 <= 600$

$-x^2<=-10$

$x^1 - 2x^2 = 0$

$x^1x^2 + x^2x^3 - 2000 <= 0$

$-x^1-x^2+ 100 <= 0$

$x^{2^2}+x^{3^2} - 1000 = 0$

$x^i>=0,\ i=1,...,3.$

Для решения задач нелинейного программирования в Matlab используется функция:

fmincon('ц_ф', x^0, a, b, a^r, b^r, x^{min}, x^{max}, 'н_о')

где 'ц_ф'- целевая функция или имя М-файла, реализующего эту функцию;

x^0 - массив-строка, задающий начальные значения переменных;

a - матрица коэффициентов линейных ограничений-неравенств (коэффициенты каждого ограничения указываются в отдельной строке матрицы);

b - массив-столбец правых частей линейных ограничений-неравенств;

a^r - матрица коэффициентов линейных ограничений-равенств (коэффициенты каждого ограничения указываются в отдельной строке матрицы);

b^r - массив-столбец правых частей линейных ограничений-равенств;

x^{min} - массив-столбец ограничений на минимальные значения переменных;

x^{max} - массив-столбец ограничений на максимальные значения переменных;

'н_о' - имя М-файла (функции), реализующего нелинейные ограничения задачи.

Создадим М-файл, реализующий целевую функцию решаемой задачи:

```
function f=cel_fun(x)
f=exp(x(l))/(x(2)*exp(x(3)));
```

Сохраним этот М-файл под именем **cel_fun.m.**

Требуется также создать М-файл, реализующий нелинейные ограничения. Этот М-файл должен представлять собой функцию, имеющую два выходных параметра: первый - массив нелинейных ограничений-неравенств, второй - массив нелинейных ограничений-равенств (именно в указанном порядке):

```
function [ner, rav]=nel_ogr(x)
ner=[x(1)*x(2)+x(2)*x(3)-2000;  -x(1)*x(2)+100];
rav=[x(2)^2+x(3)^2-1000];
```

Сохраним этот М-файл под именем **nel_ogr.m.**

Здесь **ner** - массив ограничений-неравенств, **rav** - массив ограничений-равенств.

Примечание - Если в задаче нелинейного программирования отсутствуют нелинейные ограничения какого-либо вида (равенства или неравенства), то вместо них указывается пустой массив. Например, если бы в данной задаче отсутствовали нелинейные ограничения-равенства, то М-файл, задающий нелинейные ограничения, имел бы следующий вид:

```
function [ner, rav]=nel_ogr(x)
ner=[x(1)*x(2)+x(2)*x(3)-2000; -x(1)*x(2)+100];
rav=[];
```

Для решения задачи нелинейного программирования требуется вызвать функцию **fmincon.** Как и для функции **linprog,** для нее требуется задавать много исходных данных. Поэтому для ее вызова также используем файл-сценарий:

```
x0=[1,1,1];
a=[-1, -1, -1; 3,2,0; 0,-1, 0];
b=[-50; 600; -10];
ar=[1,-2,0];
br=[0];
xmin=[0; 0; 0];
[x,e,k]=fmincon('cel_fun',x0,a,b,ar,br,xmin,[], 'nel_ogr')
```

Здесь в качестве начальных значений переменных задано $x^1 = x^2 = x^3 = 1$ (массив x0**).** Смысл выходных параметров функции **fmincon** (переменные **x, e** и **к)** тот же, что и при решении задачи линейного программирования.

Сохраним этот файл-сценарий, например, под именем **nlp.m.** Для решения задачи следует в командном окне ввести **nlp.**

Результат должен быть следующим: x^1=20, x^2=10, x^3=30, E=4,54x10^{-6}.

Примечание - Функцию fmincon можно использовать и для решения задач линейного программирования (см. подраздел 4.11). Однако делать это не рекомендуется.

Пример 4.34 - Решить задачу нелинейного программирования:

$E = x^1x^2+2x^1x^3+2x^2x^3 \rightarrow min$

$x^1x^2x^3 = 100$

$x^i>=0,\ i=1,\ldots,3.$

Пример 4.35 - Решить задачу нелинейного программирования:

$E=5x^1 - 0,2x^{1^2} + 2x^2 - 0,2x^{2^2} \rightarrow max$

$13x^1 + 6x^2<=90$
$8x^1 + 11x^2 <= 88$

$x^1 >= 0,\ x^2 >= 0.$

4.13 Решение дифференциальных уравнений

Для решения дифференциальных уравнений в Matlab имеется ряд специальных функций, называемых решателями. Они различаются по применяемым математическим методам, поэтому выбирать решатель следует в зависимости от вида дифференциального уравнения. Полная информация о решателях дифференциальных уравнений имеется в справочной системе Matlab.

Рассмотрим применение одного из этих решателей - **ode45.** Решатель **ode45** реализует метод Рунге-Кутта четвертого и пятого порядков и предназначен для решения систем обыкновенных дифференциальных уравнений следующего вида:

$y^{1}'=f^{1}(x,y^{1},y^{2},\ldots, y^{n})$
$y^{2}'=f^{2}(x,y^{1},y^{2},\ldots, y^{n})$

$y^{n}'=f^{n}(x,y^{1},y^{2},\ldots, y^{n})$

для заданного диапазона значений независимой переменной ***x*** и при заданных начальных условиях для каждой из функций y^{1},$y^{2},\ldots,y^{n}$.

Решатель **ode45** вызывается следующим образом:

[x,y]=ode45('ду',[x^{min}, x^{max}], y^{0});

где **'ду'** - имя М-файла (функции), реализующего правую часть системы дифференциальных уравнений;

x^{min}, x^{max} - границы диапазона значений независимой переменной;

y^{0} - массив начальных условий для функций y^{1}, $y^{2},\ldots,y^{n}$.

Выходные параметры решателя **ode45** имеют следующий смысл:

x - массив-столбец значений независимой переменной;

y - матрица из n столбцов, представляющих собой наборы значений переменных y^{1}, $y^{2},\ldots,$ y^{n}, полученные по результатам решения системы дифференциальных уравнений. Каждый столбец содержит набор значений одной переменной.

Пример **4.36** - Решить систему дифференциальных уравнений:

$y^{1} = y^{2}+0,Ix^{2}$

$y^{2}=-y^{1}$,

где ***x*** - независимая переменная, при следующих начальных условиях: $y^{1}(0)=0$, $y^{2}(0)=1$. Получить графики функций $y^{1}(x)$ и $y^{2}(x)$ для значений независимой переменной ***x*** от 0 до 20.

Создадим М-файл, реализующий правую часть системы дифференциальных уравнений:

```
function f=primer_du1(x,y)
f=[y(2)+0.1*x*x; -y(1)];
```

Сохраним этот файл под именем **primer_du1.m.**

Для решения системы дифференциальных уравнений будем использовать решатель **ode45.** Полученные результаты потребуется вывести на экран в виде графиков. Удобно объединить команды вызова решателя и построения графиков в одном файле-сценарии:

```
[x,y]=ode45(primer_dul',[0, 20], [0,1]);
plot(x,y(:,l),'b')
hold on
plot(x,y(:,2),'r')
grid on
legend('y1','y2')
```

Как указано выше, решатель **ode45** возвращает массив x (значения независимой переменной x в диапазоне от 0 до 20) и матрицу y, состоящую в данном случае из двух столбцов: это столбцы значений функций $y^1(x)$ и $y^2(x)$, соответствующие значениям x. Функция **plot(x,y(:,l),'b')** строит график по значениям массива x и первого столбца матрицы y, т.е. график $y^1(x)$. Цвет линии графика - синий (он задан опцией **'b'** функции **plot).** Аналогично функция **plot(x,y(:,2),'r')** строит график $y^2(x)$, используя линию красного цвета. Остальные команды, используемые здесь, рассмотрены в подразделе 4.5.

Сохраним этот файл-сценарий, например, под именем **resh_du1.m.** Для решения системы дифференциальных уравнений и вывода результатов требуется в командном окне ввести: **resh_du1.** Результаты будут иметь такой вид, как показано на рисунке 4.4.

Пример 4.37 - Решить дифференциальное уравнение:

$$y'' + 0{,}5y + y^3 = 10\sin x,$$

где *x* - независимая переменная, при начальных условиях y(0)=0, y'(0)=1. Построить график функции y(x) для значений *x* от 0 до 10.

Чтобы решить это уравнение, преобразуем его в систему дифференциальных уравнений, введя следующие обозначения: $y^1 = y$, $y^2 = y' = y^{1\prime}$. Система дифференциальных уравнений, эквивалентная решаемому уравнению, будет иметь следующий вид:

$$y^1 = y^2$$

$$y^2 = -0.5y^1 - y^{1\,3} + 10\sin x.$$

М-файл, реализующий правую часть этой системы:

```
function f=primer_du2(x,y)
f=[y(2);-0.5*y(l)-y(l)^3+10*sin(x)];
```

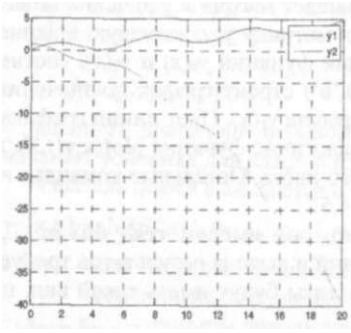
Сохраним этот файл под именем primer_du2.m.

Файл-сценарий для решения задачи:

```
[x,y]=ode45('primer_du2',[0,10],  [0,1]);
plot(x,y(:,l))
grid on
```

В данном случае матрица y, полученная в результате решения системы уравнений, будет состоять из двух столбцов: первый из них - набор значений функции $y^1(x)$, второй - $y^2(x)$. Однако, как видно из введенных в задаче обозначений, решением исходного уравнения является функция $y^1(x)=y(x)$; функция $y^2(x)$ представляет собой производную от нее. Поэтому требуется строить только один график (рисунок 4.5).

В качестве примера рассмотрим применение еще одного решателя дифференциальных уравнений - ode15s, применяемого для решения жестких систем дифференциальных уравнений.



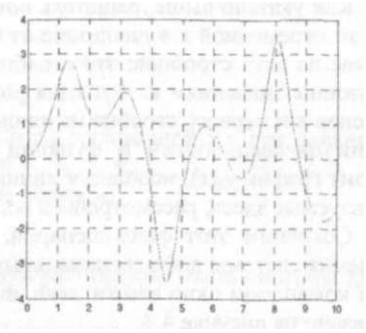


Рисунок 4.4 - Результаты примера 4.36 **Рисунок 4.5 - Результаты примера 4.37**

Пример 4.38 - Решить систему дифференциальных уравнений:

$$y^1 = y^2$$

$$y^2 = -y^1 + 1000(1 - y^{0^2})y^2.$$

Здесь y^1, y^2 - функции независимой переменной *x*. Требуется найти решение при начальных условиях $y^1(0)=2$, $y^2(0)=0$. Построить графики функций $y^1(x)$ и $y^2(x)$ для значений независимой переменной *x* от 0 до 3000. Использовать решатель ode15s.

М-файл, реализующий правую часть системы дифференциальных уравнений:

```
function f= primer_du3(x,y)
f=[y(2);-y(l)+1000*(l-y(l)^2)*y(2)];
```

Сценарий для решения задачи и вывода графиков:

```
[x,y]=odel5s('primer_du3',[0, 3000], [2,0]);
plot(x,y(:,1),'b')
grid on
hold on
plot(x,y(:,2),'r')
grid on
```

Как видно, входные и выходные параметры для решателя **odel5s** точно такие же, как и для **ode45.**

Результат будет выглядеть примерно так, как показано на рисунке 4.6. Видно, что отображается лишь один из графиков. Возможно, причина состоит в том, что диапазоны значений функций y^1 и y^2 значительно различаются. Чтобы получить оба графика, построим их в разных подокнах, используя рассмотренную в подразделе 4.5 функцию **subplot.** Результат должен иметь такой вид, как показано на рисунке **4.7.**

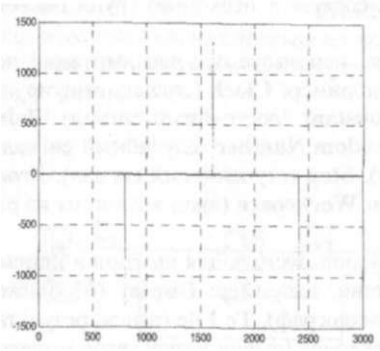


Рисунок 4.6 - Результаты примера 4.38 (вывод двух графиков в одном окне)

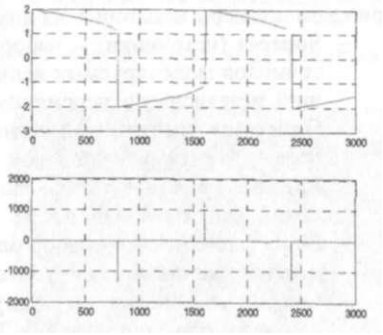


Рисунок 4.7 - Результаты примера 4.38 (вывод графиков в разные подокна)

Пример 4.39 - Решить дифференциальное уравнение:

$$y'' - y = xe^{x}$$

где x - независимая переменная, при следующих начальных условиях: y(0)=5, y'(0)=12. Построить график функции $y(x)$ для значений x от 0 до 10.

4.14 Система имитационного моделирования Simulink

4.14.1 Основные сведения о системе Simulink

Система Simulink предназначена для моделирования динамических систем, т.е. любых систем, состояние которых изменяется во времени. Система Simulink может применяться для моделирования самых разнообразных объектов и процессов: электрических схем, систем передачи и обработки сигналов, механизмов, тепловых процессов и т.д. Модель в системе Simulink строится в виде набора стандартных блоков, описывающих моделируемый объект или явление. На основе такой модели система Simulink автоматически строит описание объекта моделирования в виде систем дифференциальных уравнений, решает эти системы и отображает характеристики объекта моделирования.

Для начала работы с системой Simulink требуется в командном окне ввести команду simulink. На экран выводится окно библиотек Simulink (Simulink Library Browser).

В состав системы Simulink входят основная библиотека блоков (собственно Simulink) и ряд специализированных библиотек. Для удобства пользования библиотеки Simulink разбиты на группы и подгруппы блоков.

Набор библиотек Simulink и состав блоков, входящих в эти библиотеки, расширяются с появлением новых версий системы. Назовем некоторые из основных библиотек и групп блоков Simulink, имеющихся практически во всех версиях системы и применяемых в большинстве задач моделирования.

Основная библиотека блоков **Simulink** содержит несколько групп блоков. Приведем примеры некоторых из них:

- **Sources** (источники) - набор блоков, используемых для имитации источников моделируемых величин, например: **Clock** (сигнал, имитирующий независимую переменную), **Constant** (постоянный сигнал), **Pulse Generator** (импульсный сигнал), **Random Number** (случайный сигнал), **Sine Wave** (синусоидальный сигнал), **Step** (ступенчатый сигнал), **From File** (ввод величины из файла), **From Workspace** (ввод величины из рабочей области Matlab) и т.д.;
- **Sinks** (приемники) - набор блоков, используемых для имитации приема и отображения моделируемых величин, например: **Display** (отображение числовой величины), **Scope** (осциллограф), To **File** (вывод результатов моделирования в файл), To **Workspace** (вывод результатов моделирования в рабочую область Matlab) и т.д.;
- **Continuous** (непрерывные процессы): **Derivative** (производная), **Integrator** (интегрирование), **Delay** (задержка) и т.д.:
- **Math Operations** (математические операции): **Add** (суммирование и вычитание входных величин), **Divide** (деление и перемножение входных величин), **Gain** (умножение на число или матрицу), **Sum** (то же, что **Add), Product** (то же, что **Divide), Math Function** (набор математических функций) и т.д.

Имеется также большой набор специализированных библиотек, например, **Aerospace Blockset** (аэродинамические процессы), **Communications Blockset** (коммуникационные системы), **Neural Network Toolbox** (нейронные сети), **SimMechanics** (механизмы), **SimPowerSystems** (электротехника и электроника) и т.д.

Для создания модели требуется в окне библиотек **Simulink** выбрать команду **File - New - Model.** Создается пустое окно модели. Необходимые блоки перетаскиваются из библиотек в окно модели с помощью мыши.

Двойной щелчок мыши по любому из блоков, размещенных в окне модели, вызывает на экран окно параметров выбранного блока. Эти параметры могут представлять собой, например, амплитуду и частоту моделируемого сигнала, сопротивление резистора, формулу математического преобразования и т.д.

Результаты моделирования не только отображаются на экране (с помощью соответствующих блоков, рассматриваемых ниже), но и представляются в виде матриц, которые могут выводиться в рабочую область Matlab. Это позволяет выполнять их дальнейшую обработку, используя все средства Matlab.

Для сохранения файла модели используется команда **File - Save.** Файл сохраняется под указанным именем с расширением **MDL.**

4.14.2 Примеры моделирования в Simulink

Пример 4.40 - На вход некоторой системы поступает сигнал с амплитудой, представляющей собой случайную величину, равномерно распределенную

в диапазоне от 0 до 10. Требуется генерировать единичный сигнал каждый раз. когда входной сигнал превышает величину **5.**

Для данной задачи потребуется построить модель такого вида, как показано на рисунке 4.8.

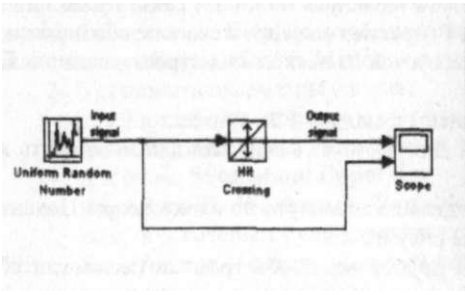


Рисунок 4.**8** - Модель для примера 4.40

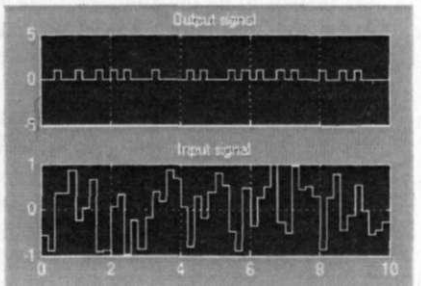


Рисунок 4.9 - Результаты примера **4.40**

Модель может быть построена следующим образом.

1 Для начала построения модели в окне **Simulink Library Browser** выбрать из меню команду **File - New - Model.** Появится пустое окно для построения модели.

2 С помощью мыши поместить в окно модели необходимые блоки. Блок **Uniform random number** (равномерная случайная величина) находится в группе **Sources,** блок **Hit crossing** (выработка единичного сигнала при заданном условии) - в группе **Discontinuities,** блок **Scope** (осциллограф) - в группе **Sinks.**

3 Установить параметры блока **Uniform random number** (т.е. параметры входного сигнала). Для ЭТОГО дважды щелкнуть мышью по блоку в окне модели. Установить следующие параметры: **Minimum** - 0, **Maximum** - 10. Нажать **OK.**

4 Аналогично установить параметры блока **Hit crossing: Hit crossing offset - 5** (выработка единичного сигнала, когда входной сигнал пересекает уровень **5). Hit crossing direction - rising** (выработка единичного сигнала, когда входной сигнал превышает **5** при нарастании).

5 Чтобы блок **Scope** отображал два сигнала сразу (как входной, так и единичный), установить для этого блока два входа. С этой целью дважды щелкнуть мышью по блоку **Scope.** На экран выводится окно, напоминающее экран осциллографа (пока оно пусто, так как моделирование еще не выполнялось). В наборе инструментов этого окна нажать на кнопку **Parameters** (при стандартной настройке - вторая слева в наборе кнопок). В появившемся окне в поле **Number of axes** указать значение **2.** Нажать **OK.** Закрыть окно блока **Scope.** У блока появляется второй вход.

6 Установить связи между блоками, как показано на рисунке 4.8. Для рисования линии связи между двумя блоками щелкнуть мышью *на выходе* одного из блоков и провести линию (не отпуская мышь) *до входа* подключаемого блока. Для рисования *ответвления* (на рисунке 4.8 - от линии входного сигнала ко второму входу блока **Scope)** держать нажатой клавишу **Ctrl.** Если линия рису-

ется *с изломами,* то в необходимых местах отпускать мышь, если требуется построить линию *с углами, отличными от прямых,* то во время рисования держать нажатой клавишу **Shift.**

7 На линиях связей нанести обозначения "Input signal" и "Output signal", как показано на рисунке 4.8. Для этого щелкнуть по линии связи правой кнопкой мыши, выбрать команду **Signal Properties** и ввести желаемое обозначение в поле **Signal Name.** Чтобы надпись состояла из нескольких строк, нажимать **Enter.**

8 Сохранить построенную модель командой **File - Save.**

9 Выполнить моделирование. Для этого из меню **Simulation** выбран, команду **Start.**

10 Для просмотра результатов дважды щелкнуть по блоку **Scope.** Появится окно, аналогичное приведенному на рисунке 4.9.

11 Выполнить автоматический подбор масштаба графика (автомасштабирование), щелкнув кнопку **Autoscale** (в виде бинокля).

12 Изменить масштаб оси Y для единичною сигнала, установив для этой оси диапазон от **-1** до **3.** С этой целью щелкнуть правой кнопкой мыши по этой оси. Из появившегося меню выбрать команду **Axes Properties.** В появившемся окне **Scope Properties** в поле **Y-min** указать значение **-1,** в поле **Y-max** - значение **3.** Нажать **OK.** Убедиться, что вид окна блока **Scope** изменился.

13 Скопировать модель в окно редактора Word. Для этого перейти в окно модели и выбрать команду **Edit - Copy Model to Clipboard.** Затем перейти в окно Word и выполнить вставку.

14 Скопировать полученные графики в окно Word. Для этого снова перейти в окно с графиками, построенными блоком **Scope.** Нажать клавиши **Alt - Print Screen,** затем перейти в окно Word и выполнить вставку.

15 Закрыть окно графика и окно модели.

Пример **4.41** - На вход некоторой системы поступает синусоидальный сигнал с амплитудой 10 и частотой 2 рад/с. Требуется ограничить этот сигнал на уровне от 0 до 3. Выполнить моделирование полученного сигнала в течение 20 с. Определить величину интеграла от полученного сигнала. Передать значение интеграла в рабочую область.

Для данной задачи потребуется модель такого вида, как показано на рисунке 4.10.

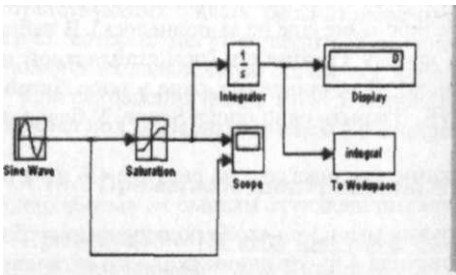


Рисунок 4.10 - Модель для примера 4.41

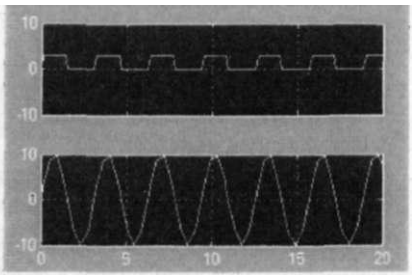


Рисунок 4.11 - Результаты примера 4.41

Порядок построения модели и получения результатов моделирования может быть следующим.

1 Командой File - New - Model создать новый файл модели. Поместить в окно модели необходимые блоки. Блок Sine Wave находится в группе Sources, блок Saturation - в группе Discontinuities, блок Integrator - в группе Continuous, блоки Scope. Display (отображение величины) и To Workspace (вывод величины в рабочую область Matlab) - в группе Sinks.

2 Установить параметры блоков:

- для блока Scope установить два входа (см. предыдущий пример);
- для блока Sine Wave: Amplitude - 10, Frequency - 2;
- для блока Saturation: Upper limit - 3, Lower limit - 0;
- для блока To Workspace: Variable name - integral (или любое другое имя. под которым будет создана переменная в рабочей области Matlab). Save format - Array (вывод данных в рабочую область Matlab в виде массива).

3 Установить связи между блоками, как показано на рисунке 4.10. Сохранить модель.

4 Указать время моделирования. Для этого из меню Simulation выбрать команду Configuration Parameters. В области Simulation Time в поле Start time установить значение 0, в поле Stop time - значение 20.

5 Выполнить моделирование (Simulation - Start). По окончании моделирования в блоке Display отображается значение интеграла.

6 Дважды щелкнув мышью по блоку Scope, перейти в окно графика результатов. Используя автомасштабирование и настройку осей (см. пример 4.40), добиться, чтобы окно блока Scope имело такой вид, как показано на рисунке 4.11.

7 Скопировать модель и график результатов моделирования в окно Word. Закрыть окно графика и модель.

8 Перейти в командное окно Matlab. Убедиться, что в рабочей области создан массив integral. Этот массив содержит значения интеграла от выходного сигнала в модели, вычислявшиеся по мере моделирования. Значение интеграла находится в *последнем* элементе массива. Следует по обозначениям в рабочей области определить размер массива и присвоить его последний элемент какой-либо переменной в командном окне.

Пример 4.42 - На вход некоторой системы поступает разность двух случайных величин, соответствующих нормальному закону распределения. Первая величина имеет среднее значение 15 и дисперсию 3, вторая - среднее значение 10 и дисперсию 2. На вход другой системы поступает постоянный сигнал, равный 5. Требуется выполнить интегрирование этих сигналов за 20 с и найти соотношение их интегралов.

Для решения этой задачи можно использовать модель, приведенную на рисунке 4.12.

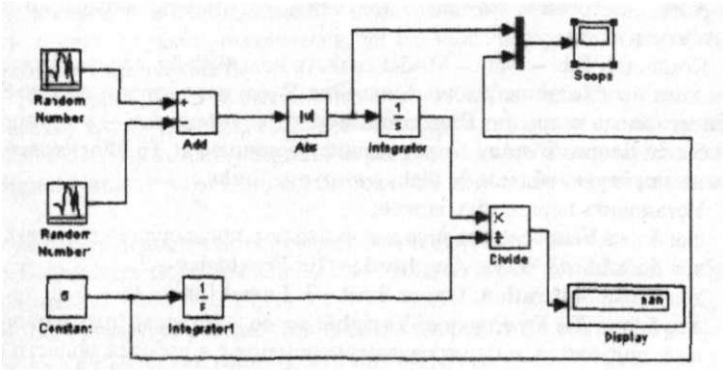


Рисунок **4**.12 - Модель для примера 4.42

Основные этапы решения задачи следующие.

1 Командой File - New - Model создать новый файл модели. Поместить в окно модели необходимые блоки. Блоки Random Number и Constant находятся в группе Sources, блоки Add. Abs (для получения разности сигналов без знака) и Divide - в группе Math Operations, блок Integrator - в группе Continuous, блоки Scope и Display - в группе Sinks, блок Mux (для подачи двух сигналов на вход блока Scope) - в группе Signal Routing.

Примечание - Блок Mux (мультиплексор) требуется для того, чтобы оба входных сигнала, поступающие на блок Scope, отображались в одной системе координат, а не на двух отдельных графиках.

2 Настроить блок Add (сумматор) так, чтобы он определял разность (а не сумму) входных величин. Для этого двойным щелчком мыши по блоку Add вызвать окно параметров блока. В этом окне в поле List of Signs (вкладка Main) установить обозначение +- (знаки "плюс" и "минус"); это значит, что вторая входная величина должна вычитаться из первой. Нажать OK.

3 Установить параметры остальных блоков:

- для блока Random Number: Mean - среднее значение. Variance - дисперсия:
- для блока Constant: Constant value - 5.

4 Установить связи между блоками, как показано па рисунке 4.12. Сохранить модель. Выполнить моделирование. Настроить изображение, полученное в блоке Scope, чтобы оно имело такой вид. как показано на рисунке 4.13. Перенести модель и результаты моделирования в Word.

Пример 4.43 - Построить схему электрической цепи, состоящей из источника постоянного напряжения (24 В) и резистора (100 Ом), а также включающей амперметр.

Модель для решения этой задачи приведена на рисунке 4.14.

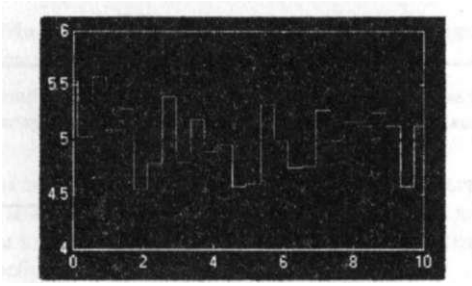


Рисунок **4.13** - Результаты примера 4.42

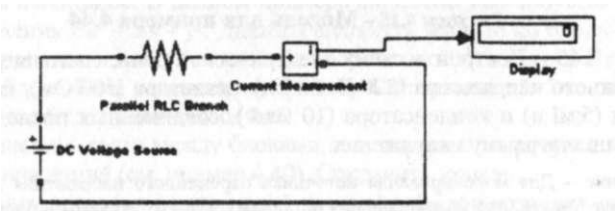


Рисунок **4.14** - Модель для примера 4.43

1 Командой File - New - Model создать новый файл модели.

2 Поместить в окно модели необходимые блоки. Для этого перейти в библиотеку SimPowerSystems. Из группы блоков Electrical Sources выбрать блок DC Voltage Source (источник постоянного напряжения), из группы Elements - блок Parallel RLC Branch (параллельное соединение резистора, конденсатора и катушки индуктивности), из группы Measurements - блок Current Measurement (измерение тока).

Примечание - Если при построении модели потребуется перевернуть или повернуть какой-либо блок, следует щелкнуть по этому блоку правой кнопкой мыши и выбрать из контекстного меню команду Format - Flip Block или Format - Rotate Block.

3 Перейти в основную библиотеку (Simulink). Из группы Sinks выбрать блок Display.

4 Вызвать окно параметров блока DC Voltage Source. В поле Amplitude установить 24 (напряжение источника). Нажать OK.

5 Исключить из схемы конденсатор и катушку индуктивности (т.е. оставить только резистор). Для этого настроить параметры блока Parallel RLC Branch следующим образом: Resistance - 100, Inductance - Inf (бесконечность), Capacitance - 0.

6 Установить связи между блоками, как показано на рисунке 4.14. Сохранить модель. Выполнить моделирование. В блоке Display отображается измеренная величина тока. Можно убедиться, что она соответствует закону Ома.

Пример 4.44 - Построить схему электрической цепи согласно рисунку 4.1**5.** Определить токи для всех резисторов.

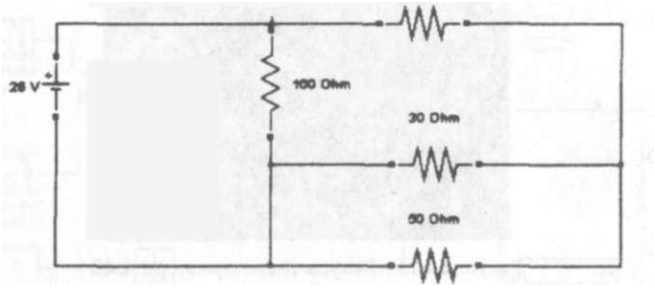


Рисунок 4.15 - **Модель для примера 4.44**

Пример 4.45 - Построить схему электрической цепи, состоящей из источника переменного напряжения **(220** В, 50 Гц), резистора **(100** Ом), катушки индуктивности (5 мГн) и конденсатора (10 мкФ), соединенных последовательно. Получить осциллограмму напряжения.

Примечание - Для моделирования источника переменного напряжения используется блок AC Voltage Source, для моделирования остальных элементов схемы - блок Series RLC Branch из библиотеки SimPowerSystems.

Пример 4.46 - Решить дифференциальное уравнение:

$$y'' = xe^{x} + 2y - y',$$

где x - независимая переменная. Построить график функции y(x) для значений x от **0** до 5.

Для данной задачи потребуется построить модель такого вида, как показано на рисунке 4.16.

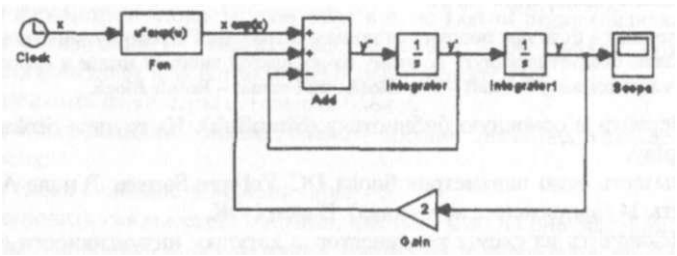


Рисунок 4.16 - Модель для примера 4.46

Порядок построения модели и получения результатов моделирования может быть следующим.

1 Командой File - New - Model создать новый файл модели. Поместить в окно модели необходимые блоки. Блок Clock (независимая переменная) находится в группе Sources, блок Fcn (произвольное математическое выражение) - в группе User-Defined Functions, блоки Add (сложение) и Gain (умножение на

число) - в группе Math Operations. Остальные блоки, используемые в модели, рассмотрены в предыдущих примерах.

Примечание - Чтобы повернуть блок Gain, как показано на рисунке 4.16, требуется щелкнуть по этому блоку правой кнопкой мыши и выбрать из контекстного меню команду Format - Flip Block.

2 В блоке Fcn задать выражение, которое требуется вычислять. Для этого дважды щелкнуть мышью по блоку. Вычисляемое выражение задается в поле Expression, причем входная величина блока Fcn обозначается как и. Поэтому в поле Expression требуется ввести: u*exp(u). Нажать OK.

3 Настроить блок Add для вычисления выражения, содержащего как сложение, так и вычитание. В данном примере блок Add используется для вычисления выражения $xe^x + 2y - y'$. Дважды щелкнуть мышью по блоку Fcn и в поле List of signs ввести: ++- (т.е. знаки "плюс", "плюс" и "минус"). Нажать OK.

4 Для блока Gain установить параметр Gain равным 2. Это означает, что входной сигнал будет умножаться на 2.

5 Установить связи между блоками, как показано на рисунке 4.16. Нанести обозначения линий (см. пример 4.40). Сохранить модель.

6 Указать интервал независимой переменной, для которого должно быть выполнено моделирование. Для этого из меню Simulation выбрать команду Configuration Parameters. В области Simulation Time в поле Start time установить значение 0, в поле Stop time - значение 5.

7 Выполнить моделирование. Результаты моделирования в блоке Scope должны иметь такой вид, как показано на рисунке 4.17.

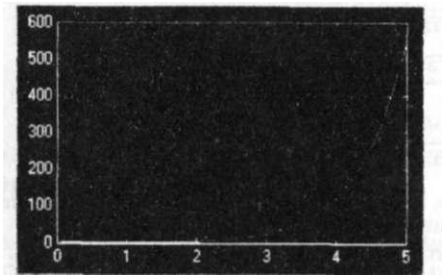


Рисунок 4.17 - Результаты примера **4**.32

Пример **4.47** - Решить дифференциальное уравнение:

$$y'' + 0{,}5y + y^3 = 10\sin x,$$

где x - независимая переменная. Построить график функции y(x) для значений x от 0 до 30.

Примечание - Для моделирования величины sin x можно использовать блок Sine Wave (группа блоков Sources).

ГЛАВА 5

СИСТЕМА КОМПЬЮТЕРНОЙ МАТЕМАТИКИ MATHEMATICA

5.1 Числовые и символьные операции в Mathematica

Прежде чем приступать к рассмотрению возможностей системы Mathematica, следует обратить внимание на ее важную особенность, отличающую ее от рассмотренной в главе 4 системы Matlab. Эта особенность - широкие возможности как для числовых, так и для символьных математических операций. Приведем простой пример, иллюстрирующий эти возможности.

Пусть требуется решить кубическое уравнение: $5x^3+3x^2-7x+15=0$. В системе Mathematica для этого можно воспользоваться следующей функцией:

```
Solve[5x^3 + 3 x^2 - 7 x + 15==0, x]
```

Результат будет следующим:

$$\left\{\left\{x\to \frac{1}{5}\left(-1-\frac{\left(2\left(927-5\sqrt{32727}\right)\right)^{1/3}}{3^{1/3}}-\frac{19\,2^{1/3}}{\left(3\left(927-5\sqrt{32727}\right)\right)^{1/3}}\right)\right\},\right.$$
$$\left\{x\to -\frac{1}{5}+\frac{\left(1+i\sqrt{3}\right)\left(927-5\sqrt{32727}\right)^{1/3}}{5\,6^{1/3}}+\frac{19\left(1-i\sqrt{3}\right)}{5\left(6\left(927-5\sqrt{32727}\right)\right)^{1/3}}\right\},$$
$$\left.\left\{x\to -\frac{1}{5}+\frac{\left(1-i\sqrt{3}\right)\left(927-5\sqrt{32727}\right)^{1/3}}{5\,6^{1/3}}+\frac{19\left(1+i\sqrt{3}\right)}{5\left(6\left(927-5\sqrt{32727}\right)\right)^{1/3}}\right\}\right\}$$

Таким образом, решение определяется в виде рациональных дробей, без каких-либо округлений.

Можно получить решение и в обычном числовом виде. Для этого следует ввести:

```
Solve[5x^3 + 3 x^2 - 7 x + 15==0, x]//N
```

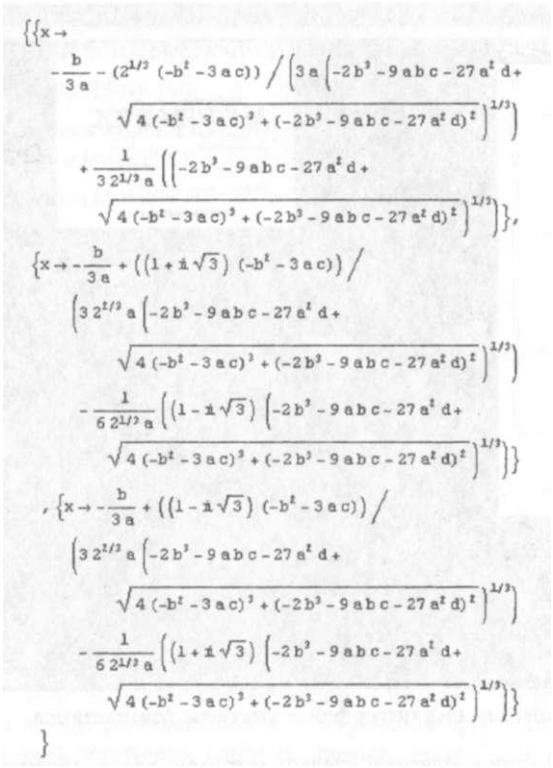
Результат будет следующим:

```
{{x -> -2.02401}, {x -> 0.712004 - 0.987551 i},
 {x -> 0.712004 + 0.987551 i}}
```

Если ввести

```
SoIve[a x^3 + b x^2 + c x + d==0, x]
```

то будет получен следующий результат:



т.е. решение задачи получено в общем виде.

5.2 Начало работы с Mathematica. Элементарные вычисления в Mathematica

В данном подразделе приводятся некоторые основные сведения, необходимые для начала работы с системой компьютерной математики Mathematica.

Стандартный вид интерфейса Mathematica показан на рисунке 5.1. Видно, что интерфейс системы Mathematica состоит из трех основных частей:

- меню;
- окно документа **(Notepad) -** область ввода команд и вывода получаемых результатов. В примере на рисунке 5.1 в окне документа показано решение кубического уравнения;
- палитры **(Palettes) -** наборы инструментов для вычислений и других операций. На рисунке 5.1 показана палитра начала работы **(Startup Palette),** из которой удобно вызывать справочную информацию.

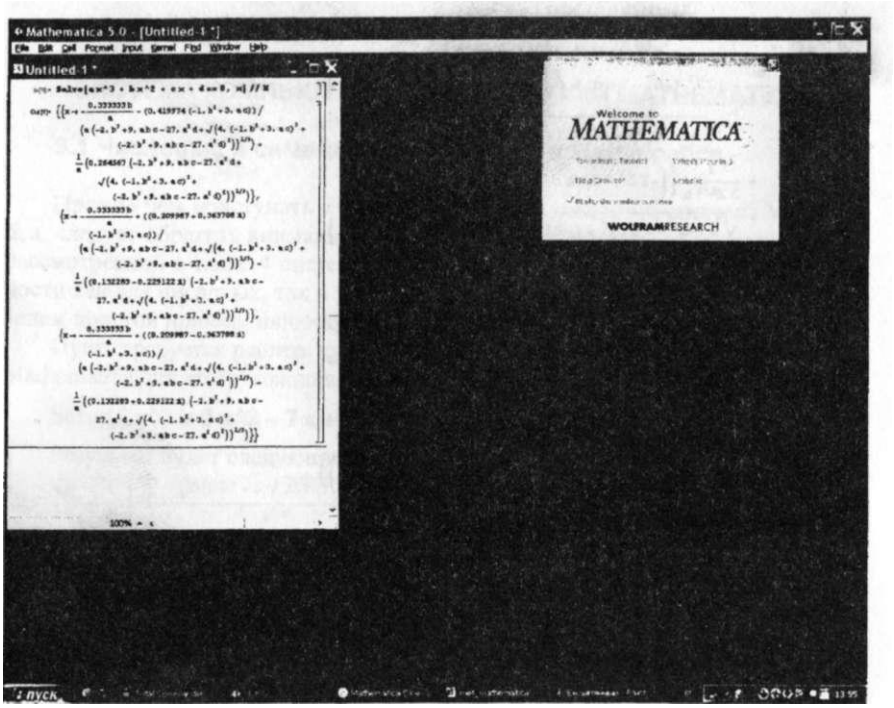


Рисунок 5.1 - Стандартный вид интерфейса системы Mathematica

Команды в окне документа вводятся в ячейках (строках) ввода, обозначаемых In[1], In[2] и т.д. Результаты выводятся в ячейках вывода: Out[1], Out[2] и т.д. Содержимое ячеек можно копировать, удалять, вносить в него изменения.

Для выполнения команд в Mathematica требуется нажимать *комбинацию клавиш* Shift - Enter.

Приведем примеры простейших команд, вводимых в окне документа. Например, если ввести:

5*8

то результат (число 40) выводится на экран, но не присваивается какой-либо переменной. Указывать знак умножения (*) необязательно, поэтому вместо 5*8 можно было ввести 5 8 (через пробел).

Если ввести:

a**=5*8**

то результат выводится на экран и присваивается переменной a.

Если команда завершается точкой с запятой, то ее результат не выводится на экран. Например, если ввести:

a=5*8;

то результат присваивается переменной a, но не выводится на экран.

Если, например, ввести:

(8-3)/(15-4)

то на экран выводится следующий результат: 5/11.

Если же ввести:

(8-3)/(15-4)//N

то результатом будет 0.4555.

Результат ввода выражения

x=2Pi

будет следующим: 2я, т.е. результат будет получен в символьном виде.

Результатом выражения

x=2Pi//N

будет число 6.28319.

Таким образом, система Mathematica может получать результаты как в числовом, так и в символьном виде. Несколько упрощая, можно считать, что в случае, если в вычисляемом выражении есть символы, результат выводится в символьном виде. Если это невозможно (например, в символьном виде задано слишком сложное уравнение, которое невозможно решить аналитически), то заданное выражение просто отображается на экране, тем самым указывая, что выполнить требуемую операцию не удалось.

Если вычисляемое выражение - числовое, то результат может быть получен в виде рациональной или десятичной дроби. Если в выражении есть хотя бы одна десятичная дробь (например, число 3.2, или 3.0, или даже 3.), то результат выводится в виде десятичной дроби, иначе - в виде рациональной дроби. Чтобы результат в любом случае выводился в виде десятичной дроби, требуется в конце выражения указать //N.

Пример 5.1 - Выполнить следующие элементарные расчеты.

1 Вычислить длину окружности радиусом 10 см. Для этого ввести: 2*Pi*10, и нажать Shift-Enter (обозначение Pi вводить с заглавной буквы). Результат выводится на экран: 20π.

2 Чтобы получить результат в виде десятичной дроби, ввести: 2*Pi*10//N.

Приведем еще несколько основных правил работы с системой Mathematica:

- имена переменных в Mathematica указываются со строчной буквы. Обозначения всех функций Mathematica (например, Sin), а также всех стандартных констант (например, Pi, E) и другие стандартные обозначения (например, Infinity - бесконечность) вводятся с ПРОПИСНОЙ буквы.
- аргументы функций указываются в квадратных скобках, например, Sin[x].

Так как система Mathematica предназначена как для числовых, так и для символьных операций, переменным в Mathematica можно присваивать не толь-

ко числовые значения, но и выражения. Рассмотрим эту возможность на следующем примере:

Пример 5.2 - Выполнить следующие расчеты (после каждой команды нажимать **Shift-Enter).**

1 Решить квадратное уравнение $5x^2+7x-25=0$. Для этого ввести:

Solve[5*x^2+7*x-25==0, x]

На экран выводятся корни уравнения.

Примечание - При вводе этого выражения важно, чтобы переменная x была при этом свободной, т.е. не имела никакого (в том числе нулевого) значения. Для очистки значения переменной (например, переменной x) используется команда x=. или Clear[x].

2 Присвоить квадратное уравнение $5x^2+7x-25=0$ переменной и. Для этого ввести:

и= 5*x^2+7*x-25==0

Важно обратить внимание, что значением переменной **и** является именно выражение **5*x^2+7*x-25==0** (а не корни этого уравнения, не ноль и не что-либо другое).

3 Снова получить решение квадратного уравнения:

Solve[u, x]

Легко убедиться, что получены те же результаты, что и при непосредственном вводе уравнения.

В ходе вычислений в системе Mathematica можно ссылаться на результаты предыдущих операций. Для ссылки на результат последней операции используется символ %, для предпоследней операции - % % и т.д.

Пример 5.3 - Выполнить следующие расчеты (после каждой команды нажимать **Shift-Enter):**

x=50*2
Sqrt[%]
%%+1

Результатом операции Sqrt[%] должно быть число 10, а результатом операции **%%+1** - число 101.

Результаты любой операции, выполненной в Mathematica, можно вывести в текстовый файл следующим образом: **операция >> "имя_файла".**

Пример 5.4 - Получить список значений функции и вывести его в файл.

1 Присвоить переменной **f** выражение $x^2\sin$ ***x:*** **f=x^2*Sin[x].** Переменная **x** при этом должна быть свободной, т.е. ей не должно быть присвоено какое-либо значение.

Примечание - Здесь, как и в примере 5.2, следует обратить внимание, что переменная **f** равна именно выражению **x^2*Sin[x],** а не какому-либо числу.

2 Получить список значений функции **f,** заданной ранее, для значений переменной x от 0 до 10 с шагом 0,1. Для этого ввести: **Table[f,{x,0,10,0.1}]//N.** Полученный список выводится на экран. Обозначение //N здесь использовано для того, чтобы все элементы списка были получены в числовом (а не в символьном) виде.

Примечание - Список в Mathematica аналогичен матрице или массиву в других программах аналогичного назначения. Подробнее понятие списка, а также назначение функции Table будет рассмотрено в подразделе **5.6.**

3 Получить такой же список еще раз и вывести его в текстовый файл. Для этого ввести: **Table**[f,{x**,0,10,0.1**}]//N>>**"имя_файла".** Например, чтобы вывести список в файл **rezlab1.txt,** расположенный в папке **c:_Users\Ivanov,** следует ввести:

Table[f,{x,0,10,0.1}]//N>>"c:_Users\\Ivanov\\rezlab1.txt"

4 *Не выходя из системы Mathematica,* убедиться, что файл со значениями функции создан.

Примечание - Вместо команды **Table[f**,{x**,0,10,0.1}]//N>>"имя_файла"** на шаге 3 можно было ввести следующую команду: **%»"имя_файла".** В файл был бы выведен результат предыдущей операции, в данном случае - тот же самый список.

5.3 Сохранение наборов команд. Восстановление результатов вычислений

Содержимое документа (т.е. набор введенных команд) сохраняется на жестком диске командой **File - Save.** Файл сохраняется с расширением **.nb.** Однако при этом сохраняется только текст введенных команд, *но не значения переменных.* Чтобы восстановить значения переменных, вычисленные ранее, необходимо после загрузки файла документа (команда **File - Open)** выделить ячейки с желаемыми формулами и выбрать команду **Kernel - Evaluation - Evaluate Cells.** Если требуется заново вычислить содержимое всех ячеек документа, используется команда **Kernel - Evaluation - Evaluate Notebook.**

Пример **5.5** - Выполнить следующие действия по сохранению и восстановлению результатов вычислений.

1 Сохранить документ, содержащий команды из примеров 5.**1** - 5.4, командой **File - Save.** Выйти из Mathematica.

2 Загрузить Mathematica. Загрузить документ командой **File - Open.**

3 В свободной строке (после загруженного текста документа) ввести букву **f** и нажать **Shift-Enter.** Убедиться, что в ответ выводится **f.** Это означает, что переменной **f** не присвоено никакого значения (хотя в загруженном документе есть ячейка с формулой **f=x^2*Sin[x]).**

4 Выделить ячейку с формулой **f=x^2*Sin[x].** Для этого щелкнуть мышью по обозначению ячейки (в виде квадратной скобки) в правой части окна документа. Выбрать команду **Kernel - Evaluation - Evaluate Cells.**

5 В конце документа снова ввести f. Убедиться, что в ответ выводится выражение x^2*Sin[x]. Таким образом, значение переменной f восстановлено.

6 Скопировать содержимое документа из Mathematica в Word. Чтобы выделить ячейки в Mathematica, провести мышью по обозначениям ячеек в правой части окна документа. После этого выполнить копирование и вставку в Word.

5.4 **Подстановки**

Понятие подстановки, очень широко применяемое в Mathematica, рассмотрим на следующем примере.

Пример 5.6 - Используя подстановку, вычислить значение функции с sin *x* при x=2.

1 Убедиться, что переменная f имеет значение f=x^2*Sin[x].

2 Ввести:

f/.x->2

Значение x=2 подставляется в выражение x^2*Sin[x], и вычисляется результат (он выводится на экран как 4 Sin[2]). Важно понимать, что значения переменных x и **f** при этом *не изменяются:* переменная x остается свободной, а **f** по-прежнему имеет значение x^2*Sin[x].

3 Ввести:

y=f/.x->2

Результат - тот же, что и на шаге 2, но теперь он присваивается переменной y. При этом переменные x и **f** не изменяются.

Подставлять можно не только числовые значения, но и выражения.

Пример 5.7 - В уравнении $5x^2+7x-25=0$ заменить переменную *x* на *z+1.*

1 Присвоить уравнение переменной и:

и= 5*x^2+7*x-25==0

2 Выполнить подстановку:

u=u/.x->z+1

На экран выводится результат: $-25+7(1+z)+5(1+z)^2==0$.

Операция присваивания (=) необходима, чтобы изменилось значение переменной и. Если ввести u/.x->z+1, то результат подстановки будет выведен на экран, но сама переменная и не изменится.

3 Для упрощения полученного выражения ввести:

u=Simplify[u]

На экран выводится результат: z (17+5 z)==13.

5.5 Вычисления с использованием палитры

Палитры представляют собой наборы инструментов для вычислений и других операций. Палитра выбирается с помощью команды **File - Palettes.**

Пример 5.8 - Вычислить интеграл $\int \frac{2x \sin x}{e^x} dx$ с помощью палитры и стандартных функций Mathematica.

1 Убедиться, что на экране имеется палитра со знаками интеграла Если ее нет, выбрать команду **File - Palettes - Basic Input.**

2 Щелчком мыши выбрать из палитры пиктограмму $\int \square d\square$.

3 Установить курсор в пустое поле под знаком интеграла (т.е. в первое из полей в полученном выражении). Выбрать из палитры пиктограмму $\frac{\square}{\square}$

4 Заполнить все элементы вычисляемого интеграла. Для ввода элемента со степенью (e^x) также использовать палитру. Окончательный вид выражения должен быть следующим:

$$\int \frac{2\,x\,\mathrm{Sin}[x]}{E^x}\,dx$$

Между умножаемыми величинами (в данном примере - между 2, **x** и **Sin)** должны быть пробелы или знаки *.

5 Для вычисления интеграла нажать **Shift-Enter.**

6 Вычислить этот же интеграл, используя одну из функций Mathematica. Для этого ввести:

```
Integrate[2x Sin[x]/Exp[x],x]
```

Здесь второй аргумент функции **(x)** указывает, что интегрирование выполняется по переменной *x*.

Убедиться, что результаты, полученные двумя способами, совпадают.

Пример 5.9 - Вычислить следующий интеграл:

$$\int_5^\infty \frac{1}{x \ln^3(x)} dx$$

двумя способами: с помощью палитры и с использованием функции численного интегрирования, имеющейся в Mathematica.

Функции, необходимые для решения задачи (для вычисления логарифма и численного интегрирования), найти самостоятельно, используя справочную систему Mathematica.

5.6 Списки и матрицы

5.6.1 Понятие списка. Способы задания списков

Список в Mathematica представляет собой совокупность произвольных данных. Матрица - частный случай списка.

Например, a={5, **3**.5, 7, 8, 4} - список из пяти элементов. Из этого примера видно, что элементы списка указываются в фигурных скобках и перечисляются через запятую.

Матрицы в Mathematica задаются как списки, элементами которых также являются списки - строки матрицы. Например, матрица

$$b=\begin{pmatrix}5 & 2 & 7\\ 4 & 6 & 2\\ 1 & 9 & 4\\ 3 & 7 & 2\end{pmatrix}$$

вводится в Mathematica следующим образом: b=**{**{5**,2**,7}, {4,6**,2},** {1,9,4}, **{3**,7**,2}}.** Здесь матрица **b** - список из четырех элементов, каждый из которых в свою очередь является списком из трех элементов.

Ссылка на конкретный элемент списка указывается в двойных квадратных скобках. Если список представляет собой двумерную матрицу, то первым указывается номер строки, затем - номер столбца.

Пусть, например, в Mathematica введены матрицы a и **b,** показанные выше. Если затем ввести a[[**2]],** то выводится **3**.5. Если ввести b[[**2]],** то выводится {4,6**,2},** т.е. второй элемент списка b**.** Если ввести **b[[3,2]],** то выводится 9.

Кроме непосредственного перечисления элементов, списки в Mathematica могут задаваться с помощью специальных функций.

Пример 5.10 - Создать список чисел от 1 до 10 с шагом 0.1, а также список значений функции $t^2+\sin t$ *для* значений переменной *t* от 1 до 10 с шагом 0,1.

1 Для получения первого из указанных списков ввести:

```
x=Range[l,10,0.1];
```

Точка с запятой в конце этой команды очень желательна, чтобы создаваемый список из 91 элемента не выводился на экран.

2 Для получения второго списка ввести:

```
y=Table[t^2+Sin[t],{t,l,10,0.1}]//N;
```

Список y можно получить и по-другому, используя уже созданный список **x,** например: **y=x^2+Sin[x]//N.**

Список не всегда является матрицей. Например, **x={5,{2,7,5}}** - список из двух элементов, первый из которых - число 5, а второй - список **{2,7,5}.**

5.6.2 Операции с элементами, строками и столбцами матриц

Операции с отдельными элементами матриц, а также их строками и столбцами (просмотр, добавление и т.д.) рассмотрим на следующем примере.

Пример 5.11 - Выполнить следующие операции с элементами, строками и столбцами матрицы, как показано выше:

$$a=\begin{pmatrix}1 & 2 & 3\\ 6 & 7 & 2\\ 9 & 2 & 8\end{pmatrix}$$

2 Присвоить переменной **elem** один из элементов матрицы **a,** например: **elem=a[[l,3]].**

3 Просмотреть вторую строку матрицы: **a[[2]].**

4 Просмотреть второй столбец матрицы **a: Transpose[a][[2]].** Здесь **Transpose [a]** - операция транспонирования матрицы.

5 Вставить в матрицу **a** строку (5,7,5) в качестве второй строки: **a=Insert[a,{5,7,5},2].**

6 Вставить в матрицу **a** два столбца (1,1,1,1) на второе и четвертое место. Для этого выполнить следующие действия:

- транспонировать матрицу: **a=Transpose[a];**
- чтобы вставить в транспонированную матрицу две строки (1,1,1,1), ввести: **a=Insert[a,{l,l,l,l},{{2},{3}}].** Здесь {{2},{3}} - список номеров строк, куда вставляются новые строки. Требуется указывать именно номера 2 и 3 (а не 2 и 4), так как номер вставляемой строки определяется по исходной матрице;
- снова транспонировать матрицу: **a=Transpose[a].**

7 Просмотреть матрицу **a** в матричной форме: **MatrixForm[a].**

8 Просмотреть матрицу **a** в табличной форме: **TableForm[a].**

5.6.3 Вычисления с матрицами

В Matlab имеются возможности для самых разнообразных операций с матрицами: от поэлементного сложения до наиболее сложных операций матричной алгебры. Рассмотрим эти операции на следующем примере.

Пример 5.12 - Выполнить следующие вычисления с матрицами.

1 Задать следующие матрицы, как показано выше:

$$a=\begin{pmatrix}3 & 2 & 8\\ 4 & 6 & 5\end{pmatrix},\quad b=\begin{pmatrix}4 & 1 & 7\\ 9 & 3 & 8\end{pmatrix},\quad c=\begin{pmatrix}4 & 8 & 5 & 7\\ 3 & 1 & 9 & 5\\ 5 & 9 & 6 & 1\end{pmatrix},\quad x=\begin{pmatrix}3 & 7 & 5\\ 4 & 1 & 9\\ 5 & 3 & 6\end{pmatrix}.$$

2 Увеличить все элементы матрицы **b** на единицу: **b=b+1.**

3 Сложить матрицы a и b, присвоив результат матрице u: u=a+b.

4 Перемножить матрицы a и b *поэлементно,* присвоив результат матрице v: v=a*b (или v=a b).

5 Извлечь квадратный корень из всех элементов матрицы b: Sqrt[b].

6 Перемножить матрицы **a** и **c** *правшам матричной алгебры:* z=a.c.

7 Транспонировать матрицу b, т.е. поменять местами строки и столбцы: b=Transpose[b].

8 Вычислить определитель матрицы x: d=Det[x].

9 Вычислить матрицу, обратную к x: xobr=Inverse[x].

10 Чтобы убедиться, что обратная матрица найдена правильно, перемножить матрицы x и xobr по правилам матричной алгебры, как показано выше. Результатом должна быть единичная матрица, т.е. матрица, в которой диагональ состоит из единиц, остальные элементы - нули.

11 Вычислить собственные значения матрицы x: Eigenvalues(x).

12 Вычислить собственные векторы матрицы x: Eigenvectors(x).

Пример 5.13 - Решить систему линейных уравнений:

$6x^1 + 14x^2 + 7x^3 = 120$

$3x^1 + 5x^2 + 9x^3 = 175$

$8x^1 + 3x^2 + 5x^3 = 100.$

Для решения системы уравнений вычислить столбец значений переменных x следующим образом: $x=a^{-1}b$, где a - матрица коэффициентов уравнений (a^{-1} - обратная матрица), b - столбец правых частей уравнений. Для проверки полученного решения перемножить матрицы a и x; должны быть получены правые части уравнений.

Решить систему линейных уравнений также с помощью функции Mathematica LinearSolve[a,b].

5.7 Пакеты расширения

Пакеты расширения (Add-on) представляют собой наборы функций Mathematica, загружаемые в память не автоматически (при загрузке самой системы Mathematica), а по указанию пользователя. В пакеты расширения включаются функции, относящиеся к определенному разделу математики (например, линейная алгебра, теория чисел и т.д.) или к определенному классу задач (например, построение графиков). Пакеты расширения разбиваются на подпакеты. Полная информация о пакетах и подпакетах расширения имеется в справочной системе Mathematica. Пакеты расширения загружаются следующей командой:

«имя_пакета'имя_подпакета'

Следует обратить внимание, что кавычки, используемые в этой команде, обычно указываются на клавише вместе с символом "~" (а также русской буквой "Ё").

Пример 5.14 - Загрузить подпакет расширения для работы с физическими константами.

Пусть решается некоторая задача, в которой часто требуется использовать физические константы (например, скорость света, массу электрона и т.д.). Необходимый подпакет расширения загружается следующей командой:

«Miscellaneous'PhysicalConstants'

Эта команда загружает подпакет PhysicalConstants пакета расширения Miscellaneous. Если теперь ввести, например:

SpeedOfLight

то на экран выводится следующее:

```
299792459 Meter
     Second
```

Другие примеры применения пакетов расширения (например, для построения графиков) будут рассмотрены в последующих разделах.

5.8 Построение графиков

Система Mathematica содержит большой набор функций для построения графиков самого разнообразного вида. Некоторые из них входят в основной набор функций Mathematica и доступны пользователю по умолчанию, другие - входят в состав пакетов расширения.

Основная функция Mathematica для построения графиков средней сложности - Plot[функция, {переменная, начало, конец}, опции]. Здесь функция - функция, график которой требуется построить (например, Sin[x]). Переменная - имя переменной, соответствующей оси абсцисс графика. Начало и конец - границы диапазона значений переменной, в котором строится график. Опции задают внешний вид графика (типы и цвета линий, подписи осей и т.д.). Для использования некоторых опций функции Plot необходимо загружать пакеты расширений.

Рассмотрим некоторые примеры построения графиков в Mathematica.

Пример 5.15 - Построить график функции y=0,25x+sin *x* **-1** для значений *x* от 0 до 10.

Для этого потребуется ввести:

```
Plot[0.25 x+Sin[x]-l,{x,0,10}]
```

Пример 5.16 - Построить графики функций y=0,25x+sin x - 1 и y=0,25x+cos *x* **- 1** для значений *x* от 0 до 10.

Для этого потребуется ввести:

```
Plot[{0.25 x + Sin[x] - 1,0.25x + Cos[x] - 1}, {x, 0,10},
   AxesLabel -> {"X", "Y"},
   PlotStyle -> {Dashing[{}], Dashing[{0.05,0.025}]}]
```

Примечание - При наборе длинной команды, не помещающейся в одну строку, переход в новую строку будет выполняться автоматически.

Здесь опция AxesLabel задает подписи осей. Опция PlotStyle с директивой Dashing задает вид графиков: первый из выводимых графиков выводится сплошной линией, второй - штриховой (здесь 0,05 - длина штриха, 0,025 - длина интервала между штрихами). Подробная информация обо всех опциях и директивах функции Plot имеется в справочной системе Mathematica.

Пример 5.17 - Построить графики функций y=0,25x+sin *x* - 1 и y=0,25x+cos *x* - 1 для значений *x* от 0 до 10 с выводом легенды.

1 Загрузить подпакет Legend пакета расширения Graphics:

```
<<Graphics'Legend'
```

2 Для построения графиков ввести:

```
Plot[{0.25x + Sin[x] - 1, 0.25x+Cos[x] - 1}, {x, 0, 10},
  AxesLabel -> {"X", "Y"},
  PlotStyle-> {Dashing[{0.15, 0.025}],
    Dashing[{0.05, 0.025}]},
  PlotLegend -> {"0.25 x+Sin[x]-1", "0.25 x+Cos[x] -1"} ,
  LegendPosition -> {-1, -1.2}, IegendSize -> {2, 0.5}]
```

Примечание - Опция LegendPosition задает расположение нижнего левого угла легенды относительно центра графика. Первой указывается позиция по горизонтали, второй - по вертикали. Сторона графика, имеющая максимальную длину, считается расположенной в диапазоне от -1 до 1. В такой же системе координат задается размер легенды (LegendSize).

Пример 5.18 - Построить график функции, заданной параметрически:

$x=6\cos^3 t$
$y=6\sin^3 t$

для значений t от 0 до 2π. Для этого ввести:

```
ParametricPlot[{6*Cos[t]^3,6*Sin[t]^3},{t,0,2*Pi}]
```

Пример 5.19 - Построить график функции, заданной в полярных координатах: r=6 cos *3t,* для значений f от 0 до 2π.

1 Загрузить подпакет Graphics пакета расширения Graphics:

```
<<Graphics'Graphics'
```

2 Чтобы построить график, ввести:

```
PolarPlot[6 Cos[3 t],{t, 0,2Pi}]
```

Пример 5.20 - Построить трехмерный график: график функции $z=x^2+2y^2$ для значений *x* и *y* от -5 до 5. Для этого ввести:

```
Plot3D[x^2+2 y^2,{x,-5,5},{y,-5,5}]
```

Построить график той же функции, срезанный на уровне z=40:

```
Plot3D[x^2+2y^2,{x,-5,5},{y,-5,5},PlotRange->{0,40}]
```

Построить тот же график без линий каркаса:

```
Plot3D[x^2+2y^2,{x,-5,5},{y,-5,5},PlotRange->{0,40},Mesh->False]
```

Построить контурный график той же функции:

```
ContourPlot[x^2+2y^2,{x,-5,5},{y,-5,5},Axes->True,AxesLabel->{"X","Y"}]
```

Пример 5.21 - Построить график функции, заданной параметрически:

$x=(\cos t)(3+\cos u)$
$y=(\sin t)(3+\cos u)$
$z=\sin 2u$

для значений t от 0 до 1,75π, u от 0 до 2π. Для этого ввести:

```
ParametricPlot3D[{Cos[t] (3+Cos[u]), Sin(t](3 + Cos[u]),
  Sin[2u]} , {t, 0, 1.75Pi}, {u, 0, 2Pi}, Axes -> True,
 AxesLabel -> {"X", "Y", "Z"}]
```

Пример 5.22 - Пусть в результате некоторого эксперимента получены следующие значения двух величин:

X	1,2	3,1	6,7	7,8	9,8	12,5	14,3
Y	15,7	12,2	10,4	6,2	6,1	5,9	5,8

Эти данные сохранены в текстовом файле. Требуется построить по этим данным точечный график, отражающий связь величин *X* и *Y*.

1 Используя Блокнот, создать текстовый файл, содержащий два столбца чисел:

```
1.2 15.7
3.1 12.2
6.7 10.4
7.8 6.2
9.8  6.1
12.5  5.9
14.3  5.8
```

2 Сохранить файл исходных данных и закрыть его.

3 Вернуться в систему Mathematica. Ввести данные из файла в какую-либо переменную (в этом примере - переменная dan). Для этого ввести:

```
dan=ReadList["имя_файла", Number, RecordLists->True]
```

Имя файла указывается, как показано в примере 5.4. Директива Number необходима для того, чтобы данные из файла вводились как отдельные числа; если не указать Number, то числа, указанные в одной строке, будут перемножаться. Директива RecordLists->True необходима для того, чтобы данные вводились в виде двумерной матрицы (если не указать эту директиву, то все числа из файла будут введены в виде одномерной матрицы). В результате переменная dan будет представлять собой матрицу из двух столбцов.

4 Чтобы построить график, ввести: ListPlot[dan].

5 Чтобы график имел лучший внешний вид (например, чтобы увеличить точки графика), ввести:

```
ListPlot[dan, PlotRange->{-l,15}, PlotStyle->PointSize[0.02]]
```

Пример 5.23 - Построить график следующей функции, заданной неявно: $(x^2+y^2)^2-x^2+y^2=0$, для значений *x* от -2 до 2. Необходимый для этого пакет расширения, подпакет и функцию для построения графика найти самостоятельно, используя справочную систему Mathematica.

Примечание - "Неявная функция" на английском языке - implicit function.

Пример 5.24 - Загрузить файл из примера 5.22. Используя операции со списками (см. подраздел 5.6), получить два списка: списки значений *X* и *Y*.

5.9 Решение алгебраических уравнений

Для аналитического решения алгебраических уравнений в Mathematica применяются следующие функции:

Solve[уравнение, переменная]

или

NSolve[уравнение, переменная]

где уравнение - решаемое уравнение;
переменная - переменная, относительно которой решается уравнение.

Функция Solve определяет решение уравнения в символьной форме или в виде рациональных дробей, функция NSolve - в числовой форме (в виде десятичных дробей).

Примечание - Более подробно о том, как определяется форма получаемого результата, см. в подразделах 5.1 и 5.2.

Если уравнение имеет несколько решений, то функции Solve и NSolve определяют *все* решения. Если получить список всех решений невозможно (например, их количество бесконечно из-за периодичности функций, входящих в уравнение), то выводится предупреждающее сообщение.

Если аналитическое решение уравнения невозможно, то функция **Solve** или NSolve выводит на экран уравнение без решения. В этом случае необходимо решать уравнение численными методами. Для этого применяется следующая функция:

FindRoot[уравнение, {переменная, начальная_точка}]

где уравнение - решаемое уравнение;
переменная - переменная, относительно которой решается уравнение;
начальная_точка - значение переменной, в окрестности которого ищется решение.

Если уравнение имеет несколько решений, то функция FindRoot находит *лишь одно* из них (ближайшее к заданной начальной точке). Другие решения требуется определять, изменяя начальную точку.

Рассмотрим применение этих функций на примерах.

Пример 5.25 - Решить уравнение $\sin x + \cos x = 0$.

1 Для решения уравнения ввести:

```
Solve[Sin[x]+Cos[x]==0,x]
```

На экран выводится следующий результат:

$$\left\{\left\{x \to -\frac{\pi}{4}\right\}, \left\{x \to \frac{3\pi}{4}\right\}\right\}$$

Выводится также предупреждающее сообщение о том, что все решения не могут быть найдены.

Как видно, результат получен в виде списка подстановок.

2 Для проверки результата подставить его в исходное уравнение:

```
Sin[x]+Cos[x]==0/.%
```

Должен быть получен ответ: {true,true}. Это означает, что при подстановке обоих найденных решений исходное уравнение является верным равенством.

Примечание - В данной операции использована подстановка (/.) и ссылка на результат последней операции (%). Подробнее об этом см. в подразделах 5.**2** и 5.4.

3 Выполнить действия, рассмотренные на шагах 1 и 2, другим способом: присвоив строку решаемого уравнения некоторой переменной (см. подраздел 5.2). Для этого ввести (после каждой команды нажимать Shift-Enter):

```
ur=Sin[x]+Cos[x]==0
Solve[ur,x]
ur/.%
```

Здесь строка уравнения (Sin[x]+Cos[x]==0) присваивается переменной ur, что позволяет сократить последующие выражения. Результаты должны быть такими же, как на шагах 1 и 2.

4 Получить решение уравнения $\sin x + \cos x = 0$ в виде десятичных дробей. Для этого ввести:

```
NSolve[ur,x]
```

Результатом является следующий список подстановок:

```
{{x -> -0.785398}, {x -> 2.35619}}
```

5 Проверить полученный результат. Для этого подставить найденные решения *в левую часть* уравнения:

```
Sin[x]+Cos[x]/.%
```

Результаты должны быть примерно следующими: $\{1.11022 \times 10^{-16}, 1.11022 \times 10^{-16}\}$, т.е. очень близкими к правой части уравнения (нулю).

Примечание - Попытка выполнить подстановку ur/.% (как это было сделано для решения, найденного в виде рациональных дробей) могла бы привести к неожиданному результату: {false, false}, хотя уравнение решено правильно. Причина в том, что решение, полученное в виде десятичных дробей - приближенное.

6 Получить решение уравнения $\sin x + \cos x = 0$ в виде списка чисел (а не подстановок, как во всех предыдущих случаях). Для этого ввести:

```
rez=x/.NSoIve[ur,x]
```

Здесь результаты, возвращенные функцией NSolve (т.е. подстановки, показанные на шаге 4), подставляются в переменную x, которая присваивается переменной rez. Переменная rez будет представлять собой список {-0.785398, 2.35619}.

Примечания

1 Если ввести команду x/.NSolve[ur,x], то результаты функции NSolve будут подставлены в переменную x и выведены на экран в виде чисел, но не сохранятся в какой-либо переменной.

2 В команде rez=x/.NSolve[ur,x] необходимо использовать именно переменную x (а не переменную с каким-либо другим именем), так как в уравнении, задаваемом переменной **иг,** используется переменная **x** (см. шаг 3).

7 Для проверки полученных результатов (т.е. решений в виде чисел) выполнить следующую подстановку:

```
Sin[x]+Cos[x]/.x->rez
```

Должны быть получены значения, очень близкие к нулю.

Пример 5.26 - Найти все решения уравнения $0{,}25x + \sin x - 1 = 0$ для значений x от 0 до 10.

1 Попытаться найти решения уравнения с помощью функции Solve. На экран выводится следующее:

```
Solve[-1+0.25 x+Sin[x]==0,x]
```

Это означает, что уравнение не может быть решено аналитически. Необходимо искать каждое из решений уравнения численными методами.

2 Чтобы примерно определить, где находятся решения уравнения, построить график функции $y=0{,}25x+\sin x - 1$ для значений x от 0 до 10, используя функцию Plot. График будет иметь такой вид, как показано на рисунке 5.2.

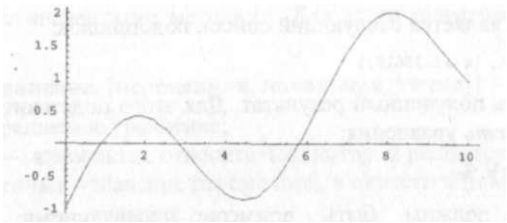


Рисунок S.2 - График функции $y=0{,}25x + \sin x$ **-1**

3 Для решения уравнения будет использоваться функция FindRoot (см. выше). Например, чтобы найти первое из решений, можно ввести:

FindRoot[0.25*x+Sin[x]-l=0,{x,0}]

Это означает, что поиск решения будет выполняться в окрестностях точки x=0. Результат будет следующим: {x->0.890487}.

4 Для проверки полученного результата ввести:

0.25*x+Sin[x]-l/.%

Это означает, что вместо x в формулу подставляется число 0,890487. Результат должен быть очень близким к нулю.

5 Получить решение уравнения в виде числа (а не в виде подстановки). Для этого ввести:

xl=x/.FindRoot[0.25*x+Sin[x]-l==0,{x,0}]

Здесь решение уравнения (число 0,890487) подставляется в переменную x, которая присваивается переменной x1.

6 Для проверки ввести:

0.25*x+Sin[x]-l/.x->xl

Результат должен быть близок к нулю.

7 Аналогично найти остальные решения уравнения в виде числовых переменных x2 и x3.

8 Получить список найденных решений. Для этого ввести: x={xl,x2,x3}.

5.10 Решение систем алгебраических уравнений

Для аналитического решения систем алгебраических уравнений используются функции Solve и NSolve, для численного - функция FindRoot. Однако системы алгебраических уравнений, для которых возможно аналитическое решение, встречаются в практических задачах достаточно редко. Поэтому в данном пособии рассматривается только численное решение систем алгебраических уравнений. Для этого применяется функция FindRoot в следующей форме:

FindRoot**[{уравнения},** {x,x^0},{y,y^0},...]

где уравнения - уравнения, входящие в решаемую систему;

x, y,... - переменные, относительно которых решается система уравнений;

x^0, y^0,... - начальные значения искомых переменных.

Пример 5.27 - Решить следующую систему уравнений:

$$x1^2 - x1x2 = e^{-x1}$$

$$x2^2 - x1x2 = e^{-x2}.$$

1 Очистить переменные x1, **x2,** x3 и x, так как они могут иметь значения, оставшиеся от предыдущих задач. Например, для очистки переменной xl следует ввести:

xl=.

или

Clear[x1]

2 Чтобы решить систему уравнений, ввести:

FindRoot[{xl^**2**-x1 **x2**=Exp[-x1], **x2^2+x1 x2**==Exp**[-x2**]},{xl**,0},{x2,0}]**

В данном случае в качестве начальных значений переменных xl и **x2** заданы нули.

Будет получено следующее решение:

{x1->0.906592, x2->0.461078}.

3 Для проверки решения подставить полученные результаты в решенную систему уравнений. Это можно выполнить, например, следующим образом:

{x1^2-x1 x2-Exp[-xl], **x2^2+x1 x2**-Exp**[-x2]}/.%**

В результате должен быть получен список из двух чисел (так как решались два уравнения), очень близких к нулю.

4 Найти решение системы уравнений: в виде списка чисел (а не списка подстановок, как это сделано на шаге 2). Для этого ввести:

x={xl**,x2**}/.FindRoot[{x1^**2**-xl **x2**=Exp[-xl], **x2^2+x1 x2**==Exp**[-x2]},** {xl**,0},{x2,0}]**

Здесь результаты, возвращаемые функцией FindRoot (xl->0.906592, x2->0.461078), подставляются в переменные xl и **x2.** Результат подстановки (список из двух чисел) присваивается переменной x.

5.11 Поиск экстремумов функций одной переменной

Для поиска минимумов функций одной переменной *y=f(x)* используется следующая функция:

FindMiniinum[функция, {переменная, начальная_точка}]

где функция - функция *fix), для* которой требуется найти минимум;

переменная - искомая переменная;

начальная_точка - значение переменной, в окрестности которого ищется экстремум.

Для поиска максимумов используется функция FindMaximum, имеющая точно такой же формат, что и FindMinimum.

Если функция имеет несколько экстремумов, то функции FindMinimum и FindMaximum находят лишь один из них. Другие экстремумы требуется определять, изменяя начальную точку.

Пример 5.28 - Найти все точки экстремума функции y=0.25 x+sin *x* - 1 на отрезке [0; 10].

Предварительно желательно определить приближенное расположение экстремумов, используя график функции (см. рисунок 5.2).

1 Чтобы определить точку минимума (из графика видно, что она находится вблизи точки x=5), ввести:

```
FindMinimum[0.25*x+Sin[x]-1,{x,5}]
```

Результат будет следующим: {-0.853319,{x->4.45971}}. Здесь первый элемент списка (число -0.853319) - значение функции в точке экстремума; второй элемент (подстановка x->4.45971) - искомая точка экстремума.

2 Получить решение в виде двух числовых переменных *(x* и y), выделив их значения решения, полученного в виде списка. Для этого ввести (нажимая Shift-Enter после каждой команды):

```
rez=%
y=rez[[1]]
x=x/.rez[[2]]
```

Первая из этих команд присваивает последний полученный результат (т.е. список, найденный на шаге 1) переменной rez. Таким образом, переменная rez получает значение списка {-0.853319,{x->4.45971}}. Вторая команда выделяет из этого списка первый элемент (число) и присваивает его переменной y. Третья команда выделяет из списка второй элемент (подстановку), подставляет его в переменную x и присваивает переменной x.

3 Найти остальные точки экстремума (см. рисунок 5.2), используя функцию FindMaximum.

5.12 Поиск экстремумов функций нескольких переменных

Для поиска минимумов функций нескольких переменных используется функция FindMinimum в следующей форме:

FindMinimum[функция, {x,x^0},{y,y^0},...]

где функция - функция нескольких переменных, для которой требуется найти минимум;

x, y,... - искомые переменные;

x^0, y^0,... - начальные значения искомых переменных.

Для поиска максимумов используется функция FindMaximum, имеющая точно такой же формат, что и FindMinimum.

Пример 5.29 - Найти точку минимума следующей функции двух переменных:$f(x^1, x^2) = (x^1-5)^2(x^2-2)^2 - (x^1-5)(x^2-2)$.

1 Ввести:

```
FindMinimum[(xl-5)^2*(x2-2)^2-(xl-5)*(x2-2),{xl,0},{x2,0}]
```

Результат будет получен в виде следующего списка:

{-0.25, {x**1** -> 1.09656, x2-> 1.87191}}.

Здесь первый элемент списка (число -0.25) - значение функции в точке экстремума; второй элемент - искомая точка экстремума (в виде списка подстановок).

2 Убедиться, что значение функции в точке экстремума вычислено верно. Для этого подставить найденные значения xl и x2 в функцию:

```
rez=%
(x1-5)^2*(x2-2)^2-(x1-5)*(x2-2)/.rez[[2]]
```

Первая из этих команд присваивает последний полученный результат (т.е. список, найденный на шаге 1) переменной rez. Таким образом, переменная rez получает значение списка {-0.25,{x1->1.09656, x2->1.87191}}. Вторая команда выделяет из этого списка второй элемент rez[[2]], т.е. список подстановок для переменных xl и x2, и подставляет эти переменные в заданную функцию. Результатом выполнения второй команды должно быть число 0,25.

3 Получить решение в виде числовых переменных (f, xl и x2), выделив их значения из списка, полученного на шаге 1 в результате выполнения функции FindRoot. Для этого ввести (нажимая Shift-Enter после каждой команды):

```
f=rez[[1]]
x1=x1/.rez[[2]]
x2=x2/.rez[[2]]
```

Здесь команда f=rez[[1]] просто присваивает переменной f первый элемент списка rez. Этот первый элемент - число (значение функции в точке экстремума). Две последующие команды позволяют получить обычные числовые переменные x1 и x2, используя для этого подстановку.

Пример 5.30 - Найти все точки экстремума функции $y = 5 - 2/x - x^2 + 10 \sin x$ и все решения уравнения $5 - 2/x - x^2 + 10 \sin x = 0$ для значений x от -10 до 10.

5.13 Решение задач линейного и нелинейного программирования

Для решения задач линейного и нелинейного программирования (см. подраздел 3.5) с целевой функцией, подлежащей максимизации, в Mathematica применяется следующая функция:

Maximize[целевая функция, {ограничения}, {переменные}]

Для решения задач с целевой функцией, подлежащей минимизации, применяется функция Minimize, имеющая точно такой же формат, что и Maximize.

Чтобы результат всегда вычислялся в виде десятичных дробей, вместо функций Minimize и Maximize могут использоваться функции NMinimize и NMaximize.

Пример 5.31 - Решить задачу нелинейного программирования:

$E=3x^1 + 5x^2 + 2x^3 \rightarrow max$

$8x^1 + 7x^2 + 5x^3 <= 1000$

$2x^1 + 5x^3 <= 200$

$x^1 + x^2 + x^3 = 100$

$x^1 + x^2 >= 20$

$x^1x^2 + 2x^1x^3 + x^2x^3 >= 2500$

$x^i>=0$, i=1,...,3.

Для решения этой задачи требуется ввести:

```
Maximze[3x1 + 5x2 + 2x3,
{8x1 + 7x2 + 5x3 <= 1000, 2x1 + 5x3 <=200,
x1 + x2 + x3 ==100, x1 + x2 >= 20,
xlx2 + 2x1x3 + x2x3 >= 2500, x1 >= 0, x2 >= 0, x3 >= 0},
{xl, x2, x3}]
```

Результат будет следующим:

$$\left\{-50\left(-5-\sqrt{13}\right), \left\{x1 \to \frac{50}{13}\left(13-2\sqrt{13}\right), x2 \to \frac{250}{\sqrt{13}}, x3 \to \frac{50}{13}\left(13-3\sqrt{13}\right)\right\}\right\}$$

Первым здесь указано максимальное значение целевой функции, затем - значения переменных, при которых оно достигается.

Чтобы получить результат в виде десятичных дробей, следует ввести:

```
Maximize[3xl+ 5x2 + 2x3,
  {8x1+ 7x2 + 5x3 <= 1000, 2x1+ 5x3 <= 200,
   x1+x2 + x3==100, x1 + x2 >= 20,
   x1x2+ 2x1x3 +x2x3 к 2500, x1 >= 0, x2 >= 0, x3 >=0},
  {xl,  x2,  x3}]//N
```

Результат будет следующим:

```
{430.278, {x1->22.265, x2->69.3373, x3->8.39749}}
```

Примечание - Тот же результат можно было получить, используя функцию NMaximize вместо Maximize.

Пример 5.32 - Решить задачу из примера 5.31 со следующим дополнительным условием: значения переменных x^1, x^2, x^3 должны быть целочисленными.

Для этого следует ввести:

```
Maximize [3 x1+ 5 x2 + 2 x3,
  {8x1+ 1x2 + 5x3 <= 1000, 2 x1+ 5 x3 <= 200,
   x1 + x2 + x3 == 100, x1 + x2 >= 20,
   x1 x2 + 2 x1 x3 + x2 x3 >= 2500, x1 >= 0, x2 >= 0,
   x3 >= 0, xl € Integers, x2 € Integers, x3 € Integers},
   {xl, x2, x3}] // N
```

Таким образом, для каждой из переменных, которая должна принимать только целочисленные значения, указывается, что она должна принадлежать множеству целых чисел (Integers). Для ввода знака € используется палитра. Результат должен быть следующим:

{430., {x1->20.,x2->70,,x3->10.}}

Функции Minimize и Maximize (а также NMinimize и NMaximize) могут применяться для поиска экстремумов функций одной или нескольких переменных без ограничений, т.е. для решения задач, аналогичных рассмотренным в подразделах 5.11 и 5.12. Более подробные сведения об этом имеются в справочной системе Mathematica.

Пример 5.33 - Решить следующую задачу линейного программирования:

$E = 100x^1+300x^2 \rightarrow max$

$20x^1 + 5x^2 <= 200$

$10x^1 + 5x^2 <= 250$

$5x^1 + 20x^2 <= 500$

$x^1 >= 0,\ x^2 >= 0$

x^1, x^2 - целые.

Пример 5.34 - Решить следующую задачу нелинейного программирования:

$$E = \frac{e^{x_1}}{x_2 \cdot e^{x_3}} \rightarrow min$$

$x^1 + x^2 + x^3 >= 50$

$3x^1 + 2x^2 <= 600$

$x^2 >= 10$

$x^1 = 2x^2$

$x^1x^2 + x^2x^3 <= 2000$

$x^1x^2 >= 100$

$x^{2^2} + x^{3^2} = 1000$

$x^i >= 0,\ i=1,...,3.$

5.14 Решение дифференциальных уравнений

Для *аналитического* решения дифференциальных уравнений в Mathematica применяется следующая функция:

```
DSolve[уравнение,y[x],x]
```

где **уравнение** - решаемое дифференциальное уравнение;

y[x] - функция независимой переменной **x,** которую требуется определить в результате решения уравнения;

x - независимая переменная.

Для решения систем дифференциальных уравнений функция **DSolve** используется в следующей форме: `DSolve[{уравнения},{y1[x], y2[x],..., yn[x]}, x]`.

Пример 5.35 - Решить дифференциальное уравнение: $y'' - y = x e^x$, где x - независимая переменная.

Для решения этого уравнения требуется ввести:

```
DSolve[y''[x] - y[x] == xExp[x], y[x], x]
```

Для указания производной второго порядка (`y''[x]`) требуется указывать две одиночные кавычки (а не двойные кавычки).

Результат будет следующим:

$$\left\{\left\{y[x] \to \frac{1}{8} e^x (1 - 2x + 2x^2) + e^x C[1] + e^{-x} C[2]\right\}\right\}$$

Здесь C[1], C[2] - постоянные интегрирования.

Пример 5.36 - Решить систему дифференциальных уравнений:

$$y1' = y2 + 0,1x^2$$

$$y2' = -y1$$

где x - независимая переменная.

Для решения задачи требуется ввести:

```
DSolve[{y1'[x] == y2[x] + 0.1*x^2, y2'[x] == -y1[x]},
 {y1[x],y2[x]},x]
```

Результат будет следующим:

```
{{y1[x] -> C[1] Cos[1. x] + C[2] Sin[1. x] + 0.1 Sin[1. x]
      (-1. (-2. + 1. x^2) Cos[1. x] + 2. x Sin[1. x]) + 0.1
      Cos[1. x] (2. x Cos[1. x] + 1. (-2. + 1. x^2) Sin[1. x]),
  y2[x] -> C[2] Cos[1. x] - C[1] Sin[1. x] + 0.1 Cos[1. X]
      (-1. (-2. + 1. x^2) Cos[1. x] + 2. x Sin[1. x]) - 0.1 Sin[
       1. x] (2. x Cos[1. x] + 1. (-2. + 1. x^2) Sin[1, x])}}
```

Полученные выражения функций $y^1(x)$ и $y^2(x)$ можно упростить, используя функцию Simplify[выражение], где выражение - математическое выражение, которое требуется упростить. В данном случае *сразу после решения системы дифференциальных уравнений* следует ввести:

Simplify[%]

Результат будет следующим:

```
{{y1[x] ->
    0.2x Cos[1. X]^2 + Sin[1. x] (C[2] + 0.2 x Sin[1. x]) +
     Cos[1. x] (C[1] + (0. + 0. x^2) Sin[1. x]) ,
   y2[x] -> (0.2 - 0.1 x^2) Cos[1. X]^2 +
     Cos[1.x] (C[2] + 0. x Sin[1. x]) +
     Sin[1.xl (-1. C[1] + (0.2 - 0.1x^2) Sin[1.x])}}
```

Для *численного* решения дифференциальных уравнений в Mathematica применяется следующая функция:

NDSolve[{**уравнение**, начальные_условия}, {y^1, y^2,..., y^n}, {x, x^{min}, x^{max}}],

где уравнения - решаемые дифференциальные уравнения (одно или несколько);

начальные_условия - начальные условия, заданные для функций независимой переменной, которые требуется определить, и для их производных;

y^1, y^2,..., y^n - функции независимой переменной x, которые требуется определить. Если решается одно дифференциальное уравнение (а не их система), то указывается одна функция y без фигурных скобок;

x - независимая переменная;

Xmim Xmax - границы диапазона значений независимой переменной.

Пример 5.37 - Решить дифференциальное уравнение из примера 5.35 при следующих начальных условиях: y(0)=5, y'(0)=12. Построить график функции *y(x)* для значений независимой переменной *x* от 0 до 10.

1 Решить уравнение. Для этого ввести:

NDSolve[{y"[x]-y[x]==x*Exp[x], y[0]=5, y'[0]==12}, y, {x,0,10}]

Результат будет получен в следующем виде:

{{y-> lnterpolatingFunction[{{0., 10.}}, <>]})

Это означает, что получена интерполяция функции *y(x)* для значений *x* от 0 до 10. Эта интерполяция представляет собой набор таблиц значений *x* (от 0 до 10) и соответствующих значений функции y(x). Эти таблицы содержат большое количество чисел и поэтому не отображаются на экране.

2 Построить график полученной функции *y(x):*

Plot[Evaluate[y[x]/.%],{x,0,10},AxesLabel->{"x","y"}]

График функции *y(x)* для данного примера приведен на рисунке 5.3.

Здесь функция Evaluate выполняет вычисление значений функции *y(x)* на основе интерполяции этой функции, полученной в предыдущей операции, т.е. при использовании функции NDSolve. Функция Plot строит график по значениям *x* (от 0 до 10) и величинам, полученным в результате применения функции Evaluate, т.е. по значениям функции *y(x)*.

Примечание - Функция Evaluate[y[x]/.%] выполняет те же действия, что и простая подстановка y[x]/.%. Поэтому в некоторых случаях при построении графиков функций, полученных при решении дифференциальных уравнений, можно не использовать функцию Evaluate. Так, в данном примере для построения графика можно было использовать функцию Plot в следующей форме: Plot[y[x]/.%,{x,0,10}, AxesLabel->{"x","y"}]. Однако в ряде случаев подстановка y[x]/.% может не давать результата (например, если значение x оказывается очень малым числом). В этих случаях использование функции Evaluate оказывается обязательным.

Пример 5.38 - Решить систему дифференциальных уравнений:

$$y^1 = y^2 + 0,1x^2$$

где *x* - независимая переменная, при следующих начальных условиях: $y^1(0)=0$, $y^2(0)=1$. Построить графики функций $y^1(x)$ и $y^2(x)$ для значений независимой переменной *x* от 0 до 20.

1 Загрузить подпакет Legend, входящий в пакет расширения Graphics (см. подразделы 5.7, 5.8). Для этого ввести:

```
«Graphics'Legend'
```

Загрузка подпакета Legend необходима, так как по результатам решения задачи потребуется строить график двух функций: $y^1(x)$ и $y^2(x)$, и поэтому желательно будет использовать легенду.

2 Решить систему дифференциальных уравнений. Для этого ввести:

```
rez = NDSoIve[{yl'[x]==y2[x]+0.1*x^2, y2'[x]== -yl[x], y1[0]==0, y2[0]==1},
{y1,y2},{x,0,20}]
```

Будет получена интерполяция функций $y^1(x)$ и $y^2(x)$ для значений x от 0 до 20. На экран при этом выводится следующее:

```
    {{y1->InterpolatingFunction[{{0.,20.}},<>],
  y2->InterpolatingFunction[{{0.,20.}},<>]}}
```

При этом полученный результат, т.е. интерполяция функций $y^1(x)$ и $y^2(x)$, присваивается переменной rez.

3 Построить графики полученных функций:

```
Plot[Evaluate[{y1[x],y2[x]}/.rez],{x,0,20},
 PlotStyle -> {Dashing[{}], Dashing [{0.05, 0.02}]},
 PlotLegend-> {"yl", "y2" }, LegendPosition-> {1, 0},
 AxesLabel->{"X", "Y"}]
```

Результат приведен на рисунке 5.4.

4 Построить графики функций $y^1(x)$ и $y^2(x)$ *по отдельности*. Например, для построения графика функции $y^1(x)$ следует ввести:

```
Plot[Evaluate[y1[x]/.rez], {x, 0, 20},
 AxesLabel -> {"X", "Y"}]
```

Аналогично построить график функции $y^2(x)$.

Примечание - В рассмотренном примере можно было не присваивать результат решения дифференциальных уравнений какой-либо переменной. В этом случае для последующих ссылок на это решение потребовалось бы использовать обозначения %, %% и т.д.

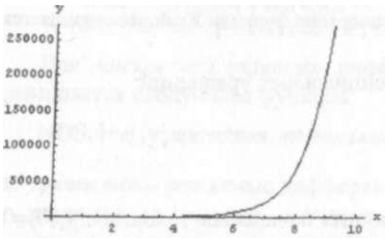


Рисунок 5.3 - Результаты примера 5.37

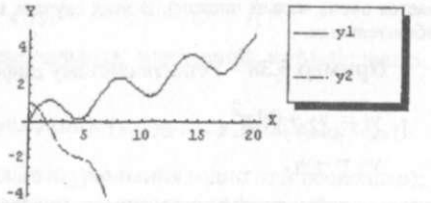


Рисунок 5.4 - Результаты примера 5.38

Хотя функции, полученные в результате численного решения дифференциальных уравнений, не могут быть выражены явно (т.е. в виде формул), можно использовать их в различных операциях, например, вычислять значения этих функций при различных значениях независимой переменной.

Пример 5.39 - Вычислить значения функции $y^1(x)$, полученной при решении примера 5.38, при различных значениях переменной ***x***.

1 Вычислить значение функции $y^1(x)$ при x=10:

```
yl[10]/.rez
```

2 Вычислить значения функции $y^1(x)$ для значений ***x*** от 0 до 10 с шагом 0,1. Вывести полученные результаты в файл. Для этого выполнить следующее:

- получить список значений переменной ***x*** от 0 до 10 с шагом 0,1:

```
x=Range[0,10,0.1];
```

(точка с запятой в конце команды требуется, чтобы полученные числа не выводились на экран);

- вычислить значения функции $y^1(x)$ для заданных значений ***x*** и вывести результаты **в** файл:

```
y1[x]/.rez>>"имя_файла"
```

Пример **5**.40 - Решить дифференциальное уравнение:

$y'' + 0{,}5y + y^3 = 10\sin x,$

где ***x*** - независимая переменная, при начальных условиях y(0)=0, y'(0)=1. Построить график функции y(x) для значений ***x*** от 0 до 10. Найти значение функции ***y(x)*** при x=8.

Пример **5**.41 - Решить систему дифференциальных уравнений:

$y1' = y2$

$y2' = -y1 + 1000(1 - y1^2)y2$

где $y1$, $y2$ - функции от независимой переменной ***x***. Найти решение при начальных условиях $y1(0)=2$, $y2(0)=0$. Построить графики функций $y1(x)$ и $y2(x)$ для значений независимой переменной ***x*** от 0 до 3000.

ГЛАВА 6

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С++

6.1 Становление объектно-ориентированного подхода к программированию

Жизненный цикл разработки программных систем:
анализ -> проектирование -> программирование -> тестирование -> эксплуатация.

Объектно-ориентированный анализ направлен на создание моделей, наиболее близких к реальности. Требования этой методологии формируются на основе понятий классов и объектов.

Объектно-ориентированное проектирование - это методология проектирования, соединяющая процесс объектной декомпозиции и приемы представления моделей проектируемой системы как логической и физической, так статической и динамической. В данном определении содержатся две важные части:

- объектно-ориентированное проектирование ведет к объектно-ориентированной декомпозиции;
- используется многообразие приемов представления моделей, отражающих логическую (структуры классов и объектов) и физическую (архитектура моделей и процессов) структуру системы. Именно поддержка объектно-ориентированной декомпозиции отличает объектно-ориентированное проектирование от структурного.

Объектно-ориентированное программирование (ООП) - это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса, а классы образуют иерархию на принципах наследуемости. В данном определении можно выделить три части: 1) ООП использует в качестве элементов конструкции объекты, а не алгоритмы; 2) каждый объект является реализацией какого-либо определенного класса; 3) классы организованы иерархически. Программа будет объектно-ориентированной только при соблюдении всех трех указанных требований.

К настоящему времени в программировании сформировалось несколько направлений:

- процедурное;
- модульное;
- объектно-ориентированное.

В процедурном программировании основное внимание уделяется алгоритму, т.е. некоторой заданной последовательности действий, выполнение которых приводит к получению результата вычислений. Главное внимание здесь уделяется построению процедур (подпрограмм). Первым процедурным языком был ФОРТРАН, потом Алгол 60, Паскаль, С и т.д.

В модульном программировании основные акценты переносятся на построение модулей. При этом необходимо определить модули, которые будут использоваться, и разделить программу на модули так, чтобы ее данные были

скрыты в этих модулях. Указанная модель переносит основные акценты на организацию данных. Модулем называется множество взаимосвязанных процедур (подпрограмм) вместе с данными, которые эти процедуры обрабатывают. Основной целью данного направления является скрытие данных в модулях. Хотя язык C++ не был специально спроектирован для поддержки модульного программирования, реализованные в нем концепции классов позволяют работать с модулями. C++ содержит все необходимое для поддержки и процедурного и модульного программирования.

В объектно-ориентированном программировании используются следующие базовые правила: определение классов, которые будут использоваться; определение всех необходимых операций для каждого класса, обеспечение расширяемости (открытости) классов на основе принципа наследуемости.

Объекты и классы

Базовыми блоками объектно-ориентированной программы являются объекты и классы. Содержательно объект можно представить как что-то ощущаемое или воображаемое и имеющее определенное поведение. Объект - это часть окружаемой нас реальности, т.е. он существует во времени и в пространстве. Объект имеет состояние, поведение и может быть однозначно идентифицирован (т.е. имеет уникальное имя).

Класс - это множество объектов, имеющих общую структуру и общее поведение. Например int X; - определение переменной X целого типа. Здесь int - это имя типа, а X - объект данного типа. Аналогично можно задать много объектов данного типа int X, Y, Z.

Мы можем говорить о типе int как об имени класса. Если объект - это что-то ощущаемое, то класс - только абстракция, используемая для описания общей структуры и поведения множества объектов. Определим теперь понятия состояния, поведения и идентификации объекта.

Состояние объекта объединяет все его поля данных (статический компонент) и текущее значение каждого из этих полей (динамический компонент). **Поведение объекта** определяет, как объект изменяет свои состояния и взаимодействует с другими объектами. **Идентификация объекта** - это свойство, которое позволяет отличать один объект от других того же или других классов.

Базовые принципы объектно-ориентированного программирования

К базовым принципам объектно-ориентированного стиля программирования относятся:

- инкапсуляция (encapsulation) или пакетирование;
- наследование (inheritance);
- полиморфизм (polymorphism);
- передача сообщений.

Инкапсуляция

Свойство инкапсуляции, присущее реальным объектам, позволяет нам видеть и различать объекты, фиксировать их реакции в ответ на внешние воздействия, наблюдать развитие процессов. Понятие инкапсуляции означает, что в качестве единого целого, называемого объектом, рассматривается некоторая структура данных, определяющая его свойства, или атрибуты, и некоторая группа методов (функций) для манипулирования этими данными, задающая по-

ведение объектов. В реализации C++ данные хранятся в структурах, напоминающих обычные структуры языка C, а поведение объектов реализуется с помощью методов (функций), связанных с этими данными специальным описанием, задающим этим функциям область действий. Кроме того, программисту предоставляются широкие возможности по управлению доступам к данным и методам объекта, что повышает надежность и модифицируемость программ.

Наследование

В реальном мире механизм наследования позволяет закрепить и передать основные черты, присущие объектам, от одного поколения к другому. Хорошей аналогией наследования может служить таксономическая схема, которой пользуются зоологи и ботаники для классификации живых организмов. В данной схеме группы более низкого уровня наследуют характеристики групп более высоких уровней. Механизм наследования помогает сделать разработку более экономной и читаемой. Совокупность классов можно описать в виде такой иерархической структуры, что:

- если класс B наследует структуру и поведение класс A, то класс A называется базовым, а класс B - производным (рисунок 6.**1).**

Рисунок 6.1 - Диаграмма наследования классов

В литературе можно встретить разные названия классов для A и B. Так, класс A называют базовым, суперклассом, родителем, предшественником. Класс B называют производным, подклассом, потомком. Различают простое наследование и множественное. В первом случае производный класс имеет один базовый класс (рисунок 6.2).

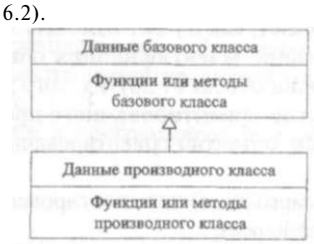


Рисунок 6.2 - Простое наследование классов

При множественном наследовании производный класс может иметь несколько базовых (рисунок 6.3).

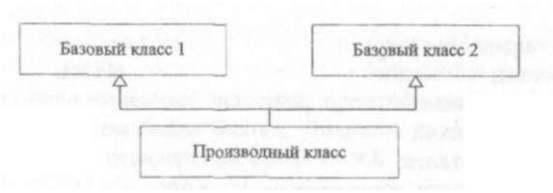


Рисунок 6.3 **- Множественное наследование классов**

Полиморфизм

Полиморфизм позволяет использовать одни и те же функции для решения разных задач. Поддержка полиморфизма в объектно-ориентированном программировании осуществляется через виртуальные функции и идею переопределения операторов. В C++ имя функции или оператор перегружаемы. Функция вызывается на основании своей сигнатуры, которая является списком типов аргументов в перечне параметров функции. Полиморфизм является одним из фундаментальных свойств ООП. Суть его заключается в том, что одни и те же имена (например, объявление переменных или методов) могут соответствовать различным классам объектов. Поэтому объект, объявленный с таким именем, будет по-разному реагировать на некоторое множество допустимых событий, в зависимости от класса объекта, который это имя представляет в текущий момент.

6.2 Основные операторы языка программирования C++

6.2.1 Операции и знаки пунктуации

Элементы программы

Составными элементами программы являются лексемы, которые состоят из символов, относящихся к базовому словарю, распознаваемому компилятором. Множество используемых символов включает в себя следующие: строчные символы - a-z, прописные символы - A-Z, цифры - 0-9 и
символы +, = , - , _ , (,) , * , ;, %, #, &, !, ", \, [,], {, }, ~, ^.

Различают пять разновидностей лексем: ключевые слова, идентификаторы, константы, операторы и пунктуацию.

Ключевые слова строго зарезервированы и имеют фиксированный смысл. Они используются для объявления типов (такие, как int, char, float), для синтаксических операторов (такие, как do, for, if).

Идентификатор (или имя переменной, функции) - это последовательность букв, цифр и подчеркиваний. Идентификатор не может начинаться с цифры. Прописные и строчные буквы обрабатываются как различные символы.

Литералы - это такие постоянные значения, как 1 или 3.14519. Все встроенные типы данных C++ имеют такие литералы, как символы, целые числа, числа с плавающей точкой и указатели. Допускаются также строковые константы. Например:
5 - целочисленная константа;
5u — и или U - определяет unsigned;

51 - 1 или L - определяет long;
05 - восьмеричная;
0x5 - шестнадцатеричная;
5.0 - с плавающей точкой;
'5' - символьная константа;
"5" - строковая константа, состоящая из '5' и '\0'.

Большому количеству символов и символьных последовательностей в C++ придается особое значение. Например:

арифметические операции:

+ (сложение), - (вычитание), * (умножение), / (деление), % (остаток от деления);

логические операции:

&& (логическое «И»), || (логическое «ИЛИ»), ! (логическое отрицание);

операции присваивании:

= (простое присваивание), +=, -=, *=, /=, %= (составные операции присваивания). Например: a=a+b; можно записать a+=b; аналогично записывается для всех составных операций присваивания;

операции сравнения:

> (больше), >= (больше или равно), < (меньше), <= (меньше или равно), == (сравнение на равенство), != (не равно).

C++ поддерживает операции инкрементации ++ и декрементации - в префиксной и постфиксной формах.

```
++i;   //i=i+1;   префиксная  форма инкрементации
--x;    //x=x-1;   префиксная форма декрементации
j=++i;  //i=i+1;j=i;
i++;    //          постфиксная форма инкрементации
x--;    //          постфиксная форма декрементации
j=i++;  //j=i;i=i+1;
```

Операции используются в выражениях. Они имеют фиксированный приоритет. Операции должны использоваться точно так, как они записаны выше, т.е. без пробелов между символами в тех операциях, которые представлены несколькими символами. Знаки препинания включают круглые скобки, фигурные скобки, запятую и двоеточие. Выражение состоит из операндов и операций. Операнд - это то, над чем выполняется действие. Операндом может быть константная или переменная величина. Операции определяют действия над операндами.

6.2.2 Типы данных

В C++ имеются следующие базовые типы данных: char (символьный), int (целый), float (вещественный), double (вещественный с двойной точностью), void (пустой тип). Базовые типы могут быть модифицированы с помощью ключевых слов: short (короткий), long (длинный), signed (знаковый), unsigned (беззнаковый). Можно использовать следующее сочетание модификаторов и типов данных:

char	signed char	unsigned char
short	int	long

unsigned short	unsigned int	unsigned long
float	double	long double

Диапазоны целочисленных значений, представленных в вашей системе, определены в стандартном файле limits.h. Диапазон значений чисел с плавающей запятой находятся в стандартном файле float.h.

6.2.3 Переменные и инициализация

Все данные программы перед их использованием должны быть объявлены или определены. Объявление переменной связывает тип с ее именем. Большинство объявлений переменной являются так же определениями. В операторах определения данных указывается их тип и перечисляются через запятую имена переменных имеющих данный тип. Каждый оператор заканчивается символом ; (точка с запятой).

Например: int a,d,c; // переменные целого типа a,d,c
float x,y; // переменные вещественного типа x, y

В модуле, в котором записано определение переменных, для каждой переменной в соответствии с типом выделяется необходимое количество байтов памяти.

Инициализировать, т.е. присвоить переменной начальное значение можно так:

```
double radius = 5.5;   // объявлено, определено и инициализировано
int a,       b=10;     // значение 10 присвоено только переменной b
```

Инициализация может включать произвольное выражение при условии, что все переменные и функции, используемые в выражении, определены. Нельзя ссылаться на неинициализированную переменную. Язык C++ позволяет многократное присвоение в одном операторе:

```
y=z=3.5;          // эквивалентно z=3.5; y=3.5;
a=b+(c=3);        // эквивалентно c=3; a=b+c;
a*=a+b;           // a=a*(a+b);
```

Преобразование типов

Если выражение имеет смешанный тип операндов, то производится преобразование типов. Согласно иерархии типов

int < unsigned < long < unsigned long < float < double

операнды в выражении преобразуется к большему типу. Например: если один операнд типа double, то и второй операнд преобразуется к типу double.

6.2.4 Ввод/вывод

Ввод/вывод не является непосредственной деталью языка. Он представляег собой дополнение в виде набора типов и подпрограмм, заложенной в стандартной библиотеке. Мы будем использовать библиотеку iostream.h.

В библиотеке iostream.h перегружаются два оператора побитового сдвига:

```
<< // поместить в выходной поток
>> // считать со входного потока
```

В этой библиотеке объявляются три стандартных потока:

```
cout   // стандартный поток вывода (экран)
cin    // стандартный поток ввода (клавиатура)
cerr   // стандартный поток диагностики (ошибки)
```

Например:

```
int i;
double x;
cout << "\n Введите число с двойной точностью";
cin >> x;     // ввод числа с плавающей точкой
cout << "\n Введите положительное число";
cin   >>i;   // ввод целого числа
cout << "i*x=" << i*x << endl;          // вывод результата
```

Идентификатор endl называется манипулятором. Он очищает поток cerr и добавляет новую строку.

6.2.5 Операторы

Операторы служат основными строительными блоками программы. Программа состоит из последовательности операторов и знаков пунктуации. Оператор является законченной инструкцией для компьютера. В языке C++ указанием на наличие оператора служит символ «точка с запятой», стоящий в конце оператора. Поэтому prim=10 является выражением, а prim=10; - оператором.

Составной оператор - это несколько операторов, заключенных в блок с помощью фигурных скобок {} или разделенных запятой. Пустой оператор - это знак «;». Пустой оператор можно использовать там, где не надо выполнять действий.

Условные операторы if и if_else

Обобщенная форма оператора if имеет вид

```
if( выражение)
        оператор;
```

Если значение "выражение" отлично от нуля (true), то оператор выполняется, если равно нулю - оператор пропускается. Выражение в операторе if - это сравнение, равенство или логическое выражение.

Оператор if_else имеет форму:

```
if( выражение)
        оператор1;
else
        оператор2;
```

Если выражение отлично от нуля, то выполняется оператор1, а оператор2 пропускается, если выражение равно нулю, то пропускается оператор1 и выполняется оператор2.

Оператор1 и оператор2 могут быть как простыми, так и составными. Например:

```
if(x<y)
        min=x;        // простой оператор
else
{
        min=y;        // составной оператор
```

```
        mm=10;
    }
        cout << "min=" << min;
```

Операторы цикла

Под циклом понимается оператор или группа операторов, которые необходимо выполнить несколько раз. Каждое выполнение цикла называется итерацией. Цикл, не содержащий внутри себя других циклов, называется простым. Сложным называется цикл, который содержит внутри другие циклы или разветвления.

Оператор for - итерационный оператор цикла, обычно используемый с переменной, которая увеличивается или уменьшается. Конструкция оператора **for** следующая:

```
for(выражение1; выражение2; выражение3)
        оператор;
```

Сначала вычисляется выражение1. Обычно выражение1 инициализирует переменную, используемую в цикле. Это выражение вычисляется только один раз. Затем вычисляется выражение2. Если оно отлично от нуля, то выполняется оператор, обрабатывается выражение3, проверяется выражение2 и т.д., пока выражение2 не станет равным нулю. Пример:

```
int i, sum=0;
for(i=0; i<=10; i++)      // вычисление суммы 10 чисел
        sum+=i;
```

В операторе for могут отсутствовать любое либо все выражения, но должны оставаться точки с запятой:

```
for(i=l, sum=0;  ; sum+=i++)  // бесконечный цикл
        cout << sum << endl;
```

Можно применять операцию уменьшения для счета в порядке убывания

```
int sum=0;
for(int n=10; n> 0 ; n--)
        sum+=n;
```

Оператор while

Обобщенная форма оператора while:

```
while(выражение)
        оператор;
```

Вначале вычисляется выражение. Если результат отличен от нуля, тогда выполняется оператор и управление переходит обратно к началу цикла **while.** Это приводит к выполнению тела цикла **while,** а именно оператора, который будет выполняться до тех пор, пока выражение не станет равным нулю. Пример:

```
int i=1, sum=0;
while(i<=10)              // вычисление суммы 10 чисел
{
        sum+=i;
        ++i;
}
```

Оператор do_while

Конструкция оператора do_while следующая:

```
do
        оператор;
while(выражение);
```

Сначала выполняется оператор, затем вычисляется выражение. Если его результат отличен от нуля, то управление переходит обратно к началу оператора do. Например:

```
int i=0, sum=0;
do
{
        sum+=i;      // суммирование положительных чисел
        cout < "\n Введите положительное целое число");
        cin >> i;    // ввод числа
} while(i>0);        // пока число положительное
```

Операторы передачи управления

Оператор switch

В программе часто необходимо в зависимости от того или иного результата реализовать ту или иную группу инструкций. Оператор switch позволяет выбрать одну из нескольких альтернатив. Он записывается в следующем формальном виде:

```
switch(целое  выражение)
{
        case   метка1: вариант 1; break;
        case   метка2: вариант 2; break;

        case   метка n: вариант n; break;
        default: вариант n+1; break;
```

Порядок работы оператора switch следующий:

1 Вычисляется выражение в круглых скобках, стоящих за switch.

2 Выполняется метка case, совпадающая с тем значением, которое было найдено на этапе 1; если ни одна из case не соответствуют этому значению, выполняется метка default; если метки default нет, switch прерывается.

3 Выполнение switch прерывается, когда встречается инструкция break или когда достигается конец switch.

Пример:

```
#include<iostream>
using namespace std;
int main()
{
        float  a,b,rez;
        char des;
        cout << "Введите числа a и b" << endl;
        cin >> a >> b;
        cout << "Введите операцию" << endl;
```

```
cin >> des;
switch(des)
{
        case '+': rez=a+b; break;
        case      rez=a-b;  break;
        case '*': rez=a*b; break;
        case ':':  rez=a/b;  break;
        default:  cout << "Операция не определена" << endl;
}
cout << "Результат" << rez << endl;
}
```

Оператор break

Оператор break используется в операторах цикла for, while, do_while и в операторе switch, он вызывает выход из самого глубоко вложенного цикла или оператора switch. Например:

```
// выход из цикла по отрицательному значению
int i;
float  x;
for(i=0; i < 10; i++)
{       cout << "Введите число" << endl;
        cin >> x;
        if( x< 0.0)
        {
                cout << "\nОтрицательное число ";
                break;      //выход из цикла по отрицательному значению
        }
}
```

Оператор continue

Оператор continue заставляет прекратить текущую итерацию цикла и начать следующую.

```
// вычисление суммы положительных чисел
int i;
float x, sum=0;
for(i=0; i < 10; i++)
{
        cout << "Введите число" << endl;
        cin >> x;     // ввод числа
        if(x< 0.0) continue;
        sum+=x;     // вычисление суммы
}
```

6.2.6 Массивы

Одномерные массивы

Массив - это тип данных, который используется для представления последовательности однородных значений. Массив представляет собой группу элементов одного типа. Объявляется одномерный массив следующим образом:

```
int temp[20];
```

Квадратные скобки [] показывают, что temp - имя массива, а число, заключенное в скобки, обозначает количество элементов массива. Нумерация элементов массива начинается с нуля, поэтому temp[0] является первым, а temp[19] - последним элементом. Отдельный элемент массива определяется при помощи его номера или индекса. Элементы массива размещаются в памяти последовательно (рисунок 6.4). Имя массива является указателем на первый элемент массива.

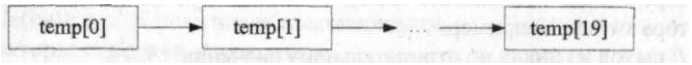


Рисунок 6.4 - Расположение элементов одномерного массива

```
float mas[10];     // массив mas содержит 10 элементов типа float
int mas1[20];      // массив mas1 содержит 20 элементов типа int
int n=5;
int mm[n];         // ошибка, переменная не может задавать размер массива
```

При объявлении массивы можно инициализировать. Для этого указывается список начальных значений элементов, заключенных в фигурные скобки:

```
int mas[5]={l,2, 3, 4, 5};
```

Количество элементов в фигурных скобках не должно превышать размерность массива. Инициализировать можно не все элементы, а любое количество первых элементов:

```
int mas[10]={l,2,3,4,5,6};
```

Все остальные четыре элемента будут проинициализированы нулями. Инициализировать массив можно следующим образом:

```
int mas[]={11,22,33,44,55};    // массив из пяти элементов
```

В этом случае количество элементов массива определяется по списку инициализации.

Пример программы: ввести массив чисел и вычислить сумму положительных значений:

```
#include<iostream>
using namespace std;
#define N  100
void main(void)
{     int mas[N];              // массив чисел
      int summa=0;             // сумма положительных значений
      int n;                   // количество чисел
      int i;
      cout << "Введите количество чисел не более"<<N<< endl;
```

```
    cin >> n;
    cout<<"Введите"<<n<<" элементов массива"<<end];
    for(i=0; i < n; i++)        // цикл по элементам массива
        cin>>mas[i];      // ввод i-го элемента массива
    for(i=0; i < n; i++)
    {
        if(mas[i]>0)          // если i-й элемент массива положительный
        summa+=mas[i];  // суммируются положительные элементы
    }
    cout<<"Сумма положительных значений равна"<<summa<<endl;
}
```

Двухмерный массив

Двухмерный массив - это массив массивов, т.е. его элементами являются массивы.

Двухмерный массив объявляется следующим образом;

```
int mas[4][5];
```

Число в первых квадратных скобках указывает количество строк массива, а число во вторых квадратных скобках указывает количество столбцов. Для доступа к элементу двухмерного массива необходимо указать все его индексы:

```
rez= mas[l][2]; //переменной rez присваивается значение третьего элемента второй строки.
```

Элементы многомерных массивов располагаются в памяти компьютера построчно. Многомерные массивы можно инициализировать. Например:

```
int  mas[3][3]={1,2,3,
                4, 5, 6,
                7, 8,9};
```

Если необходимо проинициализировать не все элементы строки, то в списке инициализации можно использовать фигурные скобки, охватывающие значения для этой строки. Например:

```
intmas[3][3]={{l},
              {2,3},
              {4,5,6}
             };
```

Пример программы: отсортировать главную диагональ двухмерного массива по возрастанию:

```
#include<iostream>
#include<iomanip>
using namespace std;
#define N 4
int main()
{     int i,j;
      int temp;
      int mas[N][N];            // двухмерный массив
      cout<<"Введите элементы массива"<<endl;
      for(i=0; i < N; i++)      // цикл по строкам массива
        for(j=0; j < N; j++)    // цикл по столбцам массива
```

```
            cin>>mas[i][j];     // ввод элемента массива
// Вывод на экран исходного массива
        cout<<"Исходный массив"<<endl;
       for(i=0; i < N; i++)
       {
         cout<<endl;
         for(j=0;j<N;j++)
            cout<<setw(4)<<mas[i][j];
       }
       for(j=N-1;j>0;j--)
             for(i=0; i< j; i++)
                   if(mas[i][i] > mas[i+1][i+1])
                   {
                         temp=mas[i][i];              // обмен местами элементов
                         mas[i][i]=mas[i+1][i+1 ]; // диагонали массива
                         mas[i+1][i+1]=temp;
                   }
       cout<<"Результат"<<end1;      // Вывод на экран результата
       for(i=0; i < N; i++)
       {
         cout<<end1;
         for(j=0;j<N;j++)
            cout<<setw(4)<<mas[i][j];
       }
}
```

В приведенном примере имя массива является постоянным указателем, который нельзя изменить.

6.2.7 Строковые данные

char - символьный тип размером в 1 байт.

Если объявить char str[10]; и инициализировать следующим образом:

```
for(int i=0; i < 10; i++)
{
       cout<<"Введите символ"<<endl;
       cin>>str[i];
}
```

то в выделенную область запишутся символы, и это будет массив символов. Если инициализировать так gets(str);, то функция gets в конец строки дописывает '\0' - признак конца строки, и это будет строка. Например:

```
       char ss[10];          // массив на 10 символов
       gets(ss);              //строка
```

В массив ss можно записать 9 символов, при этом необходимо оставлять одно место для признака конца строки ('\0').

6.2.8 Лабораторная работа №1
Работа с массивами

Цель: ознакомиться со структурой массивов. Понять, как объявлять одномерный и двухмерный массивы и обращаться к отдельным элементам массива.

Варианты заданий

1 Дано натуральное n *(n >= 2)*. Найти все меньшие n простые числа, используя решето Эратосфена. Решетом Эратосфена называют следующий способ. Выпишем подряд все целые числа от 2 до n. Первое простое число 2. Подчеркнем его, а все большие числа, кратные 2, зачеркнем. Первое из оставшихся чисел *3*. Подчеркнем его как простое, а все большие числа, кратные 3, зачеркнем. Первое число из оставшихся теперь 5, так как *4* уже зачеркнуто. Подчеркнем его как простое, а все большие числа, кратные 5, зачеркнем и т.д.

2 Пусть дан массив $a^1,...,a^n$. Требуется переставить $a^1,...,a^n$ так, чтобы массив начинался с группы элементов, численно больших того элемента, который в исходном массиве располагался на первом месте, затем должен следовать сам этот элемент, потом - группа элементов, меньших или равных ему.

3 Даны два целочисленных массива: $a^1,...,a^n$ и $b^1,...,b^n$. Вывести на печать все пары индексов, для которых произведения: *a[i]*b[i] > 10*. Подсчитать количество пар и сумму этих произведений.

4 Дан одномерный массив $a^1,...,a^n$. Найти и напечатать номер элемента, произведение которого с предыдущим

5 Рассортировать одномерный массив по возрастанию (убыванию) элементов (метод пузырька).

6 В одномерном массиве $a^1,...,a^n$ заменить отрицательные элементы нулями, подсчитать число замен *m,* вычислить *ml.*

7 Даны действительные числа $a^1...,a^n$. Получить квадратную матрицу порядка и следующего вида:

a^1 a^2 a^3 ... a^{2n-2} a^{n-1} a^n

a^2 a^3 a^4 ... a^{n-1} a^n a^1

a^3 a^4 a^5 ... a^n a^1 a^2

.....

a^n a^1 a^2 ... a^{n-3} a^{n-2} a^{n-1}

8 Дан двухмерный массив *A* размером *nm*. Определить количество положительных, отрицательных и равных нулю элементов матрицы *A*.

9 Написать программу сортировки *i-й* строки матрицы *MM* методом «пузырька». Исходную и преобразованную матрицы вывести на печать.

10 Отсортировать строки массива A размером n*m по убыванию.

6.3 Указатели и функции

6.3.1 Понятие указателя

Указатели в языке C++ используются для связи переменных с машинными адресами. В программах указатели используются для доступа к памяти и

манипуляций с адресами. Для данного типа **T** тип **T*** является "указателем на **T"** т.е. переменная типа **T*** содержит адрес объекта типа **T.** Если v - переменная, то &v - это адрес или место в памяти, где хранится ее значение. Объявление int *p; говорит о том, что переменная p будет иметь тип: "указатель на целое". Если p - указатель, то *p - значение переменной, на которую указывает p:

```
int  i=5,j;
int *p;
p=&i;   //p указывает на i
j=*p;    //j=5
```

Если объявить переменную типа "указатель", например int *ptr; int n; то с переменной типа "указатель" можно выполнять следующие операции:

p=&n; // присвоение адреса переменной n переменной p
операция увеличения(уменьшения) указателя;
операции инкремента(декремента);
операция вычитания указателей;
для указателей определяются операции отношения.

Адреса переменных можно использовать в качестве аргументов функций. В результате значения переменных могут изменяться в вызывающем окружении. Указатели используются в списке параметров для определения адресов переменных, значения которых могут изменяться. Например:

```
void order(int *, int *);  //прототип функции
int main()
{
        int i=7, j=3;       // инициализация переменных i, j
        cout<<i<<'\t'<<j<<endl;
        order(&i,&j);      // вызов функции на выполнение
        cout<<i<<'\t'<<j<<endl;
}
// определение функции order
void order(int *p, int *q)
{       int temp;
        if(*p > *q)
        {
                temp=*p;
                *p=*q;
                *q=temp;
        }
}
```

6.3.2 Массивы и указатели

В языке C++ массивы и указатели тесно связаны. Имя массива можно использовать в качестве указателя на его первый элемент. Например:

```
int mas[]={1,2,3,4,5};
int       *p1=mas;         // указатель на первый элемент массива
```

```
int *p2=&mas[0];          // указатель на первый элемент массива
int *p3=&mas[4];          // указатель на последний элемент массива
```

Графически это можно показать так (рисунок 6.5):

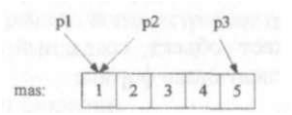


Рисунок **6**.5 - Указатели на элементы массива

Если pl,p2,p3 указывают на некоторый элемент массива, то p1+1 указывает на следующий элемент массива. Если p1 указывает на mas[0], то *p1 есть содержимое mas[0], т.е. содержит число 1.

Между именем массива и указателем, содержащим его адрес, имеется существенное различие. Указатель - это переменная, поэтому его можно изменять. Имя массива - это указатель-константа и его изменять нельзя.

6.3.3 Динамическое распределение памяти

Память под массивы можно отводить динамически, т.е. размещать в свободной памяти (free store). Свободная память - это предоставляемая системой область памяти для объектов, время жизни которых устанавливается программистом. Динамически память распределяется с помощью оператора new. В общем виде new можно использовать следующим образом:

```
new имя_типа;
new имя_типа(выражение);
new имя_типа[выражение];
```

Оператор new выделяет надлежащий объем свободной памяти для хранения указанного типа и возвращает базовый адрес объекта. Когда память недоступна, оператор new возвращает NULL либо возбуждает соответствующее исключение:

```
int *p,*q,size,i;
p=new int(5);   //выделение памяти и инициализация
cout<<"Введите размер массива"<<endl;
cin>>size;
q=new int[size]; //выделение памяти под массив
cout<<"Введите элементы массива"<<endl;
for(i=0; i < size; i++)
  cin>>q[i];
```

Можно динамически выделить память под двухмерный массив, используя "указатель на указатель". В языке C++ допустимо объявлять переменные, имеющие тип «указатель на указатель». Объявляется "указатель на указатель" следующим образом: int **mas; Фактически "указатель на указатель" - это адрес ячейки памяти, хранящей адрес указателя. Например:

```
int **mas;
int n, m, j;
```

```
cout<<"Введите количество строк и столбцов"<<endl;
cin>>n>>m;        // ввод количества строк и столбцов
mas=new int*[n];    // выделение памяти под массив указателей
for(j=0;j<n;j++)
        mas[j]=new int[m];
```

Оператор delete уничтожает объект, созданный с помощью new. Оператор delete может принимать следующие формы:

```
delete выражение;
delete [] выражение;
```

Первая форма используется, если соответствующее выражение new размещало не массив. Во второй форме присутствуют пустые квадратные скобки, показывающие, что изначально размещался массив объектов. Оператор delete не возвращает значения.

6.3.4 Массивы символьных строк

Массив символьных строк можно объявить следующим образом:

```
#define N 5    // объявление константы
char *str[N];   //массив указателей
int      n;      // количество символов в строке
// Выделить память под массив строк
for(i=0; i<N; i++)
{   cout<<"Введите    количество символов в строке"<<endl;
      cin>>n;            //ввод количества символов
      str[i]=new char[n]; //выделение памяти под строку
}
 // инициализировать массив строк
cout<<"Введите строки длиной не более"<<n-1<<"символов"<<endl;
for(i=0; i<N; i++)
      gets(str[i]);        //ввод строки
```

Используя указатель на указатель, массив строк можно объявить следующим образом:

```
char **str;
int n,m;                //количество строк и длина строки
cout<<"Введите количество строк и длину строки"<<endl;
cin>>n>>m;        //ввод количества строк и длину строки
// Выделить память под массив строк
str=new char*[n];     // выделение памяти под массив указателей
for(i=0; i<n; i++)
   str[i]=new char[m];  //выделение памяти под строку
```

Функции для работы со строками

Ввод строк

Функция gets(char *);

Аргументом функции является указатель на строку. Функция читает строку до тех пор, пока не встретит символ ввода '\n'. После считывания сим-

вола '\n' функция превращает его в символ конца строки '\0' и добавляет его в конец строки, поэтому в строке необходимо оставить место для символа'\0'.

Вывод строк

Функция puts(char *); Аргументом функции является указатель на строку. Функция puts() прекращает работу, когда встречает признак конца строки '\0':

```
charstr[10]="Зима";
puts(str);
```

Функция strlen(). Прототип функции

```
unsigned strlen(char *str); // вычисляет длину строки, не включая признак
                            // конца строки '\0'.
```

Функция strcmp(). Прототип функции

```
int strcmp(char *str1, char *str2);
```

Сравнивает строки str1 и str2. Результат - отрицательный, если str1 < str2, равен нулю, если str1==str2, и положителен, если str1 > str2. Функция возвращает разницу между кодами ASCII первой пары несовпадающих символов.

6.3.5 Функции

Функция - самостоятельная единица программы, спроектированная для реализации конкретной задачи. Программисту необходимо знать, как функцию определить, т.е. написать ее код, как к функции обращаться, т.е. вызвать ее на выполнение и как устанавливать связи между функцией и программой ее вызывающей. Чтобы установить связь между функцией и программой ее вызывающей, необходимо знать прототип функции. Прототип функции - это ее объявление. Прототип функции имеет следующую форму (рисунок 6.6):

Рисунок **6.6** - Прототип функции

Например: float s_z(int a, int b);

Список аргументов может быть пустым, содержать один аргумент или несколько, разделенных запятыми. Если функция не имеет аргументов, допускается использование ключевого слова void. Если же функция не имеет аргументов и ничего не возвращает, то ее прототип можно записать в следующем виде: void prim(void);

Вызов функции

Программа на языке С++ создается из одной или более функций, одна из которых main(), называется головной. Выполнение программы всегда начинается с функции main(). Вызов любой функции осуществляется по ее имени. Вышеописанную функцию s_z(), на выполнение надо вызвать так:

```
int a=10,  b=20;  // объявление и инициализация переменных
float rez;
rez=s_z(a,b);      // вызов функции на выполнение
```

Определение функции

В языке C++ код, описывающий, что делает функция, называется ее определением. Формально это выглядит так:

тип_возв_рез имя_ф(тип имя_n1, тип имя_n2, ...)

{ тело функции }

Синтаксически аргументы - это идентификаторы, которые могут использоваться внутри тела функции. Иногда параметры в определении функции называют формальными. Формальные параметры - это то, вместо чего будут подставлены фактические значения, передаваемые функции в момент ее вызова. После вызова функции значение аргумента, соответствующее формальному параметру, используется в теле выполняемой функции. В языке C++ такие параметры являются передаваемыми по значению. Когда применяется вызов по значению, переменные передаются функции как аргументы, их значения копируются в соответствующие параметры, а сами переменные не изменяются в вызывающем окружении.

Инструкция return

Инструкция return используется для двух целей. Когда она выполняется, управление программой немедленно возвращается в вызывающее окружение. Кроме того, если за ключевым словом return следует какое-либо выражение, то его значение также передается в вызывающее окружение. Инструкция return имеет следующие формы записи:

return;

return выражение;

return (выражение);

Структура простой программы

```
#include <iostream>
using namespace std;
void main(void)
{
    объявление прототипов функций
    объявление переменных
    ввод данных
    вызов функций на выполнение
}
    определение функций
```

Например: написать функцию, которая вычисляет среднее значение двух целых чисел

```
#include <iostream>
using namespace std;
void main(void)
{
    float s_z(int a, int b);   // прототип функции
    int a,b;                   // два исходных числа
    float rez;                 //результат
    cout<<"Введите два целых числа"<<endl;
    cin>>a>>b;
```

```
        rez=s_z(a,b);              // вызов функции на выполнение
        cout<<"Среднее значение равно"<<rez<<endl;
}
// Определение функции
float s_z(int a, int b)
{
        float r;
        r=((float)a+b)/2;       // или return (((float)a+b)/2);
        return r;
}
```

Передача параметров **в** функцию **по** ссылке

Язык C++ вводит новый тип, называемый ссылкой, которая по смыслу тесно связана с указателями. Ссылка определяется следующим образом:

тип_данных&,

что означает косвенное обращение к определенному данному.

Ссылка вводит другое имя, или синоним, для объекта. В результате можно передать ссылку(синоним) в другую функцию в виде параметра. Доступ к переменной может осуществляться через ее имя в вызывающей программе и через имя-синоним в вызываемой программе. После завершения вызываемой программы имя-синоним уничтожается, однако измененное значение переменной в вызывающей программе сохраняется. Таким образом в языке C++ аргументы можно передать в функцию тремя способами:

- по значению;
- через указатель;
- через ссылку.

Пример: функция производит обмен значениями между двумя переменными

```
void swap(int& a, int& b)  // определение функции
{
        int temp;
        temp=a;              // обмен значениями между переменными
        a=b;
        b=temp;
}
int main()
{
        void swap(int& a, int& b); // прототип функции
        int a, b;                  // объявление переменных
        cout<<"Введите a и b"<<endl;
        cin>>a>>b;
        swap(a,b);                 // вызов функции на выполнение
        cout<<"a="<<a<<"b="<<b<<endl;
}
```

6.3.6 Перегрузка функций

Обычно выбор имени функции основан на стремлении отразить в названии ее основное назначение. Иногда различные функции используются для одних и тех же целей. Перегрузка использует одно и то же имя для нескольких вариантов функций. Выбор конкретного варианта зависит от типов аргументов, используемых функцией. Компилятор выбирает функцию в соответствии с типами аргументов и их количеством. Правило, по которому осуществляется этот выбор, называется алгоритмом соответствия сигнатуре. Под сигнатурой мы понимаем список типов, который используется в объявлении функции. Язык C++ позволяет определить несколько функций с одним и тем же именем, если эти функции имеют различные наборы параметров (по крайней мере, различные типы параметров). Эта особенность называется перегрузкой функций. Перегруженные функции различаются с помощью их сигнатуры - комбинации имени функции и типов ее параметров. Компилятор кодирует идентификатор каждой функции по числу и типу ее параметров. Перегруженные функции могут иметь и различные типы возвращаемых значений, но обязательно должны иметь различные списки параметров. Для различия функций с одинаковыми именами компилятор использует только списки параметров. Перегруженные функции не обязательно должны иметь одинаковое количество параметров.

```
// Пример использования перегруженных функций:
#include<iostream>
using namespace std;
int square(int x)

    return x*x;

double square(double y)

    return y*y;

void main(void)

    cout<<"Квадрат целого числа 7="<<square(7)
        <<"\n   Квадрат числа 7.5="<<square(7.5)<<endl;
```

6.3.7 Лабораторная работа №2
Динамическое распределение памяти. Функции

Цель: изучить операции работы с указателями, научиться использовать функции динамического выделения памяти для одномерных и двухмерных массивов, создавать и применять новые функции пользователя. Понять механизм обмена информацией между функциями.

Варианты заданий

В данной лабораторной работе программу требуется разбить на функции, память под массивы - отводить динамически.

1 В квадратной матрице порядка n найти наибольший элемент по модулю. Получить квадратную матрицу порядка n-*1* путем выбрасывания из исходной матрицы каких-либо строки и столбца, на пересечении которых расположен элемент с найденным значением.

2 В двухмерном массиве среди чисел, стоящих на четных местах, определить минимальный положительный элемент массива и его индексы.

3 Рассортировать положительные элементы каждой строки матрицы по убыванию. Отрицательные элементы оставить на своих местах.

4 Рассортировать отрицательные элементы каждого столбца матрицы по возрастанию. Положительные элементы оставить на своих местах.

5 Рассортировать элементы побочной диагонали квадратной матрицы порядка и по возрастанию.

6 Рассортировать элементы главной диагонали квадратной матрицы порядка и по возрастанию.

7 В некоторых видах спортивных состязаний выступление каждого спортсмена оценивается несколькими судьями независимо друг от друга, затем изо всей совокупности оценок удаляются наиболее высокая и наиболее низкая, а из оставшихся оценок вычисляется среднее арифметическое, которое и идет в зачет спортсмену. Если наиболее высокую оценку выставили несколько судей, то изо всей совокупности удаляется только одна наивысшая оценка; аналогично поступают с наиболее низкими оценками. Даны натуральное число n, действительные положительные числа $a^1,...,a^n$ *(n>=3).* Считая, что числа $a^1,...,a^n$ - это оценки, выставленные судьями одному из участников соревнований, определить оценку, которая пойдет в зачет этому спортсмену.

8 Даны действительные числа $a^1,...,a^n$. Требуется умножить все члены последовательности $a^1,...,a^n$ на квадрат ее наименьшего числа, если $a^1 >= 0$, и на квадрат ее наибольшего числа, если $a^1 < 0$.

9 У прилавка магазина n покупателей выстроились в очередь. Время обслуживания продавцом *i-го* покупателя равно t^i *(i=1,...,n).* Пусть даны натуральное n и действительные $t^1,\ldots,t^n$. Получить $c^1,...,c^n$, где c^i - время пребывания *i-го* покупателя в очереди *(i=1,...,n).* Указать номер покупателя, для обслуживания которого продавцу потребовалось самое малое время.

10 В матрице целых чисел определить максимальный элемент на главной диагонали и проверить есть ли такой элемент ниже главной диагонали; если есть, то определить его координаты.

6.4 Классы и объекты в языке C++

6.4.1 Структуры

Если надо под одним именем собрать различные типы данных, необходимо использовать структуры.

Структурный шаблон является основной схемой, описывающей, как собирается структура.

```
struct book          // шаблон структуры
{       char tit[10];
        char aut[20];
        float value;
};
```

Ключевое слово struct определяет, что все, что следует за ним, является структурой, за которой следует имя типа структуры.

struct book libry; // объявляет libry структурой типа book

Под переменную libry отводится память (рисунок 6.7):

tit[10]	aut[20]	value
10 байт	20 байт	4 байта

Рисунок **6.7 - Выделение памяти под структуру**

Массив структур объявляется следующим образом:

struct book libry[2];

Под объявленный массив будет отведена память (рисунок 6.8).

libry[0]	**libry[0].tit**	**libry[0].aut**	**libry[0].value**
libry[1]	**libry[1].tit**	**libry[1].aut**	**libry[1].value**

Рисунок **6.8 - Выделение памяти под массив структур**

Для обращения к отдельным элементам структуры используется оператор точка

Например:

```
struct book                 // шаблон структуры
{
        char tit[40];       // название книги
        char aut[20];       // фамилия автора
        float value;        // цена книги
};
struct book libry[4];       // массив структур
for(int i=0; i < 4; i++)
{    cout<<"Введите    название книги"<<endl;
        gets(libry[i].tit);
        cout<<"Введите ФИО автора"<<endl;
        gets(libry[i].aut);
        cout<<"Введите цену книги"<<endl;
        cin>>libry[i].value);
        fflush(stdin);
}
```

Можно объявить указатель на структуру

```
struct book *pt;
```

Указатель необходимо проинициализировать, т.е. присвоить ему некоторый адрес, например:

```
pt= new book;
```

В этом случае обращение к элементам структуры осуществляется с помощью оператора '->', т.е. pt->tit; pt->aut; pt->value;

Для объявления массива структур, используя указатель на структуру необходимо выполнить следующие действия:

```
struct book                  // объявить шаблон структуры
{
      char tit[40];
      char aut[20];
      float value;
};
int n;                       // количество структур
struct book *libry;          // указатель на структуру
cout<<"Введите количество структур"<<endl;
cin>>n;                      // ввод количества структур
libry=new book[n];           // выделение памяти под n структур
for(int i=0; i < n; i++)
{
      cout<<"Введите название книги"<<endl;
      gets(libry[i]->tit);
      cout<<"Введите ФИО автора"<<endl;
      gets(libry[i]->aut);
      cout<<"Введите цену книги"<<endl;
      cin>>libry[i]->value;
      fflush(stdin);
}
```

6.4.2 Объединения

Объединение - это средство, позволяющее запоминать данные различных типов в одном и том же месте памяти. Объединение позволяет создавать массив из элементов одинакового размера, каждый из которых может содержать различные типы данных. Различаются шаблон объединения и переменная типа объединения.

Шаблон объединения объявляется

```
union hh             // шаблон объединения
{
      int d;
      double b;
      char f;
};
union hh ft;         // ft - переменная типа объединения hh
```

```
union hh mas[5];     // массив объединений
union hh *pu;        // указатель на переменную типа объединения hh
```
Под переменную ft компилятор выделяет достаточно памяти для размещения самой большой из описанных переменных. Вот как используется объединение:
```
ft.d=23;             // 23 записывается в ft
ft.b=2.5;            // 23 стирается, а 2.5 записывается
ft.f='h';            // 2.5 стирается, а записывается код символа h
```
В каждый момент времени запоминается только одно значение. Если объявлен указатель на объединение, то сначала этот указатель необходимо проинициализировать pu=new hh;. Доступ к элементам объединения в этом случае осуществляется через оператор ->, т.е. pu->d;, pu->b;, pu->f;

6.4.3 Область видимости и классы памяти

Базовый язык включает в себя два основных вида области видимости: локальную и видимости файла. Локальная область видимости относится к блоку. Тело функции - это пример блока, оно содержит набор объявлений, включая параметры функции. В область видимости файла входят имена, которые являются внешними (глобальными). Основное правило области видимости состоит в том, что идентификаторы доступны только внутри блока, в котором они объявлены. Они не известны за границами блока:
```
int a=2;                  // a объявлено вне блока
cout<<a<<endl;            // напечатается 2
{
    int a=7;              // a объявлено внутри блока
    cout<<a<<endl;        // напечатается 7
}
cout<<++a<<endl;          // напечатается 3
```
Каждая переменная и функция в языке С имеет два атрибута: тип и класс памяти. Классов памяти четыре: автоматический, внешний, регистровый и статический: им соответствуют ключевые слова:

auto, extern, register, static

Переменные, объявленные внутри тела функции, по умолчанию является автоматическими, поэтому ключевое слово auto встречается редко.

Класс памяти extern

Когда переменная объявлена вне функции (ее класс памяти extern), она будет внешней или глобальной. Внешняя переменная рассматривается как глобальная для всех функций, объявленных после нее, и при выходе из блока или функции такая переменная продолжает существовать. Ключевое слово extern используется для того, чтобы приказать компилятору «где-то поискать переменную» в этом файле или в каком-то другом. Таким образом, два файла можно компилировать отдельно. Внешние переменные существуют на протяжении всей жизни программы, их можно использовать для передачи значений между функциями.

Класс памяти static

Статические объявления имеют два различных важных применения. Статические переменные сохраняют предыдущие значения при повторном входе в блок в отличие от обычных автоматических переменных, которые теряют свои значения при выходе из блока.

Например:

```
int fn()
{       static  int c=0;
        ++c;

        return c;
}
```

В первый раз, когда выполняется функция, переменная *c* инициализируется нулем. При выходе из функции значение *c* сохраняется в памяти. Когда функция вызывается снова, *c* не переинициализируется; она сохраняет свое значение с последнего вызова функции. Второе и более тонкое применение static связано с внешними объявлениями. Применение static ограничивает область видимости переменных и функций. Статические переменные и функции видны только внутри файла, в котором они определены.

Объявление вводит имя в область видимости. Это значит, что имя может использоваться только в определенной части текста программы. Для имени, объявленного в теле функции (такое имя называют локальным), область видимости начинается с места объявления имени и заканчивается в конце блока, в котором это имя объявлено. Имя называется глобальным, если оно объявлено вне любой функции, класса или пространства имен. Область видимости глобальных имен простирается от места их объявления до конца файла, содержащего объявление. Объявление имени в блоке может скрыть объявление этого имени в охватывающем блоке или глобальное имя, т.е. имя может быть замещено внутри блока и будет ссылаться там на другую переменную. После выхода из блока имя восстанавливает свой прежний смысл. Например:

```
int x;          // глобальная переменная x
void f()
{       int x;  // локальная переменная x скрывает глобальную переменную x
        x=1;  // присваивание локальной переменной x
        {       int x;  // скрывает первую локальную переменную x
                x=2;  // присваивание второй локальной переменной x
        }
        x=3;  // присваивание первой локальной переменной x
}

int *p=&x;  // взять адрес глобальной переменной x
```

6.4.4 Пространство имен

При совпадении имен элементов в одной области действия нередко возникает конфликт имен. Наиболее часто это возникает при использовании различных пакетов библиотек, содержащих, например, одноименные классы. Про-

странство имен используется для разделения глобального пространства имен. Синтаксис пространства имен некоторым образом напоминает синтаксис структур и классов. После ключевого слова namespace следует необязательное имя пространства имен, затем описывается пространство имен, заключенное в фигурные скобки. Например:

```
namespace NAME
{       int a;  double b;
        char* fun(char *, int);
        class CLS
        {
                public:
                ...
```

Если обращение к элементам пространства имен производится вне контекста, его имя должно быть полностью квалифицировано, используя оператор принадлежности ::

```
NAME::b=2;
NAME::fun(str,NAME::a);
```

Если в каком-либо месте программы интенсивно используется некоторый контекст и все имена уникальны по отношению к нему, то можно сократить полные имена, объявив контекст текущим с помощью оператора using. Если элементы пространства имен будут интенсивно использоваться, то можно применить ключевое слово using для упрощения доступа к ним. Ключевое слово using используется и как директива, и для объявления. Синтаксис слова using определяет, является ли оно директивой или объявлением.

Ключевое слово using как директива

Директива using namespace имя позволяет предоставить все имена, объявленные в пространстве имен, для доступа в текущей области действия. Это дает возможность обращаться к этим именам без указания их полного имени, включающего название пространства имен:

```
#include <iostream.h>
namespace NAME
{
        int n1=1;
        int n2=2;
}
// int n1; приводит к неоднозначности в main для переменной n1
int main()
{
        NAME::n1=3;
        //          n1=3;           // error 'n1': undeclared identifier
        // n2=4;                    // error 'n2': undeclared identifier
        using namespace NAME;// далее n1 и n2 доступны
        n2=4;
}
```

Ключевое слово using как объявление

Объявление using имя::член подобно директиве, при этом оно обеспечивает применение нескольких уровней управления. Обычно using используется для объявления некоторого имени (из пространства имен) как принадлежащего текущей области действия (блоку).

6.4.5 Классы и объекты

Основная идея введения классов заключается в том, чтобы предоставить программисту средства для создания новых типов данных, которые могут использоваться так же, как и встроенные типы. Класс - это абстракция, точнее, это тип, определяемый пользователем. Например, мы можем задать структуру с именем point, которая содержит координаты ***x чу*** точки на экране дисплея:

```
struct point
{
  int x,y;
};
```

Пусть нам нужны две функции, которые позволяют нарисовать точку set_pixel() и прочитать ее координаты get_pixel():

```
void set_pixel(int, int);
void get_pixel(int *, int *);
```

В нашем примере данные и функции, работающие с этими данными, отделены друг от друга. Связь между ними можно установить, если задать структуру в виде:

```
struct point
{
  int x,y;
  void set_pixel(int, int);
  void get_pixel(int *, int *);
};
```

Данные x и y, объявленные в приведенной структуре, называются компонентами-данными, а функции - компонентами-функциями или методами. Объект объявляется следующим образом:

```
имя_класса имя_объекта;
```

Теперь мы можем обратиться к данным и вызвать функции, только указав имя объекта, к которому они принадлежат. Для этих целей можно использовать те же операции точка(.) и ->. Рассмотрим пример:

```
struct point
{
  int x,y;
  void set_pixel(int, int);
  void get_pixel(int *, int *);
};
int a=50,b=100;
point my_point, *pointer;
pointer=&my_point;
```

```
my_point.set_pixel(50,100);
pointer->get_pixel(&a,&b);
```

Поскольку различные структуры могут иметь функции с одинаковыми именами, при описании функции необходимо указывать, для какой структуры она описывается:

```
void point::set_pixel(int x, int y)
{
        тело функции
}
```

Синтаксис описания функции, принадлежащей структуре, имеет следующий вид (рисунок 6.9):

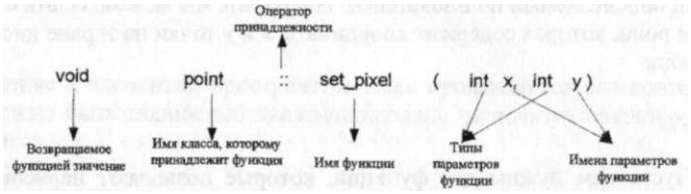


Рисунок 6.9 - Синтаксис описания функции, принадлежащей структуре

Оператор принадлежности :: иначе называется оператор разрешения области видимости. Этот оператор самого высокого приоритета.

```
::i          // унарный оператор :: указывает на внешнюю область видимости
point::i   //  бинарный оператор :: указывает на область видимости класса.
```

В языке C++ структура - это тоже класс. С другой стороны, мы можем записать вместо ключевого слова struct ключевое слово class, например:

```
class point
{
     int x,y;
     void set_pixel(int, int);
     void get_pixel(int *, int *);
};
```

В языке C++ класс, определяемый посредством ключевых слов struct, class, union, включает в себя функции и данные, создавая новый тип объектов. Компоненты класса имеют ограничения на доступ. Эти ограничения определяются ключевыми словами private, protected, public. Для ключевого слова class по умолчанию все компоненты будут private. Это означает, что они (их имена) будут недоступны для использования вне компонентов класса. Ограничения доступа для некоторого компонента можно изменить, записав перед ним атрибут модификации доступа - ключевое слово public или protected и двоеточие. Таким образом, упрощенную форму описания класса можно записать в виде:

```
class имя_класса
{
          данные и функции с атрибутом private (по умолчанию)
       protected:
```

данные и функции с атрибутом protected
public:
данные и функции с атрибутом public
} объекты этого класса через запятую.

Обычно ограничения на уровень доступа касаются элементов данных: данные имеют атрибут private или protected, а методы - public.

Смысл атрибутов доступа следующий:

private - член класса с атрибутом private может использоваться только методами собственного класса и функциями-«друзьями» этого же класса; по умолчанию все члены класса, объявленного с ключевым словом class, имеют атрибут доступа private;

protected - то же, что и private, но дополнительно член класса может использоваться методами и функциями-«друзьями» производного класса, для которого данный класс является базовым;

public - член класса может использоваться любой функцией программы, т.е. защита на доступ снимается.

Явно ограничения на доступ могут переопределяться записью атрибута перед компонентами класса. Элементы класса типа структуры (struct) и объединения (union) по умолчанию принимаются как public. Для ключевого слова struct атрибут можно явно переопределить на private или protected. Для ключевого слова union явное переопределение атрибута доступа невозможно. Класс или структура может содержать любое количество секций с заданными атрибутами. Секция начинается с ключевого слова и двоеточия после него. Секция заканчивается в конце описания структуры (класса) или началом другой секции.

Пример:

```
#include<iostream>
using namespace std;
#include<string.h>
class String
{
        char str[25];          // атрибут доступа private
    public:
        void set_string(char *);
        void display_string(void);
        char * return_string(void);
};
void String::set_string(char *s)
{
        strcpy(str,s);        // копирование s в str
}
void String::display_string(void)
{
       cout<<str<<endl;
}
char *  String::return_string(void)
{       return str;    }
```

```
void main(void)
{
        String      str1;      // объявление объекта
        str1.set_string("Минск");
        strl.display_string();
        cout<<strl.return_string()<<endl;
}
```

6.4.6 Конструкторы

Использование функций для установки начальных значений данных объекта неестественно и часто приводит к ошибкам. В связи с этим введена специальная функция, позволяющая инициализировать объект в процессе его объявления. Эта функция называется конструктором. Функция конструктор имеет то же имя, что и соответствующий класс, например:

```
class String
{
        char str[25];          // атрибут доступа private
    public:
        String(char *s)     // конструктор
        {
           strcpy(str,s);
        }
};
```

Конструктор может иметь и не иметь аргументов, и он никогда не возвращает значения (даже типа void). Класс может иметь несколько конструкторов, что позволяет использовать несколько различных способов инициализации соответствующих объектов. Иначе говоря, конструктор является функцией, а значит, он может быть перегружен. Конструктор вызывается, когда связанный с ним тип используется в определении. Пример:

```
#include <iostream>
using namespace std;
class Over
{       int i;
        char *str;
      public:
        Over()
        {
           str="Первый конструктор";
           i=0;
        }
        Over(char *s)        // Второй конструктор
        {
           str=s;
           i=50;
        }
```

```
        Over(char *s, int x)  // Третий конструктор

           str=s;
           i=x;

        Over(int *y)

           str="Четвертый конструктор\n";
           i=*y;

        void print(void)

           cout<<"i="<<i<<"str="<<str<<endl;

};
void main(void)
{
   int a=10, *b;
   b=&a;
   Over my_over;
   Over my_over1("Для конструктора с одним параметром");
   Over my_over2("Для конструктора с двумя параметрами, 100);
   Over my_over3(b);          // Для четвертого конструктора
   my_over.print();           // Активен конструктор Over()
   my_over1.print();          // Активен конструктор Over(char *s)
   my_over2.print();          // Активен конструктор Over(char *s, int x)
   my_over3.print();          // Активен конструктор Over(int *y)
}
```

Результаты выполнения программы представляются в виде:

```
i=0;    str= Первый конструктор
i=50;   str= Для конструктора с одним параметром
i=100;  str= Для конструктора с двумя параметрами
i= 10;  str= Четвертый конструктор
```

Конструктор можно записывать и в следующем виде:

```
class my_class
{
        int a,b;
     public:
        my_class(int A, int B): a(A),b(B) {}
};
```

После выражения **my_class(int** A, **int** B) записано двоеточие и затем через запятую перечислены все компоненты-данные класса **my_class,** которым необходимо присвоить значения, т.е. компоненту а необходимо присвоить значение А, компоненту **b** - значение В. После этого в теле конструктора не надо выполнять никаких действий, что записывается в виде {}. Другими словами, такая форма эквивалентна следующему описанию:

```
my_class(int A, int B)
{
        a=A;
        b=B;
}
```

Конструктор имеет следующие отличительные особенности:

- всегда выполняется при создании нового объекта, т.е. когда под объект отводится память и он инициализируется;
- может определяться пользователем или создаваться по умолчанию;
- не может быть вызван явно из пределов программы (не может быть вызван как обычный метод). Он вызывается компилятором явно при создании объекта и неявно - при выполнении оператора new для выделения памяти объекту;
- всегда имеет то же имя, что и класс, в котором он определен;
- никогда не должен возвращать значения;
- не наследуется.

6.4.7 Копирующий конструктор

Рассмотрим следующую программу:

```
class Massiv
{       int *mas;                   // указатель на массив
        int n;                      // количество элементов массива
      public:
        Massiv(int n1=0);           // конструктор с параметрами по умолчанию
        void vvod();                // функция ввода значений массива
        void display();             // функция вывода значений массива
        int fun(Massiv ob);         // функция вычисления суммы элементов массива
         ~Massiv();                 // деструктор
  };
Massiv::Massiv(int     n1)      // определение конструктора
{
        n=n1;
        mas=new int[n];
}
Massiv::~Massiv()                  // определение деструктора
{
        delete [] mas;
}
void Massiv::vvod()                // определение функции ввода значений массива
{
        fot(int i=0; i < n; i++)
                cin>>mas[i];
}
void Massiv::display()          // определение функции вывода значений массива
{       for(int i=0; i < n; i++)
```

```
            cout<<setw(4)<<mas[i];
        cout<<endl;
}
int Massiv:: fun(Massiv ob)     // функция вычисления суммы элементов массива
{
        int sum=0:
        for(int i=0; i < n; i++)
            sum+=ob.mas[i];
        return sum;
}
void  main(void)              // головная функция
{       int summa;
        int n;                  // размер массива
        cout<<"Введите размер массива"<<endl;
        cin>>n;
        Massiv ob(n);         // объявление объекта
        cout<<"Введите элементы массива"<<endl;
        ob.vvod();              // вызов функции vvod
        cout<<"Исходный массив"<<endl;
        ob.display();           // вызов функции display
        summa=ob.fun(ob); // вызов функции fun
        cout<<"сумма элементов массива"<<setw(4)<<summa<<endl;
        ob.display();           // при вызове функции display() возникает ошибка
}
```

В функцию fun() передается значение объекта типа Massiv. Даже если вызванная функция ничего не будет выполнять, произойдет ошибка, связанная с динамическим выделением и освобождением памяти. Параметр в функции fun() является локальным (автоматическим объектом) в теле этой функции. Любой автоматический объект конструируется тогда, когда встречается его объявление, и разрушается, когда блок, в котором он описан, прекращает существование. После завершения функция прекращает существовать, в результате вызывается деструктор объекта ob.

В функции main() выполняются следующие действия:

- описание Massiv ob(n); задает конструирование нового объекта ob. Конструктор объекта ob выделяет (динамически) память под массив mas с помощью оператора new;
- вызывается функция fun(ob);
- значение объекта ob копируется из функции main в стек функции fun();
- копия объекта ob содержит указатель на ту же динамическую память (указатель на динамическую память в объекте-оригинале и объекте-копии имеет одинаковые значения);
- функция fun() завершается;
- вызывается деструктор для копии объекта ob, который разрушает динамически вьделенную память под массив mas;
- теперь указатель в оригинале объекта адресует несуществующую удаленную память.

Если добавить функцию fun() в виде fun(Massiv& ob);, то ошибка будет устранена. При необходимости можно оставить и предыдущее объявление функции т.е. fun(Massiv ob). В этом случае надо устранить ошибку в самом классе Massiv. Когда объект ob копируется из функции main() в функцию fun(), то должен вызывается конструктор для копирования. Так как в нашем классе такого конструктора нет, то вызывается конструктор, заданный по умолчанию. Этот конструктор строит точную копию всех данных объекта ob, что и приводит к ошибке. Если в классе Massiv задать явно конструктор для копирования, например:

```
Massiv(const Massiv& ob)
{
        n=ob.n;
        mas=new int[n];
        for(int i=0; i < n; i++)
                mas[i]=ob.mas[i];
}
```

то ошибка будет устранена. Таким образом, если в конструкторе некоторого класса Massiv осуществляется динамическое выделение памяти, такой класс должен иметь соответствующий конструктор для копирования, а также деструктор (для освобождения памяти).

С использованием конструктора копирования, программа будет следующая:

```
class Massiv
{
        int *mas;              // указатель на массив
        int n;                 // количество элементов массива
      public:
        Massiv(int n1);      // конструктор с параметрами
        Massiv(const Massiv& ob); // конструктор копирования
        void vvod();          // функция ввода значений массива
        void    display();    // функция вывода значений массива
        int fun(Massiv ob); // функция вычисления суммы элементов массива
         ~Massiv();
};
Massiv::Massiv(int   n1)   // определение конструктора с параметрами
{
        n=n1;
        mas=new int[n];
}
Massiv::Massiv(const Massiv& ob) // определение конструктора копирования
{
        n=ob.n;
        mas=new int[n];
        for(int i=0; i < n; i++)
                mas[i]=ob.mas[i];
}
```

```
Massiv::~Massiv()          // определение деструктора
{
      delete [] mas;
}
void Massiv::vvod()      // определение функции ввода значений массива
{
      fot(int i=0; i < n;
            cin>>mas[i];
}
void Massiv::display()   // определение функции вывода значений массива
{
      fot(int i=0; i < n; i++)
            cout<<setw(4)<<mas[i];
      cout<<endl;
}
int Massiv:: fun(Massiv ob)    // функция вычисления суммы элементов массива
{
      int sum=0;
      for(int i=0; i < n; i++)
            sum+=ob.mas[i];
      return sum;
}
void  main(void)              // головная функция
{      int summa;
      int n;                  // размер массива
      cout<<"Введите размер массива"<<endl;
      cin>>n;
      Massiv ob(n);          // объявление объекта
      cout<<"Введите элементы массива"<<endl;
      ob.vvod();              // вызов функции vvod
      cout<<"Исходный массив"<<endl;
      ob.display();           // вызов функции display
      summa=ob.fun(ob);  // вызов функции fun и автоматически вызывается
                              // конструктор копирования
      cout<<"сумма элементов массива"<<setw(4)<<summa<<endl;
      ob.display();           // вызов функции display()
}
```

6.4.8 **Деструктор**

Противоположные действия, по отношению к действиям конструктора, выполняют функции-деструкторы(destructor), или разрушители, которые уничтожают объект. Деструктор может вызываться явно или неявно. Неявный вызов деструктора связан с прекращением существования объекта из-за завершения области его определения. Явное уничтожение объекта выполняет оператор delete. Деструктор имеет то же имя, что и класс, но перед именем записывается

знак тильда ~. Кроме того, деструктор не может иметь аргументов, возвращать значение и наследоваться.

Пример:

```
class String
{
      int i;
    public:
      String(int j);      // объявление конструктора
      ~String();          // объявление деструктора
      void show_i(void);
};                        // конец объявления класса
String::String(int j)     // определение конструктора
{
      i=j;
      cout<<"Работает конструктор"<<endl;
}
void String::show_i(void)   // определение функции
{
      cout<<"i="<<i<<endl;
}
String::~String()           // определение деструктора
{
      cout<<"Работает деструктор"<<endl;
}
void  main(void)
{     String  my_ob1(25);  // инициализация объекта my_obl
      String my_ob2(36);   // инициализация объекта my_ob2
      my_ob1.show_i();     // вызов функции show_i() класса String для my_obl
      my_ob2.show_i();     // вызов функции show_i() класса String для my_ob2
}
```

Результаты работы программы следующие:

```
Работает конструктор
Работает конструктор
i=25
i=36
Работает деструктор
Работает деструктор
```

Деструкторы выполняются в обратной последовательности по отношению к конструкторам. Первым разрушается объект, созданный последним.

6.4.9 Дружественные функции.

В некоторых случаях необходимо иметь доступ из одной функции к локальным компонентам разных классов (к компонентам с атрибутом private). Рассмотрим пример. Пусть заданы два графических объекта: окружность и прямоугольник. Некоторая функция (назовем ее color) должна сравнивать цвета

этих объектов и возвращать результат, свидетельствующий о совпадении или несовпадении цветов. Цвет окружности и прямоугольника задается локальными компонентами соответствующих объектов. В этом случае функция color() должна иметь доступ к локальным компонентам разных классов (рисунок 6.10).

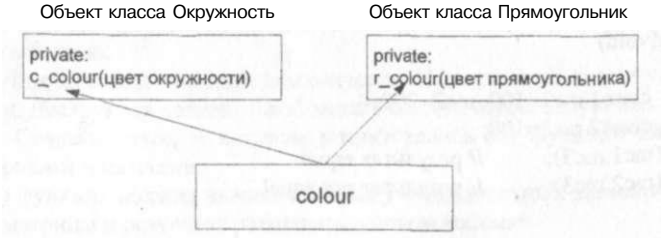


Рисунок 6.10 - Дружественная функция color()

Для этого в языке C++ введен спецификатор friend. Если некоторая функция определена как friend-функция для некоторого класса X, то:

- она не является членом этого класса;
- имеет доступ ко всем компонентам этого класса (т.е. к компонентам с атрибутами private, public, protected).

Функция является friend-функцией, если она объявлена со спецификатором friend. В целом функции со спецификатором friend являются обычными глобальными. Объявление функции со спецификатором friend в пределах класса X является действительно объявлением. Оно вводит имя функции в границы класса X. Это объявление можно поместить в любую секцию класса (public, private, protected). В любом случае friend-функция является глобальной и неважно, в какой секции она объявлена.

Пример:

```
#include<iostream>
using namespace std;
class my_class2;              // предварительное объявление класса
class my_class 1
{
      int a;
       friend void  fun(my_class1&,  my_class2&);
     public:
       my_class1(int  A):a(A) {}      //конструктор my_class1
};
class my_class2
{
      int a;
      friend void  fun(my_class1&,  my_class2&);
     public:
      my_class2(int A):a(A) {}       //конструктор my_class2
};
```

```
void fun(my_class1& M1, my_class2& M2)
{
        if(M1.a=M2.a)) cout<<"equal\n";
        else cout<<"not equal";
}
void main(void)
{
        my_class1  mc1=100, mc2=200;
        my_class2  mc3=100;
        fun(mc1 ,mc3);          // результат equal
        fun(mc2,mc3);           // результат not equal
}
```

6.4.10 Лабораторная работа №3 Классы и объекты

Цель: изучить принцип работы функции *конструктор* с параметрами и с параметрами по умолчанию, перегрузку конструкторов. Использовать функцию *деструктор* в разработанных программах.

Варианты заданий

1 Создать класс, в котором реализовать функции для работы с матрицами:

а) функция производит перемножение матриц;

б) функция производит сложение двух матриц.

Память под матрицы отводить динамически. Использовать конструктор с параметрами. Деструктор должен освобождать память, выделенную под матрицы.

2 Создать класс, в котором нужно реализовать функции для работы с одномерными массивами:

а) получить пересечение элементов массивов;

б) получить объединение элементов массивов.

Память под массивы отводить динамически. Использовать конструктор с параметрами. Деструктор должен освобождать память, выделенную под массивы.

3 Создать класс, в котором надо реализовать функции для работы с двухмерными массивами:

а) функция находит минимальный элемент ниже главной диагонали;

б) функция находит максимальный элемент выше главной диагонали.

Память под массивы отводить динамически. Использовать конструктор с параметрами. Деструктор должен освобождать память, выделенную под массивы.

4 Создать класс, в котором требуется реализовать функции для работы с матрицами:

а) функция находит минимальное число в матрице;

б) функция находит максимальное число в матрице.

Память под матрицы отводить динамически. Использовать конструктор с параметрами. Деструктор должен освобождать память, выделенную под матрицы.

5 Создать класс, в котором реализуются функции для работы с одномерными массивами:

а) функция должна найти максимальное число в одном массиве;

б) функция должна найти минимальное число во втором массиве;

в) функция должна поменять местами минимальное и максимальное значения в массивах.

Память под массивы отводить динамически. Использовать конструктор с параметрами. Деструктор должен освобождать память, выделенную под массивы.

6 Создать класс, в котором реализовались бы функции для работы с двухмерными массивами:

а) функция должна вычислять сумму отрицательных элементов в каждой строке матрицы и результат размещать в новом массиве.

Память под массивы отводить динамически. Использовать конструктор с параметрами. Деструктор должен освобождать память, выделенную под массивы.

7 Создать класс, в котором нужно реализовать функции для работы с двухмерными массивами:

а) функция должна вычислять сумму положительных элементов в каждом столбце матрицы и результат размещать в новом массиве.

Память под массивы отводить динамически. Использовать конструктор с параметрами. Деструктор должен освобождать память, выделенную под массивы.

8 Создать класс, в котором требуется реализовать функции для работы с двухмерными массивами:

а) функция должна рассортировать столбцы матрицы по возрастанию.

Память под массивы отводить динамически. Использовать конструктор с параметрами. Деструктор должен освобождать память, выделенную под массивы.

9 Создать класс, в котором реализуются функции для работы с двухмерными массивами:

а) функция должна вычислять сумму элементов строк после первого встретившегося отрицательного числа. Результат сохранить в новом массиве.

Память под массивы отводить динамически. Использовать конструктор с параметрами. Деструктор должен освобождать память, выделенную под массивы.

10 Создать класс, в котором надо реализовать функции для работы с двухмерными массивами:

а) функция должна вычислять сумму элементов строк матрицы, которые начинаются с отрицательного элемента. Результат записать в новый массив.

Память под массивы отводить динамически. Использовать конструктор с параметрами. Деструктор должен освобождать память, выделенную под массивы.

6.5 Производные классы

6.5.1 Наследование

Наследование - это механизм получения нового класса из существующего. Существующий класс может быть дополнен или изменен для создания производного класса. При создании нового класса вместо написания полностью новых данных и функций программист может указать, что новый класс должен

наследовать данные и функции ранее определенного базового класса. Этот новый класс называется производным. Каждый производный класс сам является кандидатом на роль базового для будущих производных классов. При простом наследовании класс порождается одним базовым классом. При множественном наследовании производный класс наследуется несколькими базовыми классами. Производный класс обычно добавляет свои данные и функции, так что производный класс в общем случае больше своего базового.

Шаблон объявления производного класса можно представить следующим образом:

ключ_класса имя_производного_класса:необязательный_модификатор_доступа имя_базового_класса
{ Тело производного класса };

Пример:

```
class Location
{
      int x,y;
    public:

};
class Point:public Location
{
      Тело класса Point
};
```

Класс Location является базовым и наследуется с атрибутом public. Класс Point - производный. Двоеточие (:) отделяет производный класс от базового. Атрибут класса (модификатор прав доступа) может задаваться ключевыми словами public и private. Атрибут может опускаться - в этом случае принимается атрибут по умолчанию (для ключевого слова class - private, для struct - public). Объединение (union) не может быть базовым или производным классом.

Модификатор прав доступа используется для изменения прав доступа к наследуемым элементам класса в соответствии с правилами, приведенными в таблице 1.

Таблица **1**

Оганичения на доступ в базовом классе	Модификатор наследования прав	Ограничения на доступ в производном классе
private	private	Нет доступа
protected	private	private
public	private	private
private	public	Нет доступа
protected	public	protected
public	public	public

Отметим, что в производных классах права на доступ к элементам базовых классов не могут быть расширены, а только ограничены. Пример:

```
class A
{     int a1;
```

```
    public:
        int a2;
        void f1(void);
};
class B:A
{
        int b1;
    public:
        void f1(void)
        {
            a1=1;  // ошибка, a1 - private - переменная класса A, доступна только
                   // для методов и дружественных функций собственного класса
            b1=0;  // доступ к переменной типа private из метода класса
            a2=1;  // a2 унаследована из класса A с атрибутом доступа private и
                   // поэтому доступна в методе класса
        }
};
void main(void)
{
        A     a_ob1;     // объявление объекта a_ob1 класса A
        B     b_ob1;     //объявление объекта b_ob1 класса B
        b_ob1.a2+=1;   // ошибка, так как a2 private
        a_ob1.a2+= 1; // допустимая операция
}
```

Рассмотрим еще пример, демонстрирующий наследование прав доступа к элементам базовых классов.

```
#include<iostream>
using namespace std;
#include<string.h>
#define N 10
class book
{
    protected:
      char naz[20];            // название книги
      int          k1;         // количество страниц
    public:
      book(char *, int);       // конструктор класса book
      ~book();                 // деструктор класса book
};
class avt:public book
{     char fm[10];             // фамилия автора
    public:
      avt(char *, int, char *); // конструктор класса avt
      ~avt();                  // деструктор класса avt
      void see();
};
```

```
enum razd{teh, hyd, uch};
class rzd: public book
{       razd rz;
      public:
        rzd(char *, int, razd);  // конструктор класса rzd
        ~rzd();                  // деструктор класса rzd
        void see();
};
book::book(char *s1, int i)
{
        cout<<"\n Конструктор класса book";
        strcpy(naz,s1);
        k1=i;
}
book::~book()
{
        cout<<"\nДеструктор класса book";
}
avt::avt(char *s1, int i, char *s2):book(s1,i)
{
        cout<<"\n  Конструктор класса avt";
        strcpy(fm,s2);
}
 avt::~avt()
{    cout<<"\nДеструктор    класса avt";
}
void  avt::see()
{
        cout<<"\nНазвание книги"<<naz<<endl<<"\nКоЛичество страниц"<<k1;
}
rzd::rzd(char *s1, int i, razd tp):book(sl,i)
{
        cout<<"\n Конструктор класса rzd";
        rz=tp;
}
rzd::~rzd()
{
        cout<<"Деструктор класса rzd";
}
void  rzd::see()
{        switch(rz)
        {       case teh: cout<<"\n Раздел технической литературы"; break;
                case hyd: cout<<"\nРаздел художественной литературы"; break;
                case uch: cout<<"\n Раздел учебной литературы"; break;
        }
}
```

```
void main(void)
{
        avt av("Книга 1", 123, "автор 1");
        rzd rz("Книга 1", 123, teh);
        av.see();
        rz.see();
}
```

Если базовый класс имеет конструктор с одним или более аргументами, то и любой производный класс должен иметь конструктор. В нашем примере конструктор класса book задан в виде:

```
book::book(char *s1, int i)
{    cout<"\nКонструктор    класса book";
        strcpy(naz,s1);
        kl=i;
}
```

Теперь объявление объекта в функции main (либо в другой функции) может осуществляться book my_ob("Дейтел", 1113);

В соответствии со сделанными выше замечаниями производный класс avt тоже должен иметь конструктор. В нашем примере он задан следующим образом:

```
avt::avt(char *s1, int i, char *s2):book(s1,i)
{
        cout<<"\n  Конструктор класса avt";
        strcpy(fm,s2);
}
```

6.5.2 Множественное наследование

При множественном наследовании производный класс образуется из нескольких базовых. В языке C++ допускается образовывать производный класс от нескольких базовых классов. Поэтому общий синтаксис описания конструктора производного класса представляется в виде:

имя_конструктора_производного_класса(список аргументов для производного класса): имя_базового_класса1(список аргументов для констр. базового класса1),..., имя_базового_классаN(список аргументов для базового классаN)

```
{
      тело конструктора
}
```

Пример:

```
#include<iostream>
using namespace std;
class        Base_1        // базовый класс Base_1
{
        int a;
     protected:
        int b;
      public:
```

```
        Base_1(int x, int y);
        ~Base_1();
        void show1(void)
        {
              cout<<"a="<<a<<"b="<<b;
        }
};
class Base_2                // базовый класс Base_2
{
      protected:
        int c;
      public:
        Base_2(int r);
        ~Base_2();
        void show2(void)
        {
              cout<<"c="<<c<<endl;
        }
};
class Derive: public Base_1, public Base_2 // производный класс Derive
{       int p;
      public:
        Derive(int x, int y, int z);
        ~Derive();
        void show3(void)
        {
              cout<<"a+b+c="<<p+b+c<<endl;
        }
};
Base_l::Base_1(int  x, int y)
{
       a=x;
       b=y;
       cout<<"\n Конструктор Base_1";
}
Base_1::~Base_1()
{
       cout<<"\nДеструктор Base_1";
}
 Base_2::Base_2(int  x)
{
       c=x;
       cout<<"\n Конструктор Base_2";
}
```

```
Base_2::~Base_2()
{
      cout« "\nДеструктор Base_2";
J
Derive::Derive(int x, int y, int z):Base_1(x,y), Base_2(z)
{
      p=x;
      cout<<"\n Конструктор Derive";
}
void main(void)
{
      int x,y,z;
      cout<<"\nВведите значения x, y, z";
      cin>>x>>y>>z;       // ввод значений переменных
      Derive my_d(x,y,z);       // объявление объекта
      my_d.show1();
      my_d.show2();
      my_d.show3();
}
```

Каждый класс, Base_1 и Base_2, содержит свой конструктор с параметрами. В связи с этим производный класс тоже должен содержать свой конструктор с параметрами. В функции main() объявлен объект my_d. В результате будут вызваны конструкторы базовых и производного классов. Конструктор производного класса -

```
Derive::Derive(int x, int y, int z):Base_1(x,y), Base_2(z)
{ ... }
```

Последовательно вызываются конструкторы базовых классов в соответствии с их появлением в списке, т.е. сначала вызывается конструктор Base_1, затем конструктор Base_2, после них - конструктор производного класса Derive. Выполнение деструкторов осуществляется в обратном порядке. Исходя из сказанного, результаты работы программы представляются в следующем виде:

```
Конструктор Base_1
Конструктор Base_2
Конструктор Derive
a=3 b=5 c=7
a+b+c=15
Деструктор Derive
Деструктор Base_2
Деструктор Base_1
```

6.5.3 Виртуальные функции

В языке C++ полиморфизм реализуется посредством виртуальных функций. Виртуальная функция - это функция, объявленная с ключевым словом

virtual в базовом классе и переопределенная в одном или нескольких производных этого класса. Объявление: virtual void print(void); говорит о том, что функция print может быть различной для базового и разных производных классов. В производных классах функция может иметь список параметров, отличный от параметров виртуальной функции базового класса. В этом случае эта функция будет не виртуальной, а перегруженной. Механизм вызова виртуальных функций можно поясните следующим образом. При создании нового объекта для него выделяется память. Для виртуальных функций (и только для них) создается указатель на таблицу функций, из которой выбирается требуемая функция в процессе выполнения. Если в некотором классе задана хотя бы одна виртуальная функция, то все объекты этого класса содержат указатель на связанную с их классом виртуальную таблицу. Эта таблица содержит адреса (указатели на первые инструкции) действительных функций, которые будут вызваны. Доступ к виртуальной функции осуществляется через этот указатель и соответствующую таблицу (т.е. осуществляется косвенный вызов функции). Если функция вызвана с использованием ее полного имени, то виртуальный механизм игнорируется. Свойство виртуальности проявляется только тогда, когда обращение к функции идет через указатель или ссылку на объект. Указатель или ссылка могут указывать как на объект базового, так и на объект производного класса.

Рассмотрим пример использования виртуальной функции:

```
#include<iostream>
#include<iomanip>
#include<string.h>
using namespace std;
class base              // базовый класс
{
 public:
        virtual char* name()
        {
                return "noname";
        }
        virtual double area()
        {       return 0;     }
};
class rect: public base       // производный класс "прямоугольник"
{
        int h,s;                  // размеры прямоугольника
public:
        rect(int H, int S)     // конструктор
        {       h=H;
                s=S;
        }
        virtual char* name()  // вывод на экран названия фигуры
        {
                return "прямоугольник";
        }
```

```
        double area()          // вычисление площади фигуры
        {
              return h*s;
        }
};
class circl: public base      // производный класс "окружность"

        int           r;          // радиус окружности
public:
        circl(int R)              // конструктор
        {     r=R;    }
        virtual char* name()// вывод на экран названия фигуры
        {
              return "круг";
        }
        double     area()     // вычисление площади фигуры
        {
              return 3.14*r*r;
        }
};
int main()
{
        base *p[2];           //  массив указателей на базовый класс
        rect obj1(3,4);
        circl obj2(5);
        p[0]=&obj1;
        p[1]=&obj2;
        for(int i=0; i < 2; i++)
            cout<<"площадь"<<p[i]->name()<<setw(10)<<p[i]->area()<<endl;
}
```

Массив указателей p хранит адреса объектов производных классов и необходим для вызова виртуальных функций этих классов. Если функции name() и area() в базовом классе объявлены как virtual и мы вызываем эти функции через указатель базового класса, указывающий на объекты производных классов, то программа будет динамически (т.е. во время выполнения программы) выбирать соответствующие функции name() и area() производного класса. Это называется динамическим связыванием (dynamic binding). Когда виртуальная функция вызывается путем обращения к заданному объекту по имени и при этом используется операция доступа к элементу "точка", тогда эта ссылка обрабатывается во время компиляции и это называется статическим связыванием.

Если функция была объявлена как виртуальная в некотором классе (базовом классе), то она остается виртуальной независимо от количества уровней в иерархии классов, через которые она прошла.

Приведем основные правила использования виртуальных функций:

- виртуальный механизм поддерживает полиморфизм на этапе выполнения программы. Это значит, что требуемая версия программы выбирается на этапе выполнения программы, а не компиляции;
- класс, содержащий хотя бы одну виртуальную функцию, называется полиморфным;
- виртуальные функции можно объявлять только в классах (class) и структурах (struct);
- виртуальными функциями могут быть только нестатические функции (без спецификатора static), так как характеристика virtual наследуется. Функция производного класса автоматически становится virtual;
- виртуальные функции можно объявлять со спецификатором friend для другого класса;
- виртуальными функциями могут быть только неглобальные функции (т.е. компоненты класса);
- если виртуальная функция объявлена в производном классе со спецификатором virtual, то можно рассматривать новые версии этой функции в классах, наследуемых из этого производного класса;
- для вызова виртуальной функции требуется больше времени, чем для невиртуальной. При этом также требуется дополнительная память для хранения виртуальной таблицы;
- при использовании полного имени при вызове виртуальной функции виртуальный механизм не поддерживается.

Приведем пример использования виртуальной функции:

```
#include<iostream>
#include<iomanip>
using namespace std:
#include<string.h>
class grup                  // базовый класс
{
   protected:
      char *spec;      // название специальности
      long gr;         // номер группы
   public:
      grup(char *SPEC, long GR) // конструктор
      {       spec=new char[40];
              strcpy(spec,SPEC);
              gr=GR;

      ~grup()          // деструктор

              cout<<"деструктор  класса grup"<<endl;
              delete [] spec;

      virtual void print(void); // объявление виртуальной функции
```

```
class asp: public grup            // производный класс
{       char *fam:                // фамилия
        int oc[4];                // массив оценок
   public:
        asp(char *SPEC, long GR, char *FAM, int OC[]):
        grup(SPEC,GR)       // конструктор производного класса
        {       fam=new char[30];
                strcpy(fam,FAM);
                for(int i=0; i < 4; i++)
                        oc[i]=OC[i];
        }
        ~asp()
        {
                cout<<"Деструктор  класса asp"<<endl;
                delete [] fam;
        }
        void print(void);
};
void grup::print(void)          // определение виртуальной функции
{
        cout<<"Специальность"<<setw(10)<<"Группа"<<endl;
        cout<<setw(8)<<spec<<setw(10)<<gr<<endl;
}
void asp::print(void)
{
        grup::print();            // вызов функции базового класса
        cout<<"ФИО"<<setw(10)<<fam;
        cout<<"Оценки";
        for(int i=0; i < 4; i++)
                cout<<setw(4)<<oc[i];
        cout<<endl;
}
void  main()
{       int OC[]={4,5,5,4};   // массив оценок
        char SP[40];
        long GR;
        char FAM[40];
        cout<<"Введите название специальности"<<endl;
        gets(SP);
        cout<<"Введите номер группы"<<endl;
        cin>>GR;
        cout<<"Введите фамилию"<<endl;
        fflush(stdin);
        gets(FAM);
        grup ob1(SP,GR), *p;
        asp ob2(SP,GR,FAM,OC);
```

```
    cout<<"Результат"<<endl;
    //p=&ob1;   // указатель на объект базового класса
    //p->print(); // вызов функции базового класса
    p=&ob2;
    p->print();   // вызов функции производного класса
}
```

6.5.4 Абстрактные классы

Базовый класс обычно содержит ряд виртуальных функций, которые часто фиктивны и имеют пустое тело. Эти функции существуют как некоторая абстракция, конкретное значение им придается в производных классах. Такие функции называются чисто виртуальными (pure virtual function), т.е. такими, тело которых не определено. Общая форма записи чисто виртуальной функции имеет вид: virtual прототип функции = 0;

Если класс является производным класса с чисто виртуальной функцией и эта функция в нем не описана, тогда функция остается чисто виртуальной и в этом производном классе. Следовательно, такой производный класс является абстрактным. Хотя иерархия классов не требует обязательного включения в нее каких-либо абстрактных классов, однако программы, использующие объектно-ориентированное программирование, все же имеют иерархию, порожденную абстрактным базовым классом. Абстрактные классы могут составлять несколько уровней иерархии. В качестве примера можно привести иерархию форм. Иерархия может порождаться абстрактным базовым классом *Shape*. На уровень ниже можно получить два абстрактных класса *TwoDShape* и *ThreeDShape*. При переходе еще на один уровень ниже можно определить конкретные классы для двухмерных форм (Circle, Rectangle) и трехмерных (Cube, Sphere) (рисунок 6.11). Согласно языку UML имя абстрактного класса пишется курсивом.

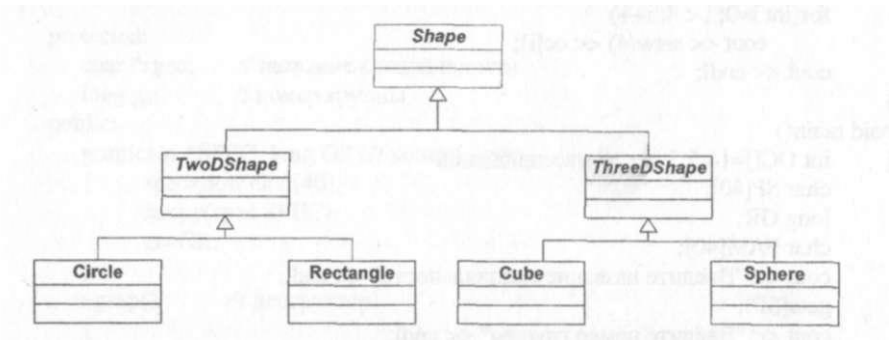


Рисунок 6.11 - Иерархия наследования классов

Рассмотрим пример программы приведенной иерархии классов, в которой будет выводиться название фигуры, площадь двухмерных и объем трехмерных фигур:

```
#include<iostream>
#include<iomanip>
```

```
using namespace std;
#include<string.h>
const float pi=3.14159;
class Shape                         // абстрактный базовый класс
{
   public:
       virtual void print()=0;    // печать названия фигуры
};
class TwoDShape: public Shape
{
  protected:
       float r;
  public:
       virtual void area() = 0;    // вычисление площади фигуры
};
class ThreeDShape: public Shape
{
  protected:
       float h;
  public:
       virtual void  volume();  // вычисление объема фигуры
};
class Circle: public TwoDShape       // класс "Окружность"
{
  public:
       void print()
       {
            cout<<"Окружность"<<endl;
       }
       void area()
       {   cout<<"Введите   радиус окружности"<<endl;
           cin>>r;
           cout<<"Площадь окружности"<<setw(10)<<pi*r*r<<endl;
       }
};
class Rectangle: public TwoDShape  // класс "Квадрат"
{
   public:
       void print()
       {   cout<<"Квадрат"<<endl;   }
       void area()
       {   cout<<"Введите   сторону квадрата"<<endl;
           cin>>r;
           cout<<"Площадь квадрата"<<setw(7)<<r*r<<endl;
       }
};
```

```
class Sphere: public ThreeDShape      // класс "Сфера"
{
   public:
      void print()
      {
            cout« "Сфера" « endl;
      }
      void volume()
      {
            cout<<"Введите радиус сферы"<<endl;
            cin>>h;
            cout<<"Объем сферы"<<setw(10)<<(4.0*pi*h*h*h)/3.0<<endl;
      }
};
class Cube: public ThreeDShape      // класс "Куб"
{
   public:
      void print()
      {
            cout<<"Куб"<<endl;
      }
      void volume()
      {   cout<<"Введите   сторону куба"<<endl;
            cin>>h;
            cout<<"Объем куба"<<h*h*h<<endl;
      }
};
int main()
{
      Shape *ptr;         // указатель на абстрактный базовый класс
      TwoDShape *ptr1;
      ThreeDShape *ptr2;
      Circle okr,
      Rectangle pr;
      Sphere sf;
      Cube kb;
// вызов функций класса "Окружность"
      ptr1=&okr;
      ptr1->print();
      ptr1->area();
// вызов функций класса "Квадрат"
      ptr1=&pr;
      ptr1->print();
      ptr1->area();
// вызов функций класса "Сфера"
      ptr2=&sf;
```

```
        ptr2->print();
        ptr2->volume();
// вызов функций класса "Куб"
        ptr2=&kb;
        ptr2->print();
        ptr2->volume();
}
```

6.5.5 Виртуальное наследование

При множественном наследовании возможно возникновение нескольких ситуаций, которые приводят к ошибке. Одна из таких ситуаций - конфликт-имен методов или атрибутов нескольких базовых классов, например:

```
class    Base_1    // 1-й базовый класс
{
   public:
        void prog()
        {
              cout<<"Базовый класс Base_1"<<endl;
        }
};
class Base_2        // 2-й базовый класс
{   public:
        void prog()
        {   cout<<"Базовый   класс   Base_2"<<endl;   }
};
class Derive: public Base_1, public Base_2
{       // выполнение каких-то действий      };
int main()
{       Derive *obj=new Derive; // указатель на производный класс
        //obj->prog();                //ошибка
        obj->Base_1::prog();
        obj->Base_2::prog();
        return 0;
}
```

Если функция prog() вызывается таким образом, то компилятор не может определить, функцию какого класса (Base_1 или Base_2) вызвать на выполнение. Ошибку можно устранить, если явно указать, какому из базовых классов принадлежит вызываемая функция:

```
obj->Base_1::prog();   или        obj->Base_2::prog();
```

Иная конфликтная ситуация возникает, если иерархия наследования классов следующая (рисунок 6.12):

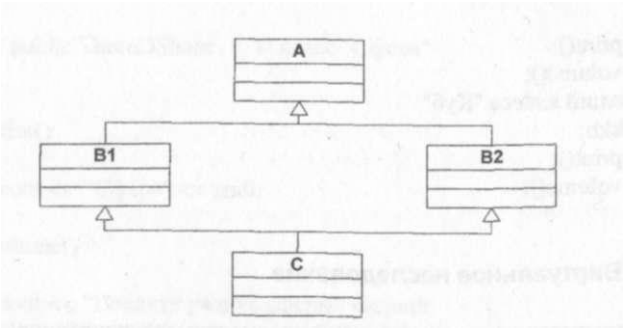


Рисунок 6.12 - Виртуальное наследование классов

Используя такую иерархию, классы B1, B2 и C можно объявить так:

```
class B1: public A
{   };
class B2: public A
{   };
class C: public B1, public B2
{   };
```

В этом случае производный класс C может содержать два одинаковых базовых класса A (A <- B1 <- C и A <- B2 <- C) и для каждого базового класса A строится свой объект (т.е. объекты-дубликаты). Такая ситуация будет неоднозначной и вызовет ошибку. Проблема объектов-дубликатов решается с помощью виртуального наследования. Если базовый класс наследуется как virtual, то только один объект базового класса A появляется в производном классе C. Этот процесс называется наследованием виртуального базового класса.

Пример программы виртуального наследования:

```
#include<iostream>
#include<iomanip>
using namespace std;
#include<string.h>
class A
{       char naz[20];              // название фирмы
   public:
       A(char *NAZ)           // конструктор
       {
              strcpy(naz,NAZ);
       }
       ~A()
       {   cout<<"Деструктор   класса    A"<<endl;    }
       void a_prnt()
       {
              cout<<"Фирма"<<setw(8)<<naz<<endl;
       }
};
```

```
class B1: virtual public A    // базовый класс A является виртуальным
{
   protected:
      long n_otd;             // номер сотрудника
      int  nom;               // номер отдела
   public:
      B1 (char *NAZ, long TN, int NOM): A(NAZ)
      {      n_otd=TN;
             nom=NOM;
      }
      ~B1()
      {
             cout<<"Деструктор класса B1"<<endl;
      }
      void b1_prnt()
      {
             A::a_prnt();
             cout<<setw(8)<<"Номер сотрудника"<<setw(4)<<n_otd<<endl
                 <<setw(10)<<"Номер отдела"<<nom<<endl;
      }
};
class B2: virtual public A        // базовый класс A является виртуальным
{
   protected:
      double zp;                  // заработная плата
   public:
      B2(char *NAZ, double ZP): A(NAZ) // конструктор
      {      zp=ZP;      }
      ~B2()
      {   cout<<"Деструктор   класса  B2"<<endl;  }
      void b2_pmt()
      {
             cout<<"Заработная плата"<<setw(8)<<zp<<endl;
      }
};
class C: public B1, public B2
{
       char *fam;           // фамилия сотрудника
    public:
//конструктор производного класса
       C(char *FAM, char *NAZ, long TN, int NOM, double ZP):
        B1(NAZ,TN,NOM),B2(NAZ,ZP),A(NAZ)
       {
             fam=new char[strlen(FAM)+1];
       }     strcpy(fam, FAM);
```

```
    ~C()
    {
        cout<<"Деструктор производного класса"<<endl;
    }
    void c_prnt()
    {    B1::b1_prnt();
        B2::b2_prnt();
        cout<<"фамилия"<<setw(8)<<fam<<endl;
    }
};
void main()
{
    C obj("Иванов","ИПНК", 10,5,555.5);
    C *pt;          // указатель на объект производного класса
    pt=&obj;
    pt->c_prnt(); // вызов функции производного класса
}
```

В приведенном примере при создании объекта obj конструктор класса A вызывается из конструктора класса C первым и только один раз. затем конструкторы классов B1 и B2 в том порядке, в котором они описаны в строке наследования классов: class C: public B1, public B2 .

6.5.6 **Лабораторная работа** №4 **Наследование**

Цель: изучить принципы наследования.

Варианты заданий

1 Реализовать базовый класс *Shape*. Создать производные *TwoDShape* и *ThreeDShape,* от которых унаследовать всевозможные конкретные формы (круг, прямоугольник, куб, цилиндр). Реализовать функции *print* (для вывода типа и размера объектов каждого класса), *area* (вычисление площади), *volume* (вычисление объема). Использовать конструктор с параметрами.

2 Создать базовый класс *A,* в котором есть переменные *a* и b, производный класс *B,* в этом классе есть переменная c, производный класс C (наследуется от класса *B),* в нем есть переменная/. Вычислить значение выражения:

$x=a^2 + b^2 + c/f.$

Использовать конструктор с параметрами.

3 В базовом классе *A* задать переменную *a*. От класса *A* получить производный класс *B,* в котором задать переменную b. От класса *B* получить производный класс C и в нем вычислить площадь прямоугольника. Использовать конструктор с параметрами.

4 Создать базовый класс *A,* в нем есть переменная *a*. От класса *A* получить производный класс *B,* в нем есть переменная b. От класса *B* получить производный класс C. В классе *C* вычислить корни квадратного уравнения. Использовать конструктор с параметрами.

5 В базовом классе *A* задана точка (т.е. ее координаты *x* и y). В производном классе *B* (наследуется от класса *A)* задаются координаты второй точки. Вычислить радиус и площадь круга. В производном классе C (наследуется от класса *A)* задается высота цилиндра и вычисляется его объем). Использовать конструктор с параметрами.

6 Создать базовый класс «Книга» включающий название книги, фамилию автора.

Реализовать производный класс «Отдел» включающий в себя название отдела. Написать программу позволяющую добавлять и удалять книги из отдела.

7 Реализовать класс «Человек», включающий имя, фамилию, отчество, год рождения и методы, позволяющие изменять/получать значения этих полей.

Реализовать производные классы:

- «Предприниматель» - содержит номер лицензии, адрес регистрации, УНН, данные о налоговых платежах (массив пар вида <дата, сумма>).
- «Турист» - содержит данные паспорта(строка), данные о пересечении границы в виде массива пар <дата, странах

Классы должны содержать методы доступа и изменения всех полей.

6.6 Перегрузка операторов

6.6.1 Основные принципы перегрузки операторов

Программы на языке C++ используют некоторые такие ранее определенные простейшие классы(типы), как int, char, float и т.д. Мы можем описать объекты указанных классов, например:

```
int a,b;
char c,d,e;
float f;
```

Здесь переменные a,b,c,d,e,f можно рассматривать как простейшие объекты. В языке определено множество операций над простейшими объектами, выражаемыми через операторы, такие как +, -, *, /,% и т.д. Каждый оператор можно применить к операндам определенного типа.

```
float a,b=3.123,c=6.871;
a=c+b;        // нет ошибки
a=c%b;        // ошибка
```

Второе является ошибочным, поскольку операция % должна быть приложена лишь к объектам целого типа. Из этого следует: операторы языка можно применить к тем объектам, для которых они были определены.

К сожалению, лишь определенное число типов непосредственно поддерживается любым языком программирования. Например, языки C и C++ не позволяют выполнять операции с комплексными числами, матрицами, строками, множествами и т.п. Однако все эти операции можно определить через классы в языке C++. Рассмотрим пример. Пусть заданы множества A и B:

A={a1,a2,a3}; B={a3,a4,a5};

И мы хотим выполнить операции типа пересечение & и объединение | множеств:

A&B={a1,a2,a3} & {a3,a4,a5}={a3};

A|B={a1,a2,a3) | {a3,a4,a5}={a1,a2,a3,a4,a5};

Языки C/C++ не поддерживают непосредственно эти операции, однако в языке C++ можно объявить класс, назвав его set (множество). Далее можно определить операции над этим классом, выразив их с помощью знаков операторов, которые уже есть в языке C++, например & и |. В результате операторы & и | можно будет использовать, как и раньше, а также снабдить их дополнительными функциями (функциями объединения и пересечения множеств). Как определить, какую функцию должен выполнять оператор: старую или новую? Надо посмотреть на тип операндов в соответствующем выражении. Если операнды - это объекты целого типа, то нужно выполнить операцию «побитового И» или «побитового ИЛИ». Если же операнды - это объекты типа set, то надо выполнить объединение или пересечение соответствующих множеств.

6.6.2 Указатель this

Каждый новый объект имеет скрытый от пользователя свой указатель. Иначе это можно объяснить так. Когда объявляется объект, под него выделяется память. В памяти есть специальное поле, содержащее скрытый указатель, который адресует начало выделенной под объект памяти. Получить значение указателя в компонентах-функциях объекта можно с помощью ключевого слова this (рисунок 6.13). Для любой функции, принадлежащей классу my_class, указатель this неявно объявлен так: my_class *const this;

Если объявлен класс и объекты

```
class string
{



}str1, str2;
```

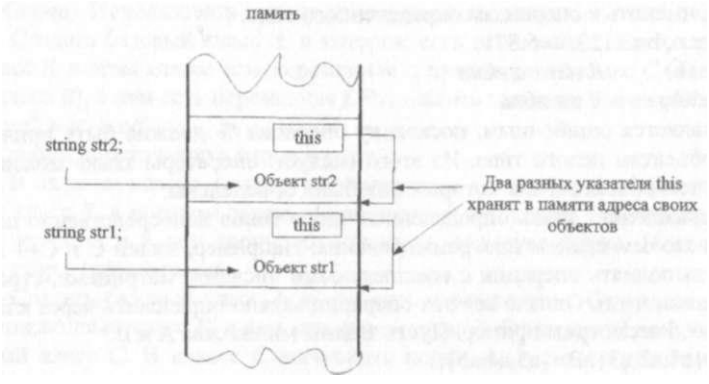


Рисунок 6.13 - Указатель this

Основные свойства и правила использования указателя this:

- каждый новый объект имеет свой скрытый указатель this;
- this указывает на начало своего объекта в памяти компьютера;
- this не надо дополнительно объявлять;
- this передается как скрытый аргумент во все нестатические (т.е. не имеющие спецификатора static) компоненты-функции своего объекта;
- this - это локальная переменная, которая недоступна за пределами объекта (она доступна только во всех нестатических компонентах-функциях своего объекта);
- разрешается обращаться к указателю this непосредственно в виде this или *this.

Пример:

```
#include<iostream>
using namespace std;
#include<string.h>
class String
{       char str[ 100];
        int k;
        int *r;
      public:
        String() {k=123; r=&k;}
        void read(){ gets(str);}
        void print() { puts(str); }
        void read_print();
};
void String::read_print()
{       char c;
        cout<<this->k<<endl;
        cout<<*(this->k)<<endl;
        cout<<"Введите строку"<<endl;
/*Ключевое слово this содержит скрытый указатель на класс String. Поэтому конструкция this->read() выбирает через указатель функцию read() этого класса*/
        this->read();
        cout<<"Введенная строка"<<endl;
        this->print();
/* в цикле for одинаково удаленные от середины строки символы меняются местами*/
        for(int i=0,j=strlen(str)-1;i<j; i++,j--)
        {       c=str[i];
                str[i]=str[j];
                str[j]=c;
        }
        cout<<"Измененная строка"<<endl;
        (*this).print();
```

```
void main(void)
{       String S;
        S.read_print();
}
```

6.6.3 Функция operator

Функция operator может быть использована для расширения области приложения следующих операторе:

```
+  -  *  /  %  &  |  и т.д.
```

Операции, которые не могут (быть перегружены:

```
.,  .*,  ::,  ?:, sizeof
```

Для перегрузки (доопределения) оператора разрабатываются функции являющиеся либо компонентами, либо friend-функциями того класса, для которого они используются. Остановимся на перегрузке пока только с использованием компонент функций класса. Для того чтобы перегрузить оператор, требуется определить действие этого оператора внутри класса. Общая форма записи функции-оператора являющееся компонентой класса имеет вид:

```
тип_возв_значения имя_класа::operator#(список аргументов)
{
        действия, выполняемые применительно к классу
}
```

Вместо символа # ставится значок перегружаемого оператора. Оператор всегда определяется по отношению к компонентам некоторого класса. В результате его старое предназначение сохраняет силу. Функция operator является компонентой класса. При этом в случае унарной операции operator не будет иметь аргументов, а в случае бинарной операции будет иметь один аргумент. В качестве отсутствующего аргумента используется указатель this на тот объект, в котором определен оператор. Объявление и вызов функции operator осуществляется так же, как и любой другой функции. Единственное ее отличие заключается в том, что разрешается использовал сокращенную форму ее вызова. Так, выражение operator#(a,b), можно записать в сокращенной форме a#b.

Рассмотрим пример. Программа доопределяет значение оператора &. В результате его можно будет использовать для выполнения операции пересечения множеств:

```
#include<iostream>
using namespace std;
class set                              // класс «множество»
{
        char str[80];
      public:
        set(char *ss):str(ss) {}       // это конструктор
        char * operator&(set);         // объявление функции operator
};
char * set::operator&(set S)           // описание функции operator
{       int t=0, l=0;
```

```
        while(str[t++]!='\0');       // длина строки str
        char *s1=new char[t];        // выделить память под новую строку s1
        for(int j=0; str[j]!='\0'; j++) // пока не конец строки
            for(int k=0; S.str[k]!='\0'; k++)
               if(str[j]==S.str[k])       // совпадение символов в строках
               {
                  s1[l]=str[j];
                  l++;
                  break;
               }
        s1[l]='\0';
        return s1;
}
void  main(void)
{
        set  S1="1f2bg5e6",  S2="abcdef";              // задаются два множества
        cout<<(S1 &           S2)<<endl;            // результат fbe
        cout<<(set("123") & set("426"))« endl;  // результат 2
}
```

Приведем еще пример программы перегрузки оператора "-" для использования его при вычитании из одной строки другой:

```
#include<iostream>
using namespace std;
#include<string.h>
class String                          // Описание класса String
{
        char str[80];
    public:
        void init(char *s);          // функция инициализации
        int operator - (String s_new);
};
void  String::init(char *s)
{
         strcpy(str,s);
}
int  String::operator - (String  s_new)
{
        for(int i=0; str[i]==s_new.str[i]; i++) // цикл пока символы в строках
         if(!str[i])return 0;                          //совпадают
        return(str[i]-s_new.str[i]);
}
void  main(void)
{
        char S1[51], S2[51];
        cout<<"Введите  первую строку"<<endl;
        cin>>S1;
```

```
        cout<<"Введите вторую строку"<<endl;
        cin>>S2;
        String my_string1, my_string2;
         my_string1.init(S1);          // инициализация объекта my_stringl
         my_string2.init(S2);          // инициализация объекта my_string2
        cout<<"\n String1-String2=";
        cout<<(my_string1-my_string2)<<endl;
}
```

6.6.4 Присваивание и инициализация объектов

Пусть объявлен

```
class String
{       char *str;              // указатель на строку
   public:
        String(char *s=""); // конструктор с параметрами по умолчанию
        ~String();              //деструктор
};
String::String(char *s)      // определение конструктора
{
        str=new char[strlen(s)+1];
        strcpy(str,s);
}
String::~String()            // определение деструктора
{ delete [] str;}
int main()
{       char ss[50];
        cout<<"Введите строку"<<endl;
        cin>>ss;
        String obj(ss);      // описание и инициализация объекта obj
        String obj1;         // описание объекта obj1
        obj1=obj;            // присваивание объекта obj объекту obj1
}
```

Выполнение оператора obj1=obj; будет ошибочным потому что значение obj1.str изменится на obj.str и мы получим два указателя на одну и ту же область памяти. Когда функция завершится, то деструктор вызовется для объектов obj и obj1 и будет освобождать дважды одну и ту же область памяти, что приводит к ошибке. Чтобы избежать этой ошибки необходимо перегрузить оператор = (присваивания). Перегруженный оператор = должен выполнять следующие действия:

- освободить динамическую память объекта obj1 (если эта память была выделена);
- выделить динамически память под данные объекта obj1 в соответствии с размерами этих компонентов, полученными из объекта obj;
- осуществить копирование значений данных объекта obj в данные объекта obj1.

Для класса String объявление перегруженного оператора = (присваивание) будет таким:

String& operator=(const String& ob);

Если необходимо инициализировать объекты так:

```
String obj("Минск");
String obj2=obj;
```

то это тоже приведет к ошибке, потому что конструируется одна строка (в объекте obj), а разрушаются две (объекты obj и obj2). Для устранения этой ошибки необходимо использовать конструктор копирования. Конструктор копирования для класса String объявляется так: String(const String&); Конструктор копирования вызывается всякий раз, когда возникает необходимость в копировании объекта, например, при вызове по значению, когда объект возвращается из вызванной функции, или при инициализации объекта, который должен быть копией другого объекта того же самого класса. Конструктор копирования вызывается в объявлении, когда объект класса String создается и инициализируется другим объектом класса String.

Ранее написанная программа с использованием конструктора копирования и перегруженного оператора присваивания будет выглядеть следующим образом:

```
class String
{
        char *str;
   public:
        String(char *s="");          // конструктор с параметрами по умолчанию
        ~String();                   // деструктор
        String(const String& ob); // конструктор копирования
        String& operator=(const String& ob); // оператор присваивания
        void print();                // функция вывода на печать строки
};
String::String(char *s)              // определение конструктора
{
        str=new char[strlen(s)+1];
        strcpy(str,s);
}
void String::print()
{     cout<<str<<endl;     }
String& String::operator=(const String& ob) // определение оператора
{                                           // присваивания
        if(this!=&ob)                // проверка нет ли копирования объекта в себя
        {
                delete [] str;
                str=new char[strlen(ob.str)+1];
                strcpy(str,ob.str);
        }
                return *this;
}
```

```
String::String(const String& ob)        // определение конструктора копирования

        str=new char[strlen(ob.str)+1];
        strcpy(str,ob.str);

String::~String()

        delete [] str,

int main()

        char ss[50];
        cout<<"Введите строку"<<endl;
        cin>>ss;
        String obj(ss);      // описание и инициализация объекта obj
        cout<<"строка объекта obj"<<endl;
        obj.print();
        String obj1;         // описание объекта obj1
        objl=obj;            // присваивание объекта obj объекту obj1
        cout<<"строка объекта obj1"<<endl;
        obj1.print();
        String obj2=obj1;//инициализация объекта obj2,  вызывается конструктор
                         // копирования для объекта obj2
       cout<<"строка объекта obj2"<<endl;
        obj2.print();
        return 1;
}
```

6.6.5 **Лабораторная работа** №5
Перегрузка операторов

Цель: изучить принципы использования перегрузки операций.

Варианты заданий

1 Реализовать класс *String* для работы со строками символов. Перегрузить операторы + (сложение строк), > (сравнение строк). Предоставить конструктор копирования.

2 Реализовать класс *String* для работы со строками символов. Перегрузить операторы =, +=. Предоставить конструктор копирования.

3 Создать класс, в котором перегрузить операторы:

& для перемножения двух матриц;

+ для сложения двух матриц.

Память под матрицы отводить динамически. Предоставить конструктор копирования.

4 Реализовать класс *String* для работы со строками символов. Перегрузить операторы =, += так, чтобы производилось сложение строки и объекта Предоставить конструктор копирования.

5 Создать класс, в котором перегрузить операторы:

6 для перемножения двух одномерных массивов;

\+ для сложения двух одномерных массивов.

Память под матрицы отводить динамически. Предоставить конструктор копирования.

6 Реализовать класс *String* для работы со строками символов. Перегрузить оператор - (минус) так, чтобы определить, насколько одна строка длиннее другой. Предоставить конструктор копирования.

7 Реализовать класс *String* для работы со строками символов. Перегрузить оператор >, так, чтобы возвратить разность кодов первой пары несовпадающих символов в строках. Предоставить конструктор копирования.

8 Создать класс, в котором перегрузить оператор & для пересечения двух множеств. Память под массивы отводить динамически. Предоставить конструктор копирования.

9 Создать класс, в котором перегрузить оператор + для объединения двух множеств. Память под массивы отводить динамически. Предоставить конструктор копирования.

6.7 Параметризованные классы

6.7.1 Шаблоны функций

Шаблоны позволяют определять при помощи одного фрагмента кода целый набор взаимосвязанных функций, называемых шаблонными функциями. Мы может написать один шаблон функции сортировки массива, на основе которого C++ будет автоматически генерировать отдельные шаблонные функции, сортирующие массивы типов int, float, double и т.д. Все описания шаблонов функций начинаются с ключевого слова template, за которым следует список формальных параметров шаблона заключенный в угловые скобки (o); каждому формальному параметру должно предшествовать ключевое слово class или typename, например:

template <class T>

или template < typename T>

или template < class T, class T1>

Формальные параметры в описании шаблона используются для определения типов параметров функции, типа возвращаемого функцией значения и типов переменных, объявляемых внутри функции. Далее, за этим заголовком, следует обычное описание функции. Необходимо заметить, что ключевое слово class или typename, используемое в шаблоне функции при задании типов параметров, фактически означает «любой встроенный тип или тип, определяемый пользователем». Каждый формальный параметр из списка описания шаблона функции должен появиться в списке параметров функции, по крайней мере, один раз. Имя формального параметра может использоваться в списке парамет-

ров заголовка шаблона только один раз. Одно и то же имя формального параметра шаблона функции может использоваться несколькими шаблонами.

Пример:

```
#include<iostream>
using namespace std;
template<class T>
void printArray(T *array, int count)
{       for(int i=0; i < count; i++)
                cout<<array[i]<<' ';
        cout<<endl;
}
int main()
{
        const int aCount=5, bCount=7, cCount=6;
        int a[aCount]= {1,2,3,4,5};
        double   b[bCount]={1.1,2.2,3.3,4.4,5.5,6.6,7.7};
        char c[cCount]="HELLO";
        cout<<"Массив a"<<endl;
        printArray(a,aCount);
        cout<<"Массив b"<<endl;
        printArray(b,bCount);
        cout<<"Массив c"<<endl;
        printArray(c,cCount);
        return 0;
}
```

T называется параметром типа. Когда компилятор обнаруживает в тексте программы вызов функции printArray, он заменяет T во всей области определения шаблона на тип первого параметра функции printArray и язык C++ создает шаблонную функцию вывода массива указанного типа данных. После этого вновь созданная функция компилируется.

Пример: функции, вычисляющие сумму нескольких аргументов:

```
#include <iostream>
using namespace std;
#include <string.h>
template <class T1,class T2>
T1 sm(T1 a,T2 b)          // описание шаблона
{                         // функции с двумя параметрами
   return (T1)(a+b);
}
template <class T1,class T2,class T3>
T1 sm(T1 a,T2 b,T3 c)   // описание шаблона функции с тремя параметрами
{       return    (T1)(a+b+c);    }
void main()
{
    cout<<"вызов функции суммирования sm(int,int) = "<<sm(4,6)<<endl;
    cout<<"вызов функции суммирования sm(int,int,int) = "<<sm(4,6,1)<<endl;
```

```
    cout<<"вызов функции суммирования sm(int,double) = "<<sm(5,3)<<endl;
cout<<"вызов функции суммирования sm(double,int,short)= "<<sm(.4,6,(short)1)<<endl;

   // cout<<sm("я изучаю","язык C++")<<endl; error cannot add two pointers
}
```

В программе описана перегруженная функция sm(), первый экземпляр которой имеет 2, а второй 3 параметра. При этом тип формальных параметров функции определяется при вызове функции типом ее фактических параметров. Используемые типы T1, T2, T3 заданы как параметры для функции с помощью выражения template <class T1,class T2,class T3>. Это выражение предполагает использование типов T1, T2 и T3 в виде ее дополнительных параметров. Результат работы программы будет иметь вид:

```
вызов функции суммирования sm(int,int)           =  10
вызов функции суммирования sm(int,int,int)       =  11
вызов функции суммирования sm(int.double)        =   8
вызов функции суммирования sm(double,int,short)=  7.4
```

В случае попытки передачи в функцию sm() двух строк, т.е. типов, для которых не определена данная операция, компилятор выдаст ошибку. Чтобы избежать этого, можно ограничить использование шаблона функции sm(), описав явным образом функцию sm() для некоторых конкретных типов данных. В нашем случае:

```
char *sm(char *a,char *b)       // явное описание функции объединения
{   char *tmp=a;                 // двух строк
    a=new char[strlen(a)+strlen(b)+1];
    strcpy(a,tmp);
    strcat(a,b);
    return a;
}
```

Добавление в main() инструкции, например,

cout<<sm("я изучаю"," язык C++")«endl;

приведет к выводу кроме указанных выше сообщения:

я изучаю язык C++

6.7.2 Шаблоны классов

Параметризированный класс - некоторый шаблон, на основе которого можно строить другие классы. Этот класс можно рассматривать как некоторое описание множества классов, отличающихся только типами их данных. В языке C++ используется ключевое слово template для обеспечения параметрического полиморфизма. Параметрический полиморфизм позволяет использовать один и тот же код относительно различных типов (параметров тела кода). Это наиболее полезно при определении контейнерных классов. Шаблоны определения класса и шаблоны определения функции дают возможность многократно использовать код, корректно по отношению к различным типам, позволяя компилятору автоматизировать процесс реализации типа.

Шаблон класса определяет правила построения каждого отдельного класса из некоторого множества разрешенных.

Спецификация шаблона класса имеет вид:

template <список параметров>

class объявление класса

Список параметров класса-шаблона представляет собой идентификатор типа, подставляемого в объявление данного класса при его генерации. Рассмотрим пример шаблона класса работы с динамическим массивом и выполнением контроля значений индекса при обращении к его элементам:

```
#include <iostream>
using namespace std;
#include <string.h>
template <class T>
class vector
{
   T*ms;
   int size;
 public:
   vector(): size(0),ms(NULL) {}
   ~vector(){delete [] ms;}
   void decrem(const T &t)       // увеличение размера массива на 1 элемент
   {
     T *tmp = ms;
      ms=new T[size+1];         // ms - указатель на новый массив
      if(tmp) memcpy(ms,tmp,sizeof(T)*size); // перезапись tmp->ms
      ms[size++]=t;             // добавление нового элемента
      if(tmp) delete [] tmp;    // удаление временного массива
   }
   void inkrem(void)            // уменьшение размера массива на 1 элемент
   { T *tmp = ms;
     if(size> 1) ms=new T[--size];
     if(tmp)
      {
        memcpy(ms,tmp,sizeof(T)*size);//перезапись без последнего элемента
        delete [] tmp;                      // удаление временного массива
       }
   }
   T &operator[](int ind)   // определение обычного метода
   {   if(ind<0 || (ind>=size)) throw IndexOutOfRange; // возбуждение
                              // исключительной ситуации IndexOutOfRange
       return ms[ind];
   }
};
void main()
{
    vector <int> VectInt;
```

```
    vector <double> VectDouble;
    VectInt.decrem(3);
    VectInt.decrem(26);
    VectInt.decrem(l2);       // получен int-вектор (массив) из трех атрибутов
    VectDouble.decrem(1.2);
    VectDouble.decrem(.26);/получен   double-вектор (массив) из двух атрибутов
     int     a=VectInt[1];     //a = ms[1]
     cout<<a<<endl;
     int b=VectInt[4];         // будет возбуждена исключительная ситуация
     cout<<b<<endl;
     double d=VectDouble[0];
     cout<<d<<endl;
     VectInt[0]=1;
     VectDouble[l]=2.41;
}
```

Класс vector наряду с конструктором и деструктором имеет две функции: decrem - добавление в конец вектора нового элемента, inkrem - уменьшение числа элементов на единицу и операция [] обращения к i-му элементу вектора.

Параметр шаблона vector - это любой тип, у которого определены операция присваивания и операция new. Например, при задании объекта типа vector <int> происходит генерация конкретного класса из шаблона и конструирование соответствующего объекта VectInt, при этом тип T получает значение типа int. Генерация конкретного класса означает, что генерируются все его компоненты-функции, что может привести к существенному увеличению кода программы.

Выполнение функций

VectInt.decrem(3);
VectInt.decrem(26);
VectInt.decrem(12);

приведет к созданию вектора (массива) из трех атрибутов (3, 26 и 12).

Сгенерировать конкретный класс из шаблона можно явно, записав:

template vector<int>;

При этом не будет создано никаких объектов типа vector<int>, но будет сгенерирован класс со всеми его компонентами.

В некоторых случаях желательно описание некоторых компонент-функций шаблона класса выполнить вне тела шаблона, например:

```
#include <iosneam>
using namespace std;
template <class T1,class T2>
T1 sm1(T1 aa,T2 bb)              // описание шаблона глобальной
{                                 // функции суммирования значений
    return      (T1)(aa+bb);     // двух аргументов
}
template <class T1,class T2>
class cls
{
   T1  a;
```

```
    T2 b;
 public:
    cls(T1 A,T2 B): a(A),b(B) {}
    ~cls(){}
    T1              sm1()              // описание шаблона функции
    {                                  // суммирования компонент объекта obj
       return (T1)(a+b);
    }
    T1          sm2(T1,T2);           // объявление шаблона функции
};
template <class T1,class T2>
T1 cls<T1,T2>::sm2(T1 aa,T2 bb)  // описание шаблона функции
{                                  // суммирования внешних данных
    return (T1)(aa+bb);
}
void main()
{
    cls <int,int> obj1(3,4);
    cls <double,double> obj2(.3,.4);
    cout<<"функция суммирования компонент объекта 1      = "
        <<obj1.sm1()<<endl;
   cout<<"функция суммирования внешних данных (int,int)  = "
        <<obj1.sm24,6)<<endl;
   cout<<"вызов глобальной функции суммирования (int,int) = "
        <<sm1(4,.6)<<endl;
   cout<<"функция суммирования компонент объекта 2       = "
        <<obj2.sm1()<<endl;
   cout<<"функция суммирования внешних данных (double,double)= "
        <<obj2.sm2(4.2,.1)<<endl;
}
```

6.7.3 Передача в шаблон класса дополнительных параметров

При создании экземпляра класса из шаблона в него могут быть переданы не только типы, но и переменные и константные выражения:

```
#include <iostream>
using namespace std;
template <class T1,int i=0,class T2>
class cls
{  T1  a;
   T2 b;
public:
   cls(T1 A,T2 B):a(A),b(B){}
   ~cls(){}
   T1 sm() //описание шаблона функции суммирования компонент объекта
   {//  i+=3;    // error member function 'int___thiscall  cls<int,2>::sm(void)'
```

```
        return (Tl)(a+b+i);
    i
};
void main()
{ cls <int,1,int> obj1(3,2);       // в шаблоне const i инициализируется 1
    cls <int,0,int>  obj2(3,2,1);  // error'cls<int,0>::cls<int,0>':no overloaded
                                   // function takes 3 parameter s
    cls <int,int,int> obj13(3,2,1); // error 'els': invalid template argument for 'i',
                                    // constant expression expected
     cout<<obj1.sm()<<endl;
}
```

Результатом работы программы будет выведенное на экран число 6.

В этой программе согласно инструкции template <class T1,int i=0,class T2> шаблон класса cls имеет три параметра, два из которых - имена типов (T1 и T2), а третий (int i=0) - целочисленная константа. Значение константы i может быть изменено при описании объекта cls <int**,1**,int> obj1(3,2).

6.7.4 Совместное использование шаблонов и наследования

Шаблонные классы, как и обычные, могут использоваться повторно. Шаблоны и наследование представляют собой механизмы повторного использования кода и могут включать полиморфизм. Шаблоны и наследования связаны между собой следующим образом:

- шаблон класса может быть порожден обычным классом;
- шаблонный класс может быть производным от шаблонного класса;
- обычный класс может быть производным от шаблона класса.

Ниже приведен пример простой программы, демонстрирующей наследование шаблонного класса oper от шаблонного класса vect:

```
#include <iostream>
using namespace std;
template <class T>
class vect                          //класс-вектор
{protected:
   T *ms;                           //массив-вектор
   int size;                        //размерность массива-вектора
 public:
    vect(int n): size(n)            //конструктор
    {
       ms=new T[size];
    }
    ~vect(){delete [] ms;}          //деструктор
    T &operator[] (const int ind)// доопределение операции []
    {  if((ind>0) && (ind<size)) return ms[ind];
       else return ms[0];
    }
};
```

```
template <class T>
class oper : public vect<T>       // класс операций над вектором
{
  public:
   oper(int n): vect<T>(n) {}     // конструктор
   ~oper() {}                     // деструктор
   void print()                   // функция вывода содержимого вектора
   {    for(int i=0;i<size;i++)
          cout<<ms[i]<<'';
        cout<<endl;
    }
};
void main()
{
     oper <int> v_i(4);                 //  int-вектор
      oper <double> v_d(4);             // double-вектор
      v_i[0]=5;     v_i[1]=3;     v_i[2]=2;    v_i[3]=4;     //инициализация int
      v_d[0]=1.3;  v_d[1]=5.1;  v_d[2]=.5;  v_d[3]=3.5;  //инициализация double
      cout<<"int вектор = ";
      v_i.print();
      cout<<"double вектор = ";
      v_d.print();
}
```

Как следует из примера, реализация производного класса от класса-шаблона в основном ничем не отличается от обычного наследования.

Сводный список рекомендуемой литературы

К главе 1

1 *Акулов, О.А.* Информатика: базовый курс: учеб. пособие / О.А. Акулов, Н.В. Медведев. - 2- изд. - М.: Омега-Л, 2005. - 552 с.

2 *Алфимов, М.В.* Государственная политика предоставления доступа к научно-технической информации в условиях инновационного развития общества // Библиотековедение. - 2005. - № 4. - С. 23-25.

3 *Арме, В.* Электронные библиотеки (перевод с англ.). - М.: ПИК ВИНИТИ, 2001.-274 с.

4 *Велихов, А.В.* Основы информатики и компьютерной техники: учеб. Пособие /. - А.В. Велихов. - М.: СОЛОН-Пресс, 2003. - 544 с.

5 Информатика: учебник. / Н.В. Макарова [и др.]; под ред. Н.В. Макаровой. - 3-е перераб. изд..- М.: Финансы и статистика, 2007. - 765 с

6 История информатики и философия информационной реальности: учебное пособие для вузов / [Тузов В. В. и др.]; под ред. Р. М. Юсупова, В. П. Котенко. - М.: Акад. Проект, 2007. - 430 с.

7 *Ковалев, М., Шади А.-С.* Создание электронного правительства с учетом международного опыта// Банкаускі веснік. -2006. - чэрвень. - С. 16 - 25. [Электронный ресурс]. - Режим доступа: http://www.nbrb.bv/bv/narch/345/2.pdf. - Дата доступа: 18.12.2007.

8 Электронные документы: создание и использование в публичных библиотеках: справ. / науч. Ред. проф. Р.С. Гиляревский, проф. Г.Ф. Гордукалова. - СПб: Профессия, 2007. - 664 с.

К главе 2

9 *Борисов, Е.С.* Основные модели и методы теории искусственных нейронных сетей / Е.С. Борисов [Электронный ресурс]. - Режим доступа: http://mechanoid.narod.ru/nns/base/index.html. - Дата доступа: 18.12.2007.

10 *Гмурман, В.Е.* Теория вероятностей и математическая статистика: учеб. пособие для вузов / В.Е. Гмурман. 7-е стер. изд. - М.: Высш. шк., 2001. -479 с.

11 *Головко, В.А.* Нейроинтеллект: теория и применение. Книга 1: Организация и обучение нейронных сетей с прямыми и обратными связями / В. А. Головко / - Брест: Брестский политехнический институт, 1999. - 264 с.

12 Зачем нужны аналитические технологии [Электронный ресурс]. - Режим доступа: http://www.neuroproject.ru/what.php. - Дата доступа: 18.12.2007.

13 *Кальченко, Д.* Нейронные сети: на пороге будущего / Д. Кальченко [Электронный ресурс]. - Режим доступа: http://alt-future.narod.ru/Ai/neiro.htm. - Дата доступа: 18.12.2007.

14 Нейроматематика. Книга 6: учеб. пособие для вузов / В.А. Агеев [и др.]; под общ. ред. А.И. Галушкина. - М.: ИПРЖР, 2002. - 448 с.

15 *Ступчак, А.Я.* Аналитические технологии для прогнозирования и анализа данных // Донецкий национальный технический университет [Электронный ресурс]. - Режим доступа: http://www.uran.donetsk.ua/~masters/2000/fkita/stupchak/oglavl.htm. - Дата доступа: 18.12.2007.

16 Элементы нейронных сетей [Электронный ресурс]. - Режим доступа: http://www.inmit.ni/department/database/ datamining/ 1 l/2.html. - Дата доступа: 18.12.2007.

К главе 3

17 *Лавренов, СМ.* Excel: Сборник примеров и задач / СМ. Лавренов. - М.: Финансы и статистика, 2002. - 336 с.
18 *Мур, Дж.Х.* Экономическое моделирование в Microsoft Excel / Дж.Х. Мур, Л.Р. Уэдерфорд. - М.: Издательский дом "Вильямс", 2004. - 1024 с.

К главе 4

19 *Алексеев, Е.Р.* Решение задач вычислительной математики в пакетах MathCAD 12, Matlab 7, Maple 9 / Е.Р. Алексеев, О.В. Чеснокова. - М.: НТПресс, 2006.-496 с.
20 *Дьяконов, В.П.* Matlab 6/6.1 / 6.5. Simulink 4.5. Основы применения / В.П. Дьяконов. - М.: Солон-пресс, 2004. - 768 с.
21 *Иглин, СП.* Математические расчеты на базе Matlab / СП. Иглин. - СПб.: БХВ-Петербург, 2005. - 634 с.

К главе 5

22 *Дьяконов, В.П.* Mathematica 4.1, 4.2, 5.0 в математических и научно-технических расчетах / В.П. Дьяконов. - М.: Солон-пресс, 2004. - 670 с.
23 *Прокопеня, А.Н.* Решение физических задач с использованием системы Mathematica / А.Н. Прокопеня. - Брест, БГТУ, 2005. - 260 с.

К главе 6

24 *Буч, Г.* Объектно-ориентированный анализ и проектирование / Г. Буч. - СПб.: Издательство «Невский Диалект», 2001. - 558 с.
25 *Дейтел, Х.* Как программировать на С++ / Х. Дейтел, П. Дейтел. - М.: ЗАО «Издательство БИНОМ», 2001. - 1152 с.
26 *Демидович, Е.М.* Основы алгоритмизации и программирования. Язык Си / Е.М. Демидович. - Минск: Бестпринт, 2001. — 411 с.
27 *Луцик, Ю.А.* Объектно-ориентированное программирование на языке С++ / Ю.А. Луцик, А.М. Ковальчук, И.В. Лукьянова. - Минск: БГУИР, 2003. - 202 с.
28 *Скляров, В.А.* Язык С++ и объектно-ориентированное программирование / Мн.:Выш.шк., 1997.-478 с: ил.
29 *Страуструп, Б.* Язык программирования С++ / Б. Страуструп. - М.: ЗАО «Издательство БИНОМ», 2004. - 1098 с.

Учебное издание

Батин Николай Владимирович, Богданова Ирина Феликсовна,
Ковальчук Анна Михайловна и др.

Основы информационных технологий

Учебно-методическое пособие

Редактор, корректор А.А. Сычев

**Подписано в печать 23. 07. 2008. Формат 60х84 1/16. Бумага офсетная. Гарнитура Таймс.
Печать ризографическая. Усл. печ. 13,82 л. Уч.-изд. 12,44 л. Тираж 150 экз. Заказ 150.**

**ГУО «Институт подготовки научных кадров Национальной академии наук Беларуси»
220049, г. Минск, ул. Кнорина, 1. ЛИ № 02330 / 0133427**

**Отпечатано в РУП «Научно-практический центр Национальной академии наук Беларуси
по механизации сельского хозяйства» 220049, г. Минск, ул. Кнорина, 1.
ЛЛ № 02330-0150026 от 10.05.2007 г.**