А.П. Полищук, С.А. Семериков

АВТОМАТИКА

Учебное пособие для студентов вузов

Кривой Рог
Издательский отдел КГПИ
1999

Полищук А.П., Семериков С.А.

Автоматика: Учебное пособие. – Кривой Рог: Издательский отдел КГПИ, 1999. – 276 с.

Учебное пособие, ориентированное на программирующего пользователя, посвящено анализу динамики линейных систем методами операционного исчисления и динамического программирования. Приведен лабораторный практикум по компьютерному моделированию линейных стационарных динамических систем операторным методом с исходными текстами программ в C++ Builder и Delphi.

Для студентов высших учебных заведений, аспирантов, научных и инженерно-технических работников.

Рецензент:

д-р физ.-мат. наук, проф. **В.Н. Соловьёв**

Оглавление

Трудно жить на свете
Пастушонку Пете –
Трудно хворостиной
Управлять скотиной.

1. Управление и регулирование: основные понятия и определения

Управление действительно одна из важнейших и труднейших проблем, с которыми приходится сталкиваться человеку в его профессиональной деятельности. Она возникает в связи с необходимостью целенаправленного воздействия на связанные с этой деятельностью процессы, а в более широком смысле – с необходимостью создания очагов упорядоченности, организованности участвующей в этих процессах материи. Самоподдерживающийся порядок – не что иное, как хаос, а достижение всякой цели требует немалых усилий (прежде всего интеллектуальных – памяти, знаний, опыта, способности принимать решения в зависимости от обстоятельств) по преодолению стремления живой или неживой материи к безграничному росту энтропии (дезорганизованности).

Если до недавнего времени основной задачей техники была разработка машин, облегчающих физический труд человека, и основные усилия были направлены на средства получения, передачи, преобразования и использования различных видов энергии, то теперь возросшая сложность задач управления этими средствами вынудила сместить центр тяжести технических задач в направлении создания методов и средств получения, передачи, преобразования и использования необходимой для управления информации. Количество информации, которое необходимо перерабатывать в единицу времени для обеспечения современных требований к точности, качеству, быстродействию сложных технологических и организационных процессов стало столь большим, что человек не справляется с управлением им же созданными машинами и их комплексами. Кроме того, часто сам характер процессов (атомная, химическая промышленность) требует замены человека автоматами.

Эта насущная потребность общества в увеличении «ин-

теллектуальной мощности» участвующего в управлении человека привела к попыткам разработать на основании методов точных наук общий подход к изучению всего многообразия процессов управления объектами самой различной природы – такая общая теория управления получила название *Кибернетика* после публикации Норбертом Винером в 1948 году книги «Кибернетика или управление и связь в живых организмах и машине». Раньше этим термином пользовались Платон при описании науки о кораблевождении и Ампер для обозначения предполагаемой им науки о способах управления обществом.

Кибернетика абстрагируется от природы управляемых объектов (механических, химических, экономических, биологических и т.д.) и сосредоточивает внимание на установлении общих принципов и законов управления, направленного на достижение поставленной цели и основанных на методах получения, передачи, переработки и использования информации.

В последние годы широкое распространение получил термин ***информатика***; его определение совпадает с определением кибернетики, за исключением функции использования обработанной информации для целей управления. Таким образом, информатику можно либо считать составной частью кибернетики, изучающей методы получения информации, предназначенной для управления, либо можно считать обе науки частями еще более общей области исследований, пока не имеющей названия, либо оба названия считать пока синонимами в определении одной и той же области знаний.

В зависимости от того, к какой отрасли знаний применяется такой информационно-алгоритмический подход, говорят о *технической, биологической, экономической кибернетике* (и информатике тоже).

Кибернетика изучает общие методы управления безотносительно к участию или неучастию человека в контуре

выработки управляющих воздействий.

Автоматика – один из разделов технической кибернетики – представляет собой науку об общих принципах и методах построения автоматических систем, т.е. устройств, механизмов, машин, агрегатов, цехов, заводов, выполняющих поставленные перед ними цели без непосредственного участия человека. Под *автоматизацией* при этом понимают процесс превращения этих объектов из неавтоматических (управляемых непосредственно человеком) в автоматические.

Системы автоматики на сложных объектах выполняются, как правило, многоуровневыми – на нижнем уровне выполняются при этом ***задачи регулирования*** – автоматического поддержания на заданных уровнях или изменения по заданной программе отдельных параметров, определяющих режимы работы соответствующих объектов (температуры, давления, расходы рабочих сред, концентрации компонентов в смесях, скорости, позиции механизмов и пр.). На более высоких уровнях осуществляется формирование этих заданий с участием человека и компьютерных программ.

Всякое действие, называемое управлением, подразумевает наличие ***объекта***, реализующего некоторый ***процесс***, и ***управляющего устройства*** – их совокупность представляет собой ***систему управления***, поведение которой определяется внешней средой, свойствами системы и целью управления.

Под ***процессом*** понимают изменение некоторой величины во времени и пространстве – это может быть изменение температуры металла, нагреваемого в проходной печи перед прокаткой, изменение химического состава вещества в химическом реакторе, изменение прибыли предприятия, изменение координат движущейся ракеты, изменение численности определенного вида животных в некотором регионе и т.п.

Процесс начинается и протекает благодаря внешним

воздействиям на реализующий его объект. Эти воздействия могут быть целенаправленными (***управляющими***), приложенными для обеспечения желаемого характера протекания процесса – изменение подачи топлива в горелки нагревательного устройства, изменение подачи реагентов в химический реактор, изменения объема используемых оборотных средств предприятия, изменение тяги ракетного двигателя или положения рулей самолета, изменение факторов, влияющих на прирост численности животных и т.д.

Другой тип воздействий на процесс называют ***возмущениями*** (нежелательными воздействиями) – колебания силы ветра, отклоняющего летательный аппарат от расчетного курса, изменения теплотворной способности используемого топлива в нагревательных устройствах, колебания спроса на продукцию предприятия и др.

Управление протекающим в объекте процессом состоит в определении и последующей реализации таких управляющих воздействий, которые обеспечили бы желательное или по возможности близкое к нему течение процесса, подвергающегося действию нежелательных и часто непредсказуемых и неконтролируемых возмущений.

Но для определения необходимых управляющих воздействий необходимо знать, как процесс реагирует на эти воздействия – другими словами, для вычисления управлений необходимо знать их взаимосвязь с управляемыми (выходными) величинами и эта взаимосвязь должна быть выражена в виде математических соотношений, которые называют ***математической моделью процесса***. Таким образом, математическое моделирование процессов является неотъемлемой составной частью процесса управления.

2. О классификации систем управления

Быстрое расширение функционального назначения управляющих систем, ставшее возможным благодаря применению компьютеров в качестве управляющих устройств, не позволяет дать исчерпывающую их классификацию по признакам, связанным с областью применения.

Прежде всего, все системы делят на две основные группы – замкнутые и разомкнутые.

На рисунке мы представили примерную общую схему так называемой ***замкнутой*** системы автоматического регулирования скалярного или векторного выходного параметра управляемого процесса, использующей для выработки регулирующего воздействия информацию об отклонении регулируемой величины от задания. Другое название таких систем – ***системы с обратной связью*** или ***системы, работающие по отклонению*** регулируемой величины. В этой системе устройство управления содержит сведения о математической модели управляемого процесса, в каждый момент времени получает информацию о действующем возмущении и отклонении выходной величины от задания,

вычисляет управляющее воздействие для уменьшения рассогласования между заданным и фактическим значением выхода процесса и подает его на вход объекта управления.

В ***разомкнутых системах*** управляющее устройство не получает информации о действительном текущем состоянии управляемого процесса. Такие системы могут обеспечить сравнительно приемлемую точность регулирования только при условии измерения действующих на процесс возмущений – в этом случае устройство управления может вычислить и подать на вход объекта такое управляющее воздействие, которое скомпенсирует, нейтрализует действие возмущения, что приведет выходную величину в соответствие с требуемым значением.

Этот процесс называют ***компенсацией возмущений***, а соответствующая цепь воздействия не является линией обратной связи, так как по ней передается входная, а не выходная величина. Как правило, измерение возмущений трудно реализуемо и область применения чисто компенсационных разомкнутых систем ограничена.

По-видимому, наиболее общими классификационными признаками для управляющих систем являются:

- характеристики управляемого процесса;
- предъявляемые к управляемому процессу требования;
- используемая в системе информация об управляемом процессе.

Первое направление классификации связано с характером зависимости его выходных величин $\boldsymbol{x}$ от управляющих $\boldsymbol{u}$ и возмущающих $\boldsymbol{z}$ воздействий:

$$\boldsymbol{x}=\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{z}).$$

Здесь $\boldsymbol{F}$ – в общем случае оператор, т.е. закон соответствия между двумя множествами функций и чаще всего задается в виде дифференциальных уравнений. Если это система обыкновенных дифуравнений, то объект относят к классу объектов ***с сосредоточенными параметрами***, а в случае уравнений с частными производными – к объектам ***с***

распределенными параметрами. Другой способ классификации по свойствам объектов делит системы на ***непрерывные, дискретно-непрерывные*** (с квантованными по времени и неквантованными по уровню величинами с оператором, к примеру, в виде системы конечно-разностных уравнений) и ***дискретные*** (с квантованием величин как по времени, так и по уровню).

В число характеристик объектов входят также ограничения на управляющие воздействия и на координаты управляемого процесса. Важность ограничений очевидна – нельзя применять сколь угодно большие напряжения или токи при разгоне электродвигателей и т.д.

Второе направление классификации систем связано с целевой функцией управления – обычно она формулируется как необходимость достижения экстремума некоторого ***критерия оптимальности***:

$$Q(\boldsymbol{x}, \boldsymbol{x}^*, \boldsymbol{u}, \boldsymbol{z}, t)=min.$$

В качестве критерия могут быть выбраны различные технические или экономические показатели – производительность, качество продукции, затраты сырья или энергоносителей, время регулирования или максимальное отклонение регулируемой величины.

Важным направлением классификации является разделение систем по характеру априорной и оперативной информации об управляемом процессе в управляющем устройстве. В этом направлении прежде всего различают системы ***с полной и неполной информацией***, которая состоит из:

- информации об операторе объекта (его математической модели);
- информации о действующих возмущениях;
- информации о текущем состоянии объекта – например, о значении выходного параметра и всех его производных;
- информации о цели управления.

Трудно представить себе систему, полностью соответствующую классу «с полной информацией» – слишком много возможных каналов проникновения неопределенности в автоматическую систему: например, в следящих системах это, прежде всего, неопределенность будущих значений задания. Примером такой системы может служить система поддержания заданного соотношения «топливо – воздух» для управления горением с непредсказуемым изменением расхода топлива, копировально-фрезерные станки с копиром, самонаводящиеся на движущуюся цель ракеты и вообще все системы, реализующие алгоритм «погони собаки за зайцем». Оптимальное поведение отслеживающего задание автомата очевидно – это движение наперерез к прогнозируемой точке встречи с «зайцем» – но какой прогноз может быть достаточно точным? О невозможности измерения и прогнозирования различных возмущений мы уже говорили. И даже цепь обратной связи – мощное средство повышения помехоустойчивости систем – тоже является источником неопределенностей, прежде всего из-за необходимости учитывать производные выходного сигнала. Даже маленькие помехи высоких частот вносят огромные погрешности при дифференцировании и сведения о текущем состоянии объекта получаются искаженными. И, наконец, сам оператор объекта не может считаться неизменным весь срок своей службы – объект изнашивается, стареет в процессе эксплуатации, а следовательно меняются априорно заложенные в управляющее устройство его характеристики – например, коэффициенты уравнений и даже их структура.

Неполнота информации об управляемом процессе приводит к необходимости возложения на управляющее устройство дополнительных функций: помимо вычисления управляющих воздействий для приведения объекта к заданному состоянию приходится также определять его характеристики и состояние. При этом накопление информа-

ции об объекте может носить характер ***пассивного наблюдения или активного эксперимента*** с нанесением пробных возмущений (так называемое ***дуальное управление***). Системы с неполной информацией составляют быстро развивающийся класс ***самонастраивающихся систем***. К ним относятся:

- ***системы экстремального регулирования***, рабочей информацией которых является отклонение от экстремума (возможно «дрейфующего») некоторой функции одной или нескольких переменных; для определения отклонения от эктремума объекту наносятся поисковые возмущения в его окрестностях;
- ***системы с самонастраивающимися корректирующими устройствами***, позволяющие обеспечить устойчивость и требуемое качество процесса регулирования в условиях изменения и неполного знания свойств регулируемого объекта. Эти системы, в свою очередь, подразделяются на ***системы с разомкнутыми цепями самонастройки*** (по измеряемым возмущениям), с ***замкнутыми цепями настройки*** и с ***экстремальной настройкой корректирующих устройств***.

Существует много других используемых классификационных признаков; например, существует деление систем по виду используемой для управления энергии или рабочей среды – электрические, электронные, гидравлические, пневматические системы.

В теории автоматических систем рассматриваются классы ***линейных*** (для них справедлив принцип суперпозиции) и ***нелинейных*** систем, ***стационарных*** (с неизменными во времени параметрами) и ***нестационарных*** систем.

3. Физические основы измерительных преобразователей автоматических систем

Помимо управляемого объекта и управляющего устройства в автоматической системе функционально необходимы еще два типа элементов – ***измерительные и исполнительные устройства***. В нашем вводном курсе мы не будем останавливаться на различных типах (электрических, гидравлических, пневматических) серводвигателей, усилителей, переключающих устройств и механизмов, передающих управляющие воздействия непосредственно на управляемый объект путем перемещения задвижек, клапанов, рулевых устройств и пр. или изменяющих, например, углы зажигания в тиристорных преобразователях питания электродвигателей.

Измерительные устройства и преобразователи – это органы чувств автоматической системы, поставляющие управляющему устройству информацию о значениях выходных координат и контролируемых возмущений – без них система будет слепой и глухой и не сможет выполнять своих функций. В кратком курсе нет возможности останавливаться на таких важных характеристиках измерительных устройств как ***чувствительность, точность, быстродействие, надежность, экономичность*** и пр. Мы рассмотрим только физические основы для преобразования некоторых величин в электрические сигналы – именно электрическая форма их представления наиболее приемлема для ввода в современные, как правило электронные, управляющие устройства.

3.1. Физика преобразователей температуры

Для измерения и преобразования сигнала температуры используются самые различные физические явления и эффекты.

Эффект температурного расширения металлов и

сплавов обычно используется в регуляторах температуры бытовых приборов; чаще всего такой измеритель температуры имеет вид биметаллической пластинки из двух металлов с различными коэффициентами температурного расширения. При нагревании такая пластинка изгибается и к ней можно прикрепить либо контакт для подачи питания на нагреватель (как в утюге), либо преобразователь перемещения со шкалой для индикации температуры.

Эффект температурной зависимости удельного омического сопротивления металлов – промышленность выпускает медные (до 200 °С) и платиновые (до 500 °С) термометры сопротивления; они представляют собой проволоку, намотанную на металлический каркас и заключенную в защитный металлический трубчатый чехол. Для преобразования в сигнал электрического тока это сопротивление включается в мостовую схему.

Эффект различной работы выхода электронов у различных металлов и сплавов; для использования этого эффекта концы двух кусков проволоки из сплавов с различной работой выхода сваривают и помещают в место измерения температуры (горячий спай), а свободные неспаянные концы (холодный спай) – в место с постоянной температурой или в термостат. На свободных концах появляется термоэлектродвижущая сила, пропорциональная разности температур холодного и горячего спаев. Устройство называется термопарой; промышленность выпускает их из пар железо–никель, хромель–копель, хромель–алюмель, платина–платинородий в виде трубчатых металлических или огнеупорных чехлов с заключенными в керамические трубки электродами внутри.

Эффект излучения нагретых до высокой температуры тел используется для дистанционного измерения температур в радиационных пирометрах: радиационный поток фокусируется на батарее термопар или терморезисторов с помощью собирательных линз из стекла или кварца. Эффект

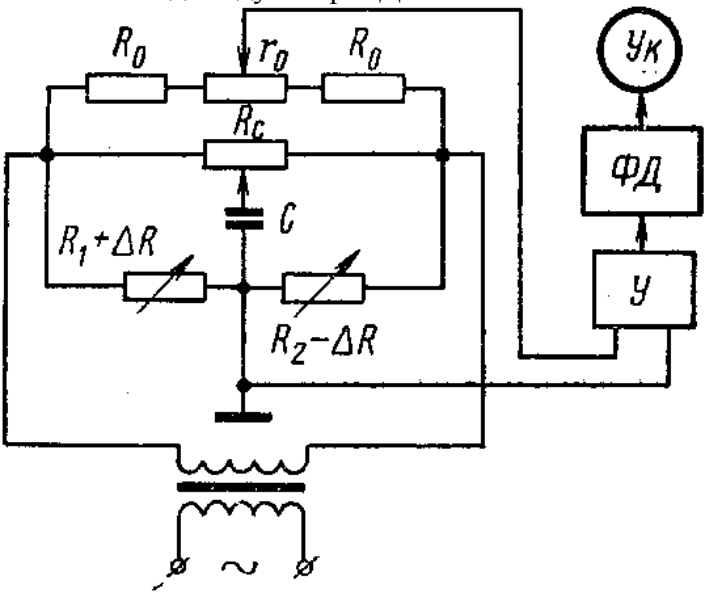
излучения используется и в яркостных пирометрах – приборы сравнивают яркость исследуемого объекта с яркостью образцового излучателя (нагретая током плоская вольфрамовая нить) в узком участке спектра яркости. Этот же эффект используется в цветовых пирометрах, измеряющих отношения интенсивности излучения на двух или больше разных длинах волн из красной и синей областей спектра – их преимущество в независимости показаний от расстояния до объекта измерения и возможность измерения сверхвысоких температур в сотни тысяч градусов, а недостаток в сложности, обусловленной необходимостью для получения результата решать нелинейное уравнение, составленное по зависимости спектральной плотности излучения от температуры.

3.2. Физика измерения усилий

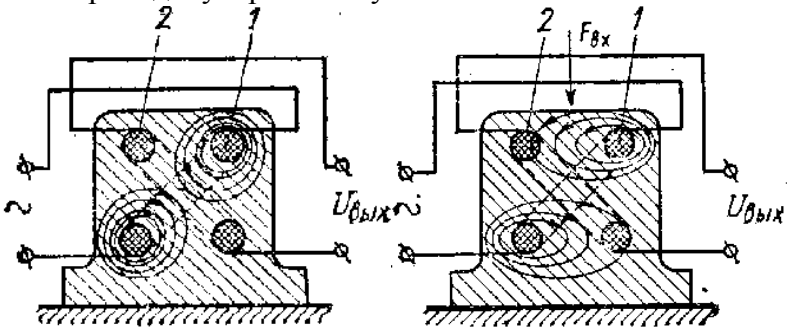
Нуждающиеся в измерении усилия и крутящие моменты, действующие на детали машин и механизмов, элементы конструкций и сооружений могут принимать значения в очень широком диапазоне – в задачах контроля управляемых технологических процессов они достигают по верхней границе 10^6–10^8 Н, а по нижней – 10^{-5}–10^{-12} Н. Таким образом, полный диапазон измеряемых усилий составляет 10^{20}, и задача создания единых измерителей на весь диапазон потребностей пока неразрешима. Помимо очевидных методов уравновешивания и использования деформации упругих пружин для измерения механических усилий используют различные физические свойства материалов, в частности:

Зависимость омического сопротивления некоторых сплавов (константан, нихром, платинородий) ***от величины упругих деформаций*** используется для измерения усилий в ***тензометрических*** преобразователях с проволочными, фольговыми или полупроводниковыми тензорезисторами. Тензорезистор приклеивается к несущему нагрузку упру-

гому элементу с близкими к тензоматериалу коэффициентами температурного расширения и испытывает одинаковые с ним упругие деформации под действием приложенных к несущему элементу сил. Сам тензорезистор включается в измерительный резисторный мост для преобразования результата (изменения омического сопротивления) в электрический сигнал. Полупроводниковые тензорезисторы изготавливаются из монокристаллов кремния, германия, арсенида галлия и др. – они отличаются высоким коэффициентом тензочувствительности (-200 – +850) при малых размерах (до 2,5 мм). Ниже на рисунке приведена типовая мостовая схема с тензорезисторами R_1 и R_2, один из которых испытывает деформацию растяжения, а другой деформацию сжатия: такое дифференциальное включение позволяет устранить температурную погрешность и вдвое увеличить чувствительность. Снимаемый с измерительной диагонали моста сигнал требует усиления по мощности, поэтому подается на указатель Ук через усилитель У и фазочувствительный демодулятор ФД.



Эффект изменения магнитных свойств материалов (электротехническая сталь, никелевые сплавы, пермаллой) ***под действием механических напряжений (магнитная упругость)***; измеряемая величина в магнитоупругих преобразователях – индуктивность катушки, намотанной на нагружаемый сердечник, преобразуемая в электрический сигнал с помощью индуктивной мостовой схемы. Другой вариант таких преобразователей основан на ***магнитной анизотропии упруго нагруженных ферромагнетиков*** и использует для преобразования взаимоиндукцию двух взаимно перпендикулярных катушек.



Магнитоупругий преобразователь взаимоиндуктивного типа в ненагруженном и нагруженном состоянии

Пьезоэффект – возникновение электрических зарядов на гранях некоторых кристаллов (пьезоэлектриков) при приложении к другим граням механических усилий; типичными представителями пьезоэлектриков являются кварц (двуокись кремния) с ненарушенной (например, плавлением) кристаллической решеткой и титанат бария. Величина заряда при этом не зависит от размеров пьзоэлемента, а определяется только значением силы и величиной пьзоэлектрической постоянной (модуля, зависящего от свойств материала – для кварца $2{,}1 \cdot 10^{-12}$ Кл/Н). В связи с большим внутренним сопротивлением пьезоэлектрического преобразователя и малой мощностью сигнала, выходное

напряжение требует усиления с помощью усилителей с большим входным сопротивлением (10^{14} Ом).

3.3. Методы измерения параметров движения

Параметрами однонаправленного поступательного, вращательного или колебательного движения (с периодически изменяющимся направлением) являются пройденный ***путь***, ***скорость*** и ***ускорение***, связанные между собой простейшими дифференциальными или интегральными зависимостями (скорость и ускорение могут быть найдены как первая и вторая производные пройденного пути, а скорость и путь интегрированием ускорения и скорости соответственно).

Приборы для измерения скорости и пути поступательного движения называют счетчиками и спидометрами, измерители скорости вращательного движения – тахометрами, измерители секундных расходов и количеств жидкостей и газов – расходомерами и счетчиками количества. Приборы для измерения параметров вибрационного движения подразделяют на виброметры (измерители амплитуды вибраций), велосиметры (измерители скорости) и акселерометры (измерители ускорений). Диапазон изменения всех перечисленных параметров движения в зависимости от области приложений чрезвычайно широк и не может быть перекрыт каким-либо универсальным измерительным устройством – такие приборы строятся как специализированные устройства для узких интервалов изменения. В ряде случаев (например, в задачах инерциальной навигации) требуется перекрытие значительного диапазона изменения измеряемого параметра: так, инерциальная система управления полетом ракеты предназначена для определения пройденного пути двукратным интегрированием ускорения (очень больших запускных порядка 100 м/сек^2 и малых тормозных порядка 10^{-5} м/сек^2). Эта задача в полной мере пока не решена.

Измерение пути осуществляют обычно преобразованием линейных перемещений во вращательное движение и подсчетом числа оборотов этого движения.

Для измерения скорости часто используется вращающееся магнитное поле – с вращающимся валом связывают магнит, индуцирующий ток в расположенном через воздушный зазор проводящем диске; этот диск будет испытывать вращающий момент, пропорциональный угловой скорости вращения и этот момент можно измерить как угол скручивания измерительной упругой пружины. Дистанционное измерение скорости можно осуществить также с помощью индукционных преобразователей, называемых тахогенераторами – это маломощные генераторы постоянного или переменного тока, выходное напряжение или частота которых пропорциональны измеряемой скорости движения.

Для измерения количества протекающей в трубопроводе жидкости или газа используют индукционные преобразователи, сочлененные с измерительными турбинами, свободно вращающимися в потоке. В тело крыльчатки такой турбины встраивают стержень из магнитномягкого материала, а снаружи трубы устанавливают постоянный магнит с надетой на него катушкой – в этой катушке возникает при вращении крыльчатки э.д.с., частота которой равна удвоенной частоте вращения.

Измерение мгновенных расходов газов и жидкостей в трубопроводах чаще всего осуществляют установкой внутри труб сужающих устройств (диафрагм, труб или сопел Вентури) и измерением создаваемого на таком сужении динамического перепада давлений до и после сужения с поправками на температуру и давление протекающей по трубопроводу среды. Используют и приборы, построенные на других методах, например, тепловой расходомер газа представляет собой участок трубы с электрическим нагревателем и стабилизирующим регулятором температуры по-

сле нагревателя – до и после нагревателя газ имеет различную температуру, измеряемую с помощью терморезисторов. Чем больше расход измеряемой среды, тем больше нужна подводимая к нагревателю мощность для поддержания заданной разности температур до и после нагрева и измеряющий эту мощность ваттметр можно отградуировать в единицах расхода газа. Если вместо ваттметра включить счетчик электроэнергии, то его показания можно пересчитать в единицы количества протекшего газа.

Измерение параметров вибраций и ускорений чаще всего осуществляют с помощью механических колебательных систем и электрических преобразователей, воспринимающих колебания системы. При небольшой инерционной массе, прикрепленной к основанию относительно жесткой пружиной (при высокой собственной частоте подвижной системы) перемещения массы относительно основания пропорциональны внешнему ускорению, действующему на основание. При большой массе, закрепленной на мягкой пружине (низкой собственной частоте), эта масса остается практически неподвижной, несмотря на колебания основания и относительное перемещение основания равно перемещению корпуса преобразователя. Таким образом, этот принцип можно использовать как для построения виброметров, так и акселерометров (но соотношения между собственной резонансной частотой механической системы и диапазоном частот измеряемых вибраций для виброметров и акселерометров прямо противоположны). Наиболее часто измерения производятся в области частот 20–3000 Гц, при этом собственные частоты механических систем виброметров лежат в пределах 2–7 Гц, а акселерометров – 10–15 кГц и выше.

3.4. Физические основы измерения состава и концентрации вещества

Автоматический анализ качественного и количествен-

ного состава твердых, жидких, газообразных, сыпучих и пр. материалов имеет решающее значение для управления многими технологическими процессами. Задача обычно осложняется тем, что измерения необходимо проводить в сложных многокомпонентных средах при различных температурах, давлениях, скоростях перемещения и др. Диапазон изменения измеряемой концентрации достаточно широк – например, определение микроконцентраций хлора, ацетилена, токсичных газов в производственных условиях необходимо осуществлять в пределах до 10^{-4}% объемной концентрации, а при производстве сверхчистых металлов и полупроводников необходимо измерять содержание примесей с концентрацией не более 10^{-8}%. Многообразие измеряемых веществ и широкий диапазон измерений обусловили использование многочисленных методов.

Электрохимические методы используют зависимости состав–сигнал или свойство–сигнал, наиболее распространенными из них являются кондуктометрический, потенциометрический, кулонометрический и полярографический методы.

Кондуктометрический метод основан на измерении электропроводности растворов и применяется для измерения концентрации в них солей, а также для определения концентрации газов по изменению электропроводности растворов при введении в них проб анализируемого газа (например, при вводе CO_2 в водный раствор КОН образуется соль K_2CO_3 и это приводит к изменению электропроводности раствора).

Потенциометрический метод основан на измерении электродных потенциалов и широко применяется в pH-метрах (приборах для измерения активности водородных ионов) и в газоанализаторах.

Кулонометрический метод основан на измерении количества электричества или тока при электролизе исследуемого вещества. Например, кулонометрический гигрометр

представляет собой изоляционную трубку, на внутренней поверхности которой (покрытой тонкой пленкой фосфорного ангидрида P_2O_5 с большим сопротивлением в сухом виде и малым при увлажнении) расположены два спиральных электрода; исследуемый газ с постоянной скоростью подается через трубку, вызывая поглощение влаги пленкой с образованием фосфорной кислоты и электролиз воды с регенерацией фосфорного ангидрида:

$$P_2O_5+H_2O\rightarrow 2HPO_3$$

$$4HPO_3\rightarrow 2H_2\uparrow+O_2+2P_2O_5.$$

Установившийся ток электролиза будет пропорционален абсолютной влажности газа.

Полярографический метод предусматривает снятие вольтамперной характеристики (полярограммы) при электролизе исследуемого раствора в специальном преобразователе с одним электродом большой площади и вторым – малой. При наличии в растворе различных ионов полярограмма представляет собой ступенчатую кривую, величины токов зависят от концентрации соответствующих ионов и используются для количественного анализа, а напряжения в точках скачка токов соответствуют потенциалам выделения ионов и используются для качественного анализа. Наиболее распространены полярографы ртутно-капельного типа (анод – ртуть на дне преобразователя, катод – ртутная капля диаметром 1 мм, периодически вытекающая из капилляра) и твердоэлектродные полярографы с электродами из платины, золота, никеля. Порог чувствительности полярографов – 10^{-9} моль/л.

Ионизационные методы основаны на ионизации анализируемого вещества и измерении ионного тока, пропорционального концентрации ионизированного вещества. Применяются в вакуумметрах, масс-спектрометрах, ионизационно-плазменных анализаторах.

Спектрометрические методы основаны на избирательной способности различных веществ поглощать, излу-

чать, отражать, рассеивать или преломлять различного рода излучения. При этом используется широкий спектр длин волн от звукового диапазона 10 кГц до рентгеновских и гамма-излучений 10^{18} Гц. Эти методы и подразделяются в зависимости от диапазона длин волн:

Электроакустический метод основывается на зависимости скорости звука от состава и концентрации вещества в исследуемой среде; применяется для анализа бинарных смесей газов, например кислорода с азотом, и для измерения влажности.

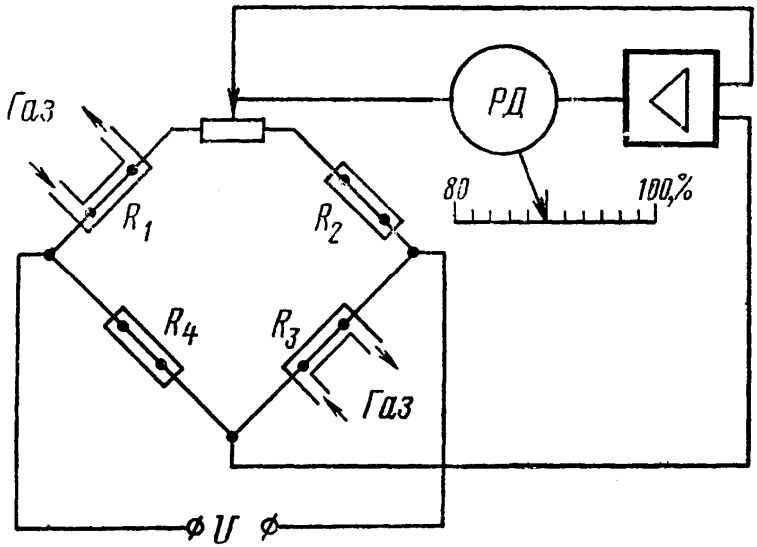
Ультразвуковой метод основан на различии в затухании или скорости распространения ультразвуковых колебаний в различных жидкостях или газах; применяется для анализа органических смесей или водородосодержащих газов (в водороде ультразвук распространяется в 4 раза быстрее, чем в воздухе).

Радиоспектрометрические методы – метод ядерного магнитного резонанса и электронного парамагнитного резонанса, микроволновая спектроскопия. Эти методы требуют слишком много времени и места для их изложения и рассматриваются в соответствующих курсах физики, а здесь мы ограничимся только их упоминанием.

Электрооптические методы основаны на избирательном поглощении, излучении или рассеянии различными компонентами анализируемого вещества светового излучения в видимом, инфракрасном, ультрафиолетовом диапазонах. Например, ***метод инфракрасной спектроскопии (оптико-акустический метод)*** использует избирательное поглощение различными газами инфракрасной радиации, модулированной низкой частотой, и преобразование возникающих акустических колебаний в электрические сигналы с помощью микрофона; метод широко применяется для анализа большинства двухатомных газов и паров (кроме H_2, O_2, N_2, Cl_2).

Радиоактивные методы основаны на различии в ин-

тенсивности поглощения или отражения рентгеновского и радиоактивного излучений компонентами анализируемого вещества; используются для определения состава бинарных жидкостей, концентрации тяжелых элементов в растворах, измерения влажности грунтов, торфа, стройматериалов. Для измерения влажности часто используют ***нейтронный метод***, основанный на способности ядер водорода замедлять быстрые нейтроны (излучаемые, например, полоний-бериллиевым источником), превращая их в тепловые, регистрируемые, например, с помощью газоразрядных счетчиков.



Тепловые методы анализа основаны на измерении тепловых свойств вещества или на определении температурных измерений при различных физико-химических превращениях вещества. Наиболее распространен ***метод теплового газового анализа,*** использующий зависимость теплопроводности газовой смеси от состава и концентрации компонентов. Его применяют для измерения концентрации H_2, He, CO_2, SO_2, Cl_2, теплопроводность которых значительно отличается от теплопроводности других газов; ме-

тод применим для измерения одного компонента при неизменном составе остальных. В качестве измерительных преобразователей газоанализаторов обычно применяют нагреваемые электрическим током платиновые терморезисторы – изменение концентрации измеряемого компонента приводит к изменению теплоотдачи, температуры и сопротивления резистора. Схема такого газоанализатора приведена на рисунке: терморезисторы R_1, R_3 помещены в камеры и обтекаются газом, а в другие плечи моста включены такие же терморезисторы, но помещенные в герметические камеры с газовой смесью с известной концентрацией измеряемого компонента, соответствующей началу шкалы прибора.

Магнитные методы анализа применяются для определения концентрации компонентов с отличающимися от остальных компонентов смеси магнитными свойствами – например, для анализа кислорода, обладающего повышенной магнитной восприимчивостью, выпускают магнитные газоанализаторы.

4. Основные задачи исследования автоматических систем

С общей задачей управления связаны три основные ее составляющие:

1) ***Задача идентификации математической модели*** управляемого процесса возникает, если математические соотношения, связывающие входы и выходы процесса, неизвестны. Эти соотношения могут быть определены теоретически с использованием основных законов соответствующей прикладной области или экспериментально (что чаще всего и случается, по крайней мере, в технических системах).

Наиболее распространенным методом построения теоретических моделей является ***метод анализа баланса*** – сил, электрических напряжений или токов, финансовых потоков, потоков жидкостей, газов или любых других материальных рабочих сред и пр.

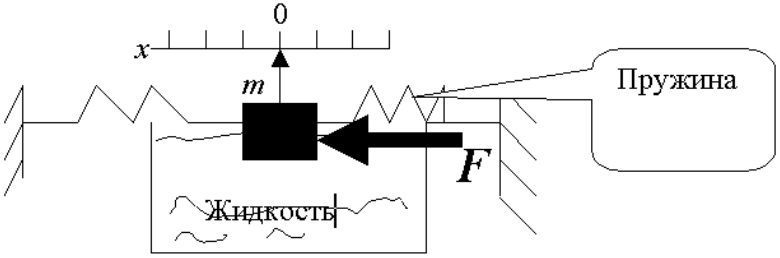
Рассмотрим некоторые примеры ***теоретического решения задачи идентификации***.

а) Пусть груз массой m перемещается в вязкой жидкости и центрируется в нейтральном положении (x=0) пружинами с линейной характеристикой. Управляющее воздействие – сила $F(t)$**,** приложенная к грузу, выходная величина – перемещение груза x. В соответствии с законами механики входное воздействие уравновешивается сопротивлением деформируемых пружин $k_{\text{п}}x$, возникающей во время движения силой вязкого трения $r\dfrac{dx}{dt}$ (r – коэффициент вязкого трения) и силой инерции $m\dfrac{d^2x}{dt^2}$ (m – масса тела).

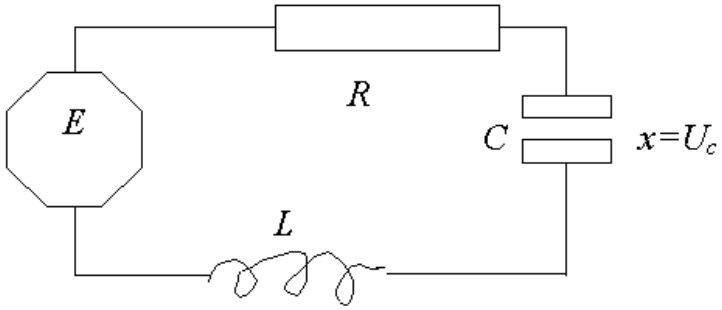
Уравнение процесса движения тела имеет вид:

$$m\frac{d^2x}{dt^2}+r\frac{dx}{dt}+k_{\text{п}}x=F(t)$$

и представляет собой линейное дифференциальное уравнение с коэффициентами, которые могут быть постоянными или зависящими от времени.



б) Пусть к источнику электрического тока подключена электрическая цепь из включенных последовательно омического сопротивления, конденсатора и катушки индуктивности.



Напряжение на конденсаторе будем считать выходом процесса, $E(t)$ – управлением, внутреннее сопротивление источника пренебрежимо мало. В соответствии с законами Кирхгофа сумма падений напряжений в контуре уравновесит ЭДС источников и уравнение процесса формально не отличается от случая механического движения в предыдущем примере:

$$L\frac{d^2x}{dt^2}+R\frac{dx}{dt}+(1/C)\text{x}=E(t).$$

в) Пусть математическая модель строится для определения высоты подъема ракеты при различных углах запуска к горизонту. Обозначим горизонтальную координату полета x, вертикальную y, в момент старта $x(0)=f(0)=0$. Будем считать, что полет происходит в одной вертикальной плоскости (при отсутствии бокового ветра).

Основой для составления математической модели является в этом случае второй закон Ньютона $d(m\boldsymbol{v})/dt=\boldsymbol{F}$, где $m(t)$ – масса ракеты, изменяющаяся во времени из-за расхода топлива, $\boldsymbol{v}$ – вектор скорости с горизонтальной $x^{(1)}(t)$ и вертикальной $y^{(1)}(t)$ составляющими, модулем

$$\upsilon(t)=[(x^{(1)}(t)^2+y^{(1)}(t)^2]^{1/2}$$

и углом к горизонту

$$\theta(t)=\mathrm{arctg}(y^{(1)}(t)/x^{(1)}(t)).$$

$\boldsymbol{F}$ – сумма действующих на ракету сил: силы тяги $\boldsymbol{T}(t)$, силы гравитации $m\boldsymbol{g}$ и силы сопротивления $c\rho s\upsilon^2$, пропорциональной плотности воздуха ρ, поперечному сечению ракеты s и квадрату скорости. Запишем уравнение движения ракеты через координаты x и y с учетом того, что силы тяги и сопротивления действуют вдоль оси ракеты и в сумме дают $T–c\rho s\upsilon^2$:

$$m^{(1)}x^{(1)}+mx^{(2)}=(T–c\rho s\upsilon^2)\cos\theta,$$
$$m^{(1)}y^{(1)}+my^{(2)}=(T–c\rho s\upsilon^2)\sin\theta–mg.$$

Или:

$$x^{(2)}=(1/m)(T–c\rho s\upsilon^2/2)\cos\theta–(m^{(1)}/m)x^{(1)},$$
$$y^{(2)}=(1/m)(T–c\rho s\upsilon^2/2)\sin\theta–(m^{(1)}/m)y^{(1)}–g.$$

Получили систему двух нелинейных дифференциальных уравнений второго порядка с начальными условиями $\upsilon(0)=0$, $\theta(0)=0$. В системе только один свободный параметр θ_0 и его изменение будет определять траекторию.

Если моделируется полет простого снаряда с заданной начальной скоростью, отсутствием тяги и изменения массы, то уравнения значительно упрощаются:

$$x^{(2)}=(1/2m)(–c\rho s\upsilon^2/2)\cos\theta,$$

$y^{(2)}=(1/2m)(-c\rho s\upsilon^2/2)\sin\theta-g$ при $\upsilon(0)=\upsilon_0$, $\theta(0)=\theta_0$.

г) Пусть моделируется задача о популяции двух видов по типу хищник-жертва. Скорости изменения численности жертв $x^{(1)}$ и хищников $y^{(1)}$ определяются их численностями x и y и вероятностью встречи между собой, то есть произведением xy:

$$dx/dt=ax+bxy,\ dy/dt=cy+dxy\ (a>0,\ b<0,\ c<0,\ d>0)$$

с некоторыми начальными численностями $x(0)=x_0$, $y(0)=y_0$. Эти уравнения известны под названием уравнений Лотки-Вольтерра.

Задача ***экспериментальной идентификации математической модели*** требует (помимо наличия экспериментальных данных о реакциях объекта на различные возмущения) привлечения достаточно сложных математических методов обработки экспериментальных данных. Даже в простейших случаях экспериментальной идентификации математических моделей линейных динамических систем, обладающих удобным для обработки результатов ***свойством суперпозиции*** (результирующая реакция равна сумме реакций на отдельные возмущения), задача оказывается достаточно сложной. Рассмотрим вкратце предположительную схему ее решения.

Предположим, что мы воздействовали на исследуемый процесс одним из стандартных возмущений (кратковременный импульс, ступенька или синусоида) и записали через равные промежутки времени значения реакции на это возмущение. Будем считать также, что нам известен априори порядок подлежащего экспериментальному восстановлению линейного дифференциального уравнения объекта (или по характеру переходного процесса есть достаточно оснований выдвинуть правдоподобную гипотезу о величине этого порядка). Из теории линейных систем (мы рассмотрим некоторые ее основы ниже) известно, что реакция системы будет представлять собой сумму экспонент вида:

$$f(t)= A_0e^{p_0t} + A_1e^{p_1t} + ... + A_{k-1}e^{p_{k-1}t}.$$

Имея множество равноотстоящих узлов $t=t_j$ (j=1, …, n), можем без ограничения общности считать $t_j=j$.

Если все члены $e^{p_i t}$ (i=0, 1, …, k–1) удовлетворяют некоторому разностному уравнению k-го порядка с постоянными коэффициентами, то характеристические корни этого уравнения равны ρ^{p_i}. Следовательно, $f(t)$ также удовлетворяет этому разностному уравнению, и оно может иметь такой вид:

$$f(j)+c_1 f(j+1)+\ldots+c_k(j+k)=0 \ (j=0, 1, 2, \ldots).$$

Если мы имеем столько таких уравнений, сколько неизвестных c_m (m=1, 2, …, k) и определитель $|f(j+n)|$ не равен нулю, то можно решить уравнения для c_j, а, зная c_j, мы можем записать характеристическое уравнение

$$\rho^k+c_1\rho^{k-1}+\ldots+c_k=0$$

и из его корней найти p_i. При известных p_i можно решить первые k уравнений для A_i. Таким образом, $2k$ равномерно расположенных узлов f(t) определяют $2k$ неизвестных p_i и A_i.

Если имеется больше чем $2k$ узлов, то можно использовать метод наименьших квадратов, получить систему нормальных уравнений и из них найти по очереди p_i и A_i.

На практике часто число членов в аппроксимирующей функции неизвестно и его надо определить – при этом решение задачи значительно усложняется и не всегда может быть получено с достаточной степенью точности. Отступать перед трудностями не рекомендуется, но удовлетворительные общие методы найти в литературе нам не удалось – наиболее существенные трудности возникают при попытке оценить численно степень подобия между аппроксимирующей и аппроксимируемой функциями.

Совершенно очевидно, что задача экспериментальной идентификации нелинейных нестационарных систем в условиях помех – задача на много порядков большей сложности, но рассмотрение методов ее решения не входит в наш краткий вводный курс.

2) ***Задача анализа.*** Если математическая модель процесса идентифицирована, она позволяет осуществить имитационные исследования процесса – задаваясь различными функциями управления, можно решением описывающих процесс дифференциальных уравнений получить функции, описывающие реакцию процесса на эти управления. Эта задача является вспомогательной для решения основной задачи – ***задачи управления***.

Задача анализа может рассматриваться в различных постановках.

Если заданы значения выходной величины процесса и всех ее производных до (n–1)-й включительно (n – порядок старшей производной в дифференциальном уравнении процесса) в некоторый момент времени, принимаемый начальным, при отсутствии управляющего воздействия – это ***задача определения свободного движения процесса***. Она рассматривается на полубесконечном интервале времени и имеет смысл, если хотя бы одна из n–1 производных не равна нулю – в противном случае движение отсутствует и выход процесса постоянен. Эта же задача может решаться и при наличии управляющего воздействия – в этом случае решение дифуравнения представляет собой комбинацию из собственного движения и вынужденного под действием управления. Обе подгруппы задач с начальными условиями носят название ***задачи Коши для дифференциальных уравнений***.

Если процесс анализируется на ограниченном интервале времени и часть условий заданы на левой границе интервала, а часть на правой (общее количество дополнительных условий должно быть равно порядку уравнения), мы имеем дело с так называемой ***граничной задачей*** определения такого решения, которое удовлетворяет заданным условиям на обеих границах.

3) ***Задача управления.*** В задаче управления ставится задача найти управление, переводящее процесс из произ-

вольного (известного) состояния, характеризуемого значениями выходной функции и ее производных в заданное конечное состояние, тоже заданное значением выходной функции и ее производных. Очевидно, что при избыточном количестве дополнительных условий для получения единственного решения необходимо определить, какое из множества возможных управлений надо считать лучшим, чем остальные. В этой постановке мы приходим к классу ***задач оптимального управления***. Например, при управлении движением ракеты можно в одних случаях считать наилучшей такую управляющую функцию, которая обеспечит вывод ракеты в заданную точку с наименьшим количеством израсходованного топлива (управление, оптимальное по затратам ресурсов); в других случаях наилучшим может считаться управление, выводящее ракету в заданную точку за кратчайшее время (управление, оптимальное по быстродействию).

Управляющее устройство и управляемый объект вместе представляют собой систему, описываемую общей системой дифференциальных уравнений. Параметры объекта (им соответствуют коэффициенты уравнения движения), как правило, не поддаются целенаправленному изменению в процессе управления – они определяются его конструкцией и могут сами дрейфовать в процессе старения объекта во время эксплуатации.

Но параметры другой составляющей системы – управляющего устройства – могут изменяться и это обстоятельство привело к разработке способа управления, основанного на изменении структуры системы в процессе ее движения – это так называемые ***системы с переменной структурой***. Проиллюстрировать это можно следующим примером. Пусть в электрическом контуре зарядки конденсатора с последовательно включенной индуктивностью и омическим сопротивлением у нас есть возможность изменять это сопротивление. Если мы сделаем его маленьким, процесс

будет колебательным и его выход на заданный уровень будет длительным. Если сделать его настолько большим, что процесс зарядки станет апериодическим, то длительность процесса тоже будет большой. Но можно применить следующий прием – при большом отклонении напряжения на емкости сделать сопротивление маленьким, что обеспечит движение в автоколебательном режиме с большой скоростью, а при приближении к заданному уровню переключиться на большое сопротивление и небольшое оставшееся рассогласование отработать уже в апериодическом режиме – мы получим процесс зарядки с коротким временем перехода в заданное состояние.

Выводы. Итак, в математике процесс – это функция, описывающая изменение значений управляемых величин процесса в зависимости от управляющих и возмущающих воздействий. Задачи, связанные с решением дифференциальных уравнений, чаще всего возникают при математическом моделировании и исследовании на модели динамики процессов в различных объектах или системах. При этом входное управляющее или возмущающее воздействие на систему задается в виде некоторой, в общем случае произвольной, функции времени в правой части неоднородного дифференциального уравнения. Вызванное этим воздействием изменение процесса на выходе системы называют ***вынужденным движением***.

Изменение выходной величины во времени при снятом входном воздействии зависит от конструктивных особенностей системы и начальных условий и носит название ***собственного движения***, определяемого как решение однородного уравнения без правой части. Для линейных систем результирующее движение представляет собой сумму собственного и вынужденного движений в предположении, что возмущающее воздействие приложено в момент t=0 и отсутствует в предшествующие моменты, а для t=0 заданы значения выходной координаты и ее производных.

5. Операционное исчисление и его применение к исследованию динамики стационарных линейных систем

5.1. Общие сведения

Мы уже отмечали, что наиболее просто вычисляются решения линейных дифференциальных уравнений; для линейного уравнения первого порядка это решение – экспонента, для уравнения n-го порядка – сумма экспонент. Этот тип дифференциальных уравнений и описываемых с их помощью линейных (или искусственно линеаризованных для упрощения) систем широко используется на практике при решении всех перечисленных классов задач – идентификации, анализа и управления.

Еще в конце 19-го века английский физик О. Хевисайд предложил способ вычислений, который назвал ***операционным***. Он рассматривал знак дифференцирования d/dt как оператор p и операцию дифференцирования записывал как $pf(t)$, n-я производная записывалась как $p^n f(t)$ – то есть оператор p n раз прилагался к функции $f(t)$. Оператор $1/p$ или p^{-1} представлял операцию интегрирования, так как приложение оператора p к p^{-1} снова давало $f(t)$: $pp^{-1}f(t)=f(t)$. При этом формулы с дифференциалами и интегралами приводились к алгебраической форме – Хевисайд называл это алгебраизацией задачи. До работ Карсона и Леви, давших методу фундаментальное математическое основание, обращаться с оператором p как с алгебраическим числом в вычислениях было небезопасно – в этом приеме скрывалось множество скрытых ловушек и только гениальная интуиция Хевисайда спасала его от ошибок в вычислениях.

Современное операционное исчисление рассматривает либо функции, связанные интегральным преобразованием Карсона

$$F_K(p)=p\int_0^\infty f(t)e^{-pt}dt\ ,$$

либо преобразованием Лапласа:

$$F_L(p)=\int_0^\infty f(t)e^{-pt}dt\ .$$

В результате обоих преобразований функция $f(t)$ вещественного переменного t преобразуется в функцию $F(p)$ комплексного аргумента $p=\sigma+i\omega$, (σ и ω – вещественные, $i=\sqrt{-1}$).

Между функциями $F_K(p)$ и $F_L(p)$ очевидна взаимосвязь вида

$$pF_L(p)=F_K(p).$$

Преобразование Карсона удобно использовать в анализе электрических цепей, а мы будем использовать преобразование Лапласа (опуская индекс L в обозначении) в соответствии со сложившейся практикой в большинстве прикладных областей.

Приведенное функциональное соотношение записывают в виде $F(p)\subset f(t)$ или $f(t)\supset F(p)$ и говорят, что «$F(p)$ есть изображение $f(t)$» или «$f(t)$ есть оригинал $F(p)$». Достаточным условием существования изображения функции $f(t)$ является требование ее кусочной непрерывности и существования таких положительных чисел M и σ, чтобы $|f(t)|<Me^{\sigma t}$.

Рассмотрим некоторые правила операционного исчисления.

Сложение. Так как преобразование Лапласа – линейная операция, то *изображение суммы равно сумме изображений* $\sum_i F_i(p)\subset\sum_i f_i(t)$. Справедливо и обратное – оригинал суммы равен сумме оригиналов.

Дифференцирование f(t). Умножим на p обе части преобразования Лапласа для $f(t)$ и проинтегрируем по частям:

$$pF(p)=\int_0^\infty pe^{-pt}f(t)dt=[-e^{-pt}f(t)]_0^\infty+\int_0^\infty e^{-pt}f^{(1)}(t)dt,$$

то есть

$$pF(p)–f(0)\subset f^{(1)}(t).$$

Если $f(0)=0$, то

$$pF(p)\subset f^{(1)}(t).$$

Повторив n раз тот же прием, получим последовательным интегрированием по частям

$$p^nF(p)–p^{n-1}f(0)–p^{n-2}f^{(1)}(0)–\ldots–pf^{(n-2)}(0)–f^{(n-1)}(0)\subset f^{(n)}(t).$$

Если $f(0)=f^{(1)}(0)=\ldots=f^{(n-1)}(0)$, то $p^nF(p)\subset f^{(n)}(t)$.

Интегрирование f(t).

$$\frac{F(p)}{p}\subset\int_0^t f(t)dt \text{ и } \frac{F(p)}{p^n}\subset\int_0^t dt\int_0^t dt...\int_0^t f(t)dt,$$

то есть дифференцирование и интегрирование $f(t)$ соответствует соответственно умножению и делению изображения на p.

Теорема смещения. $F(p+\lambda)\subset e^{-\lambda t}f(t)$.

Теорема запаздывания. $e^{-\lambda p}F(p)\subset f(t–\lambda)\,\Upsilon(t–\lambda)$, где $\Upsilon(t–\lambda)$ – единичная ступенчатая функция. Если $f(t)=\Upsilon(t–\lambda)$, то ее изображение для запаздывания λ будет $\frac{1}{p}e^{-\lambda p}\subset\Upsilon(t-\lambda)$.

Теорема разложения Хевисайда. Теория разложения рациональных функций на простые дроби показывает, что если знаменатель $P(p)$ – полином m-й степени с только простыми корнями a_n, а числитель $Q(p)$ – любой полином более низкой степени, то имеет место тождество

$$\frac{Q(p)}{pP(p)}\equiv\frac{Q(0)}{pP(0)}+\sum\frac{Q(a_n)}{a_nP^{(1)}(a_n)}\cdot\frac{1}{p-a_n}$$

(суммирование по n от 1 до m).

Так как $\frac{1}{p-a_n} \subset e^{a_n t}$, то для функции $f(t)$, оригинал которой соответствует изображению $f(t) \supset \frac{Q(p)}{pP(p)}$, получим:

$$f(t)=\frac{Q(0)}{P(0)}+\sum_{n=1}^{m}\frac{Q(a_n)}{p_n P^{(1)}(a_n)}e^{a_n t}.$$

Если $f(t) \supset \frac{Q(p)}{P(p)}$, то только для простых корней

$$f(t)=\frac{Q(0)}{P(0)}+\sum_{n=1}^{m}\frac{Q(a_n)}{P^{(1)}(a_n)}e^{a_n t}.$$

Для случая кратных корней формула имеет более сложный вид:

$$\frac{Q(p)}{P(p)} \subset \sum_{k=1}^{r}\sum_{j=1}^{n_k}\frac{A_{kj}}{(n_k-j)!}t^{n_k-j}e^{a_k t}, \text{ где}$$

$$A_{kj}=\frac{1}{(j-1)!}\left[\frac{d^{j-1}}{dp^{j-1}}\frac{(p-a_k)^{n_k}Q(p)}{P(p)}\right]_{p=a_k}. \quad (*)$$

r – количество разных корней, n_k – кратность k-го корня, k – текущий номер корня, j – текущая кратность.

Теорема свертывания (Бореля).

Даны две функции – $f_1(t)$ и $f_2(t)$ с изображениями $F_1(p)$ и $F_2(p)$ соответственно. Оригинал произведения изображений $F_1(p)$ и $F_2(p)$ будет равен интегралу произведения функций по параметру τ при смещении аргумента одной из них на величину τ.

Если $F_1(p) \subset f_1(t)$, $F_2(p) \subset f_2(t)$, то

$$F_1(p)F_2(p) \subset \int_0^t f_1(\tau)f_2(t-\tau)d\tau = \int_0^t f_1(t-\tau)f_2(\tau)d\tau.$$

Изображения типовых возмущающих (управляющих) функций.

Ступенчатая функция	$\gamma \supset 1/p$
Единичный импульс	$\gamma^{(1)} \supset 1$
Синусоида	$\sin\omega t \supset \omega/(p^2+\omega^2)$
Косинусоида	$\cos\omega t \supset p/(p^2+\omega^2)$
Экспонента	$\exp(-\omega t) \supset 1/(p+\omega)$

5.2. Решение линейных уравнений с постоянными коэффициентами

Уравнение имеет вид:

$$a_n f^{(n)}(t)+a_{n-1}f^{(n-1)}(t)+\ldots.+a_1 f(t)+a_0=u(t).$$

Начальные условия: $f(0)=f_0$, $f^{(i)}(0)=f_i$, $i=1, 2, \ldots, n-1$.

Применим к функциям $f(t)$, $u(t)$ и их производным преобразование Лапласа, обозначив изображения $f(t)$ через $F(p)$, а $u(t)$ через $U(p)$, получим алгебраическое уравнение

$$F(p)[a_n p^n+a_{n-1}p^{n-1}+\ldots.+a_1 p+a_0]=U(p)+f(0)[a_n p^{n-1}+ +a_{n-1}p^{n-2}+\ldots.+a_1]+f^{(1)}(0)[a_n p^{n-2}+a_{n-1}p^{n-3}+\ldots.+a_2]+\ldots+ +f^{(n-1)}(0)[a_0]$$

или в свернутом виде

$$F(p)\sum_{i=0}^{n} a_i p^i = U(p)+\sum_{i=0}^{n-1}\left[f^{(i)}(0)\sum_{j=1}^{n-i} a_{n-j+1}p^{n-j-i}\right].$$

Обозначим слагаемое, обусловленное ненулевыми начальными условиями

$$N(p)=\sum_{i=0}^{n-1}\left[f^{(i)}(0)\sum_{j=1}^{n-i} a_{n-j+1}p^{n-j-i}\right],$$

сумму $Q(p)=U(p)+N(p)$, а полином $\sum_{i=0}^{n} a_i p^i = P(p)$ назовём характеристическим полиномом.

Тогда изображение решения дифференциального уравнения

$$F(p)=\frac{Q(p)}{P(p)}.$$

Используя общую формулу разложения, получим ре-

шение уравнения:

$$\frac{Q(p)}{P(p)} \subset \sum_{k=1}^{r} \sum_{j=1}^{n_k} \frac{A_{kj}}{(n_k - j)!} t^{n_k - j} e^{a_k t} = f(t),$$

где $A_{kj} = \frac{1}{(j-1)!} \left[\frac{d^{j-1}}{dp^{j-1}} \frac{(p - a_k)^{n_k} Q(p)}{P(p)} \right]_{p=a_k}$.

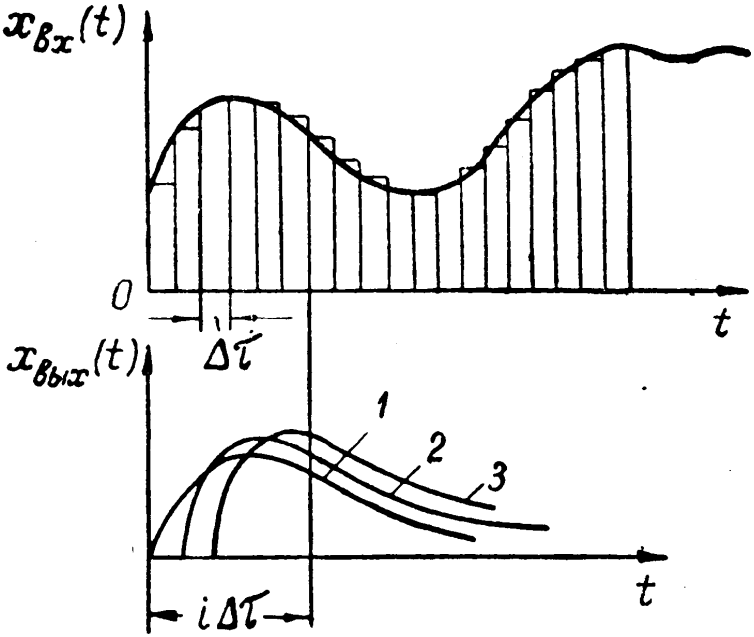
Здесь мы встретились с одним из положительных качеств символического метода – начальные условия вводятся сразу, еще при постановке задачи и не вызывают никаких осложнений при ее решении. При решении однородного уравнения опускается управляющая функция, а если нас интересует только вынужденное движение, то задаются нулевые начальные условия.

Единственная трудность, которая нас поджидает, состоит в получении изображения управляющей функции $u(t)$, да еще и желательно в виде полинома или рациональной полиномиальной дроби. Если это одна из табулированных функций или ее преобразование Лапласа находится достаточно легко, то проблема решается.

Если же это произвольная функция времени, то при компьютерном решении она будет задана последовательностью своих значений на конечном временном интервале. Общее решение ищется в этом случае следующим образом.

Заменим вначале нашу управляющую функцию единичным импульсом, имеющим изображение 1, и найдем общее решение системы в виде реакции на этот импульс. Обозначим это решение через $h^{(1)}(t)$.

Если предположить, что дискретные значения управляющей функции взяты через достаточно малые интервалы аргумента $\Delta\tau$, то эту функцию можно приближенно заменить последовательностью прямоугольных импульсов продолжительностью $\Delta\tau$ и амплитудой $u(i\Delta\tau)$ где i – порядковый номер значения, а реакцию системы на каждый из прямоугольных импульсов заменить реакцией на импульсную функцию $A_i Y^{(1)}$, где $A_i = u(i\Delta\tau)\Delta\tau$.



Приближенное вычисление реакции на произвольное входное воздействие

Если реакция системы на $Y^{(1)}$ есть $h^{(1)}(t)$, то реакция на i-й прямоугольный импульс будет приближенно равна $h^{(1)}(t\text{-}i\Delta\tau)u(i\Delta\tau)\Delta\tau$, причем она будет существовать только для $t \geq i\Delta\tau$, так как реакция не может предшествовать воздействию. Реакция системы в момент времени $t=n\Delta\tau$ будет равна сумме реакций от каждого предшествующего импульса:

$$f(t) \approx \sum_{i=1}^{n} h^{(1)}\left(t - i\Delta\tau\right)u\left(i\Delta\tau\right)\Delta\tau \, .$$

При желании можно перейти к предельному выражению, считая что $\Delta\tau \to d\tau$ и прямоугольный импульс стремится к $Y^{(1)}$, величина $i\Delta\tau$ стремится к непрерывной величине τ, а сумма – к интегралу, дающему точное значение τ:

$$f(t)=\int_0^{\tau} h^{(1)}(t-\tau)u(\tau)d\tau\ .$$

Этот интеграл – свертка функций $h^{(1)}(t)$ и $u(t)$ или ***интеграл Дюамеля*** (мы уже упоминали о нем под названием теоремы свертывания Бореля); функции под интегралом можно поменять местами и представить интеграл в виде

$$f(t)=\int_0^{\tau} h^{(1)}(\tau)u(t-\tau)d\tau\ .$$

Примеры интегрирования линейных дифференциальных уравнений с постоянными коэффициентами операторным методом.

1) Дифференциальное уравнение

$$\frac{d^2y}{dt^2}+4\frac{dy}{dt}+3y=1$$

при начальных условиях при $t=0$ $y_0=3$, $y_0^{(1)}=-2$ дает уравнение в изображениях вида

$$(p^2+4p+3)Y(p)=1+p^2y_0+py_0^{(1)}+4py_0=3p^2+10p+1,$$

откуда

$$Y(p)=\frac{3p^2+10p+1}{p^2+4p+3}.$$

Оригинал $y(t)$ для изображения $Y(p)$ найдется по теореме разложения, в нашем случае корни характеристического полинома p^2+4p+3 равны $p_1=-1$, $p_2=-3$ и

$$y(t)=\frac{F_1(0)}{F_2(0)}+\sum_{k=1}^{n}\frac{F_1(p_k)}{p_kF_k^1(p_k)}e^{p_kt}=1/3+3e^{-t}-1/3e^{-3t}.$$

2) Уравнение

$$dy/dt+3y=e^{-2t}$$

при начальном условии $y_0=0$ в изображениях имеет вид

$$(p+3)Y(p)=p/(p+2),$$

изображение решения

$$Y(p)=p/((p+2)(p+3)),$$

а оригинал

$$y(t)=e^{-2t}-e^{-3t}.$$

3) Дифференциальное уравнение

$$\frac{d^2y}{dt^2}+2\frac{dy}{dt}+2y=\cos(t)$$

с начальными условиями $y_0=0,\ y_0^{(1)}=0$ в изображениях имеет вид

$$(p^2+2p+2)Y(p)=\frac{p^2}{p^2+1},$$

изображение решения

$$Y(p)=\frac{p^2}{(p^2+1)(p^2+2p+2)},$$

корни характеристического полинома $p_1=i$, $p_2=-i$, $p_3=-1+i$, $p_4=-1-i$ и по теореме разложения

$$y(t)=\frac{e^{it}}{2(1+2i)}+\frac{e^{-it}}{2(1-2i)}+\frac{(-1+i)e^{(-1+i)t}}{2i(1-2i)}+\frac{(1+i)e-^{(1+i)t}}{2i(1+2i)}$$

после некоторых преобразований получаем

$$y(t)=\frac{\cos t}{5}(1-e^{-t})+\frac{\sin t}{5}(2-3e^{-t}).$$

6. Передаточные функции линейных динамических систем

Благодаря операционному исчислению стало возможным перейти от классических методов количественного описания динамических свойств линейных систем в виде дифференциальных уравнений к более экономным средствам – передаточным функциям, временным и частотным характеристикам.

Передаточную функцию можно трактовать как ***комплексный коэффициент преобразования*** входного воздействия динамической системы в ее реакцию на выходе. Для формирования передаточной функции дифференциальное уравнение системы относительно функций вещественного переменного преобразуют в уравнение для функций комплексного переменного с использованием рассмотренных интегральных преобразований.

Если ввести понятие передаточной функции динамической системы в форме преобразования Лапласа $W(p)$ как ***отношение изображения выходной функции*** $F(p)$ ***к изображению входной (управляющей)*** $U(p)$ при нулевых начальных условиях $W(p)=F(p)/U(p)$, то оказывается, что изображение интеграла Дюамеля, являющегося изображением выходной функции, равно произведению передаточной функции на изображение управляющей функции.

Сложные динамические объекты редко идентифицируются сразу как единое целое – обычно получают их описание по частям или звеньям, а затем находят общее описание системы как описание соединения отдельных звеньев.

В этом случае передаточные функции оказываются удобным инструментом определения общего описания линейной системы. Вначале рассмотрим примеры передаточных функций некоторых элементарных звеньев.

Рассмотренные нами ранее уравнения 2-го порядка для механической колебательной системы или электрического

колебательного контура, записанные в общем виде как

$$a_2\frac{d^2x}{dt^2}+a_1\frac{dx}{dt}+a_0x=ku,$$

приводят к передаточной функции вида

$$W(p)=\frac{k}{a_2p^2+a_1p+a_0}.$$

Если масса механической системы или индуктивность электрической цепи пренебрежимо малы, то приведенное уравнение вырождается в уравнение первого порядка $a_1\frac{dx}{dt}+a_0x=ku$ и его передаточная функция $W(p)=k/(a_1p+a_0)$.

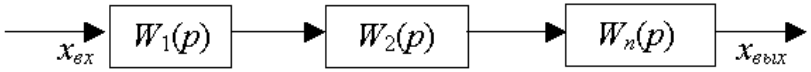
Уравнение интегрирующего звена $Tdx/dt=u$, его решение – $x=\int_0^t udt$, а передаточная функция $W(p)=1/Tp$.

Звено с постоянным запаздыванием (типа, например, конвейера) описывается уравнением $x(t)=u(t-\tau)$ и его передаточная функция $W(p)=e^{-p\tau}$.

В управляющих устройствах систем управления используют звенья, имеющие передаточные функции, обратные передаточным функциям колебательного и инерционного звеньев, например, форсирующее звено первого порядка с передаточной функцией $W(p)=Tp+1$, или форсирующее звено второго порядка с передаточной функцией $W(p)=a_2p^2+a_1p+1$, или дифференцирующее звено $W(p)=ap$.

Звенья в системе могут соединяться ***последовательно***, ***параллельно*** и ***встречно-параллельно (обратной связью)***.

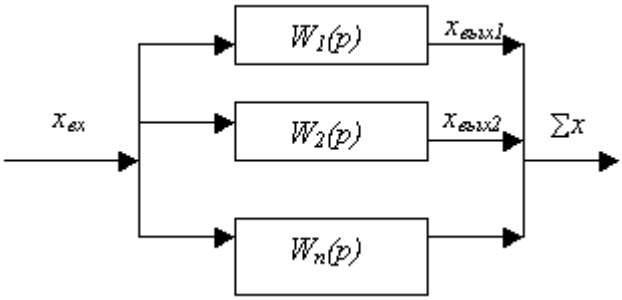
Передаточная функция ***последовательного*** соединения звеньев равна произведению входящих в цепочку звеньев:



$$W(p)=\prod_{i=1}^{n}W_i(p).$$

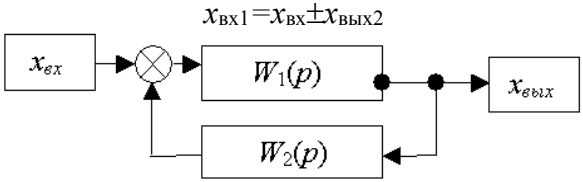
При ***параллельном*** соединении звеньев результирую-

щая передаточная функция равна сумме передаточных функций входящих в соединение звеньев:



$$W(p)=\sum_{i=1}^{n} W_i(p).$$

И, наконец, соединение двух звеньев ***по принципу обратной связи***, когда



может быть получена так:

$$x_{вых}=W_1(p)x_{вх1};\ x_{вых2}=W_2(p)x_{вых}$$

$$W(p)=\frac{W_1(p)}{1\mp W_1(p)W_2(p)}.$$

При этом верхний знак «–» относится к положительной обратной связи, а нижний «+» к отрицательной.

Обратную связь принято называть ***жесткой***, если $W_2(p)$=const, то есть звено в обратной связи есть простой усилитель. Нежесткие обратные связи имеют ряд разновидностей – ***гибкие, изодромные, скоростные, запаздывающие*** и пр.

7. Частотные характеристики линейных динамических систем

Эти характеристики – эффективный инструмент исследования свойств системы, позволяющий определить, как подавляются высокие или усиливаются резонансные частоты, как сдвигаются по фазе входные гармоники при прохождении через систему. Для получения частотной характеристики необходимо найти частное решение неоднородного уравнения системы при входном воздействии

$$u(t)=A_{\text{вх}}\,e^{j(\omega t+\varphi)},$$

где $u(t)$ – комплексная величина, которую на комплексной плоскости можно изобразить в виде вектора, образующего с вещественной осью угол $\omega t+\varphi$, линейно возрастающий в функции t; поэтому вектор вращается против часовой стрелки с угловой скоростью ω.

Установившееся движение на выходе линейной передающей системы – гармонические колебания с частотой входных и частное решение уравнения системы ищется в форме входного воздействия, то есть $f(t)=A_{\text{вых}}(t)\,e^{j(\omega t+\varphi_{вых})}$ – выходной вектор вращается со скоростью входного, но имеет другой модуль и смещен относительно входного на угол $\varphi=\varphi_{вых}-\varphi_{вх}$. Подстановка указанного решения в уравнение системы и определение отношения $f(t)/u(t)=W(j\omega)=Q(j\omega)/P(j\omega)$ приводят к комплексному коэффициенту передачи или амплитудно-фазовой характеристике системы. Запись последней формулы в показательной форме $W(j\omega)=W(\omega)\,e^{j\varphi(\omega)}$, где $W(\omega)=A_{\text{вых}}/A_{\text{вх}}$ – амплитудно-частотная характеристика, $\varphi(\omega)=\varphi_{\text{вых}}-\varphi_{\text{вх}}$ – фазо-частотная характеристика. Так как $W(j\omega)$ – дробно-рациональная функция, то амплитудно-частотная характеристика вычисляется как отношение модулей числителя и знаменателя и определяется просто: $W(\omega)=\dfrac{|Q(j\omega)|}{|P(j\omega)|}$, а фазовая характери-

стика как разность фазовых углов: $\varphi(\omega)=\text{arctg}\frac{I_Q(\omega)}{R_Q(\omega)}-$ $\text{arctg}\frac{I_p(\omega)}{R_p(\omega)}$.

Частотные характеристики элементарных звеньев.

Усилительное звено.

$W(j\omega)=k$; $W(\omega)=k$; $\varphi(\omega)=0$; $U(\omega)=k$; $V(\omega)=0$.

Годограф $W(j\omega)$ вырождается в точку на вещественной оси, так как при любых частотах $W(j\omega)$ равно единственному значению k. Фазовый сдвиг отсутствует и все частоты входного сигнала усиливаются одинаково.

Запаздывающее звено.

$W(j\omega)=e^{-j}$; $W(\omega)=1$; $\varphi(\omega)=-\omega\tau$, то есть амплитуда на выходе равна входной, а фаза отстает от входной на угол, пропорциональный частоте. Годограф $W(j\omega)$ представляет собой окружность с центром в начале координат.

Инерционное звено.

$$W(j\omega)=\frac{k}{Tj\omega+1};\quad W(\omega)=\frac{k}{\sqrt{T^2\omega^2+1}};\quad \varphi(\omega)=-\text{arctg}\quad T\omega;$$

$$U(\omega)=\frac{k}{T^2\omega^2+1};V(\omega)=\frac{kT\omega}{T^2\omega^2+1}.$$

Усиление звена падает с ростом частоты – при близких к нулю частотах сигнал воспроизводится почти как у усилительного звена, а при высоких звено вообще не пропускает сигнал на выход и подавление высоких частот тем интенсивнее, чем больше постоянная времени T. Фазовая характеристика отрицательна и выходные колебания отстают по фазе от входных, угол отставания при $\omega\to\infty$ стремится к $\pi/2$. При $\omega=1/T$ $\varphi(\omega)=-\pi/4$. Годограф АФХ – окружность с радиусом с центром на вещественной оси на расстоянии $k/2$ от начала координат, положительным частотам соответствует нижняя полуокружность.

Колебательное звено второго порядка.

$$W(j\omega)=\frac{k}{T^2(j\omega)^2+2\zeta Tj\omega+1}=\frac{k\omega_0^2}{\omega_0^2-\omega^2+j2\zeta\omega_0\omega};$$

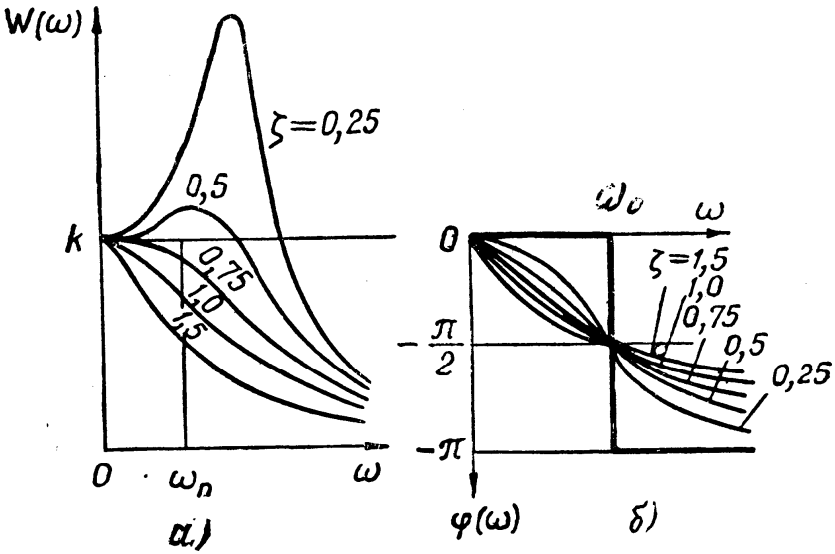
$$W(\omega)=\frac{k}{\sqrt{\left[1-\left(\frac{\omega}{\omega_0}\right)\right]^2+4\zeta^2\left(\frac{\omega}{\omega_0}\right)^2}},\qquad \varphi(\omega)=-\text{arctg}\frac{2\zeta\frac{\omega}{\omega_0}}{1-\left(\frac{\omega}{\omega_0}\right)^2};$$

$$U(\omega)=\frac{\left[1-\left(\frac{\omega}{\omega_0}\right)^2\right]k}{\left[1-\left(\frac{\omega}{\omega_0}\right)^2\right]^2+4\zeta^2\left(\frac{\omega}{\omega_0}\right)^2};$$

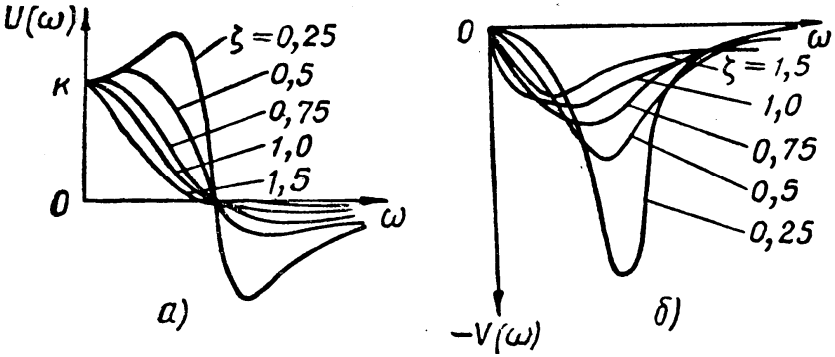
$$V(\omega)=\frac{-2k\zeta\frac{\omega}{\omega_0}}{\left[1-\left(\frac{\omega}{\omega_0}\right)^2\right]^2+4\zeta^2\left(\frac{\omega}{\omega_0}\right)^2}.$$

Здесь $\omega_0=1/T$ – частота собственных колебаний звена, ζ – коэффициент затухания.

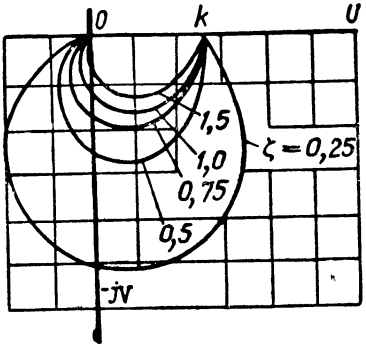
Амплитудные характеристики – типичные резонансные кривые, явление резонанса лучше проявляется при малом затухании ζ при частоте, близкой к собственной ω_0. Максимум кривой $W(\omega)$ наступает при частоте $\omega_p=\sqrt{1-2\zeta^2}$, величина максимума $W(\omega_p)=\frac{k}{2\zeta\sqrt{1-\zeta^2}}$, при $\omega=\omega_0$; $W(\omega_0)=k/2\zeta$.



Амплитудная (а) и фазовая (б) частотные характеристики колебательного звена при различных коэффициентах затухания ζ.



Вещественные (а) и мнимые (б) частотные характеристики колебательного звена при различных коэффициентах затухания ζ.



Годограф $W(j\omega)$ колебательного звена при различных коэффициентах затухания ζ.

Кривая $W(\omega)$ имеет максимум только при $\zeta < \sqrt{2}/2$. При $\zeta = \sqrt{2}/2$ кривая имеет максимум при нулевой частоте, а при больших затуханиях $W(\omega)$ – убывающая функция частоты. При резонансе и слабом затухании усиление во много раз больше коэффициента усиления звена и это широко используется для усиления слабых сигналов в приемных радиоустройствах. В системах авторегулирования, напротив, используют колебательные звенья с сильным затуханием в полосе частот ниже собственной.

Фазовый сдвиг между входом и выходом уменьшается с уменьшением затухания, при $\omega = \omega_0$ и нулевом затухании происходит скачок фазы и при более высоких частотах сдвиг по фазе равен π.

Вещественная частотная характеристика меняет знак в точке $\omega = \omega_0$.

Амплитудно-фазовая характеристика начинается на вещественной оси в точке k, годограф ее вектора захватывает 2 квадранта комплексной плоскости и при бесконечной частоте касается отрицательной вещественной полуоси. При нулевом затухании годограф вырождается в 2 полупрямые на вещественной и мнимой полуосях.

8. Введение в теорию устойчивости линейных стационарных систем авторегулирования

Реакция линейной системы на управляющее или возмущающее воздействие всегда состоит из двух составляющих – собственного и вынужденного движения. Если входные функции имеют дробно-рациональные изображения, то

$$x_{своб}(t)=\sum_{i=1}^{n} c_i e^{p_i t}\ ,\ x_{вын}(t)=\sum_{k=1}^{l} b_k e^{p_k t}$$

где p_i – полюсы передаточной функции (или нули характеристического полинома), p_k – полюсы изображения воздействия.

Система автоматического регулирования может нормально функционировать только в том случае, если собственные или свободные движения, возникающие в силу различных причин, с течением времени затухают до нуля; если система удовлетворяет этому условию, то ее называют ***устойчивой***. Если собственное движение системы расходится, то система ***неустойчива***. Собственное движение представляет собой сумму экспоненциальных составляющих, порожденных корнями характеристического полинома – затухание или незатухание соответствующей компоненты полностью определяется значением соответствующего полюса передаточной функции – вернее, его вещественной частью. Если вещественный корень отрицателен или комплексный корень имеет отрицательную вещественную часть, то соответствующая компонента свободного движения системы затухает, положительность вещественного корня или вещественной части комплексного корня порождают расходящийся процесс. При нулевой вещественной части комплексных корней (при чисто мнимых корнях) порождается колебательная компонента с постоянной амплитудой колебаний – такая система считается нахо-

дящейся ***на границе устойчивости***. Очевидно, что достаточно одной расходящейся компоненты, чтобы система в целом потеряла устойчивость. Поэтому определение устойчивости звучит просто:

> ***Линейная стационарная система устойчива, если все вещественные корни ее характеристического уравнения отрицательны, а все комплексные корни имеют отрицательную вещественную часть. Система, у которой хотя бы один из корней характеристического уравнения располагается левее мнимой оси – неустойчива.***

Для анализа устойчивости системы в теории автоматического регулирования разработан ряд правил, позволяющих анализировать устойчивость систем без решения их характеристических уравнений – их называют ***критериями устойчивости***. Сегодня не составляет труда с помощью компьютерной программы решить уравнение любого порядка и по его корням определить устойчивость соответствующей системы. Но значения корней ничего не говорят о причинах возможно обнаруженной неустойчивости и о том, какие параметры системы надо изменить для обеспечения устойчивости, поэтому разработанные критерии сохраняют свое значение и в настоящее время. Наиболее известные из критериев устойчивости следующие:

Критерий устойчивости Рауса-Гурвица – условия отсутствия у многочлена нулей с положительной вещественной частью формулируются в виде системы неравенств, составленных по коэффициентам многочлена. Неравенства Гурвица записываются в форме определителей (их количество равно порядку уравнения), составленных из коэффициентов многочлена по специальным правилам; доказывается, что полином не имеет нулей в правой полуплоскости, если все определители положительны.

Частотный критерий устойчивости Михайлова. Из характеристического полинома подстановкой вместо аргу-

мента $j\omega$ образуется функция комплексного аргумента $A(\omega)e^{j\varphi(\omega)}$. Критерий формулируется так: *Многочлен является многочленом Гурвица (т.е. система устойчива), если полное приращение фазы при изменении частоты от нуля до бесконечности равно $n\pi/2$, где n – степень полинома.*

Амплитудно-фазовый критерий Найквиста. Позволяет судить об устойчивости замкнутой системы по поведению годографа амплитудно-фазовой характеристики разомкнутой и формулируется так: *Замкнутая система устойчива, если полное приращение аргумента $1+W(j\omega)$ равно $2k\pi/2$, где k – число полюсов передаточной функции разомкнутой системы W(p), находящихся справа от мнимой оси комплексной плоскости.*

Ограничения области применения.

Мы кратко рассмотрели удобный и эффективный метод исследования динамических систем. Это аналитический метод – численные решения мы получаем только для корней характеристического полинома и при вычислении реакции на входное воздействие, заданное в виде дискретной последовательности значений; этот факт показывает, что даже при использовании точных аналитических методов трудно избежать использования приближенных вычислений.

Следует помнить, что символический метод применим только к линейным системам (с сосредоточенными или распределенными параметрами, т.е. с обыкновенными дифференциальными уравнениями или уравнениями в частных производных), для которых справедлив принцип суперпозиции – реакция на сумму воздействий может быть вычислена как сумма реакций на отдельные воздействия.

Существенные трудности возникают и при анализе линейных (относительно функций) систем с переменными коэффициентами – приходится аппроксимировать зависимость коэффициентов от времени, например полиномами, что приводит к появлению производных в уравнениях для

изображений со старшей степенью, равной порядку аппроксимирующего полинома, то есть к дифференциальным уравнениям и тоже с переменными коэффициентами – первоначальное намерение алгебраизовать задачу остается неосуществленным.

Кроме того, реальные системы, как правило, нелинейны – мы показали это на очень упрощенных моделях из области экологии и баллистики; наше обычное стремление привести модель системы к линейной структуре может дать решение, не очень близкое к решению первоначальной задачи. Поэтому разработка эффективных методов для решения нелинейных дифференциальных уравнений будет всегда актуальной вычислительной задачей.

9. О качественном анализе динамических систем

В математике существует понятие ***фазового портрета системы*** – под ним понимают совокупность траекторий решений, описывающих систему уравнений в n-мерном пространстве R_n – его называют ***фазовым пространством системы.*** Так как для пространств размерностью выше двух графическое отображение траекторий становится проблематичным, то реальное построение осуществляют на плоскости для сравнительно простых систем. Но само понятие фазовых траекторий оказывается полезным инструментом анализа динамики систем.

Рассмотрим для примера одномерную механическую систему, описываемую дифференциальным уравнением Ньютона для материальной точки:

$$m\frac{d^2x}{dt^2}=f(x).$$

Запишем его в виде системы двух уравнений первого порядка, введя новую переменную – импульс $p=m\frac{dx}{dt}$ и потенциальную энергию $U(x)=-\int f(x)dx$.

При этом возникает понятие гамильтониана (в нашем случае это просто полная энергия системы, но в других случаях это может быть значительно более сложное и глубокое понятие):

$$H(p, x)=U(x)+\frac{p^2}{2m}$$

и система может быть записана в виде

$$\frac{dp}{dx}=-\frac{\partial H}{\partial x};\ \frac{dx}{dt}=\frac{\partial H}{\partial p}.$$

Теперь можно рассмотреть плоскость (p, x), являющуюся фазовым пространством (но не являющуюся геометри-

ческим пространством) – изображение движения системы будет более простым и понятным, хотя реально оно по-прежнему осуществляется вдоль единственной прямой.

Для простого гармонического осциллятора типа подвешенного на пружине груза полная энергия (гамильтониан):

$$H(p, x)=\frac{kx^2}{2}+\frac{p^2}{2m}.$$

Траектории на фазовой плоскости совпадают с линиями уровня гамильтониана

$$\frac{kx^2}{2}+\frac{p^2}{2m}=E,$$

представляя собой семейства эллипсов $\frac{x^2}{a^2}+\frac{p^2}{b^2}=1$, где

$$a=\sqrt{2E/k};\ \ b=\sqrt{2mE}\ .$$

В общем случае объекта, описываемого системой обыкновенных дифференциальных уравнений

$$x_i^{(1)}=f_i(t, x_1, \ldots, x_n, u_1, \ldots, u_r),\ i=1, \ldots, n$$

с начальными условиями $t=t_0$, $x_i(t_0)=x_{i0}$ при выборе определенного допустимого управления $\boldsymbol{u}(t)$ движение объекта определено единственным образом и конец вектора $\boldsymbol{x}(t)$ опишет траекторию в пространстве своих координат. Эти координаты мы можем считать ***фазовыми координатами***, а вектор $\boldsymbol{x}(t)$ будем называть ***фазовым вектором объекта***, под которым будем подразумевать всякий вектор, компоненты которого полностью характеризуют текущее состояние объекта и при выбранном допустимом управлении и заданных начальных условиях полностью определяют движение объекта в рассматриваемые моменты времени**.**

В такой трактовке переходный процесс колебательного звена второго порядка может быть отображен на фазовой плоскости «функция решения – первая производная решения» и при затухающем переходном процессе годограф фазового вектора имеет вид спиралей, стягивающихся к нача-

лу координат или к эллипсам при гармонических воздействиях или нулевом затухании.

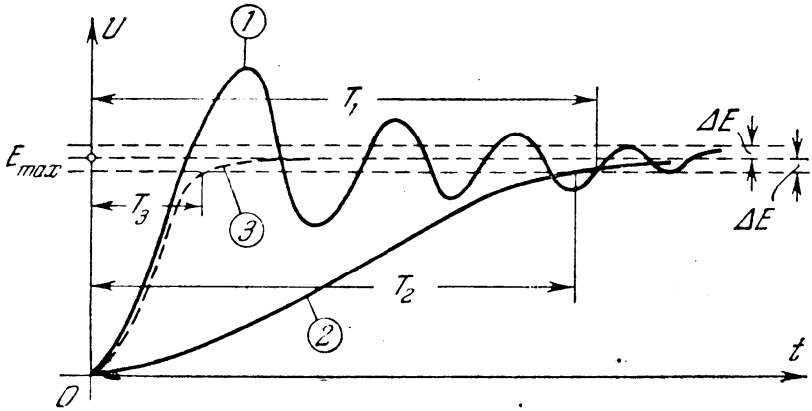
При выполнении компьютерного моделирования в лабораторных работах мы построим простейшие фазовые портреты управляемых объектов.

10. О проблеме оптимального управления

В современных системах устройством управления, которое принимает на своих входах заданные и измеренные значения выходных параметров, значения измеряемых возмущений, содержит зависимости между входами и выходами управляемого объекта, вычисляет и подает на его вход необходимые изменения управляющих воздействий для наиболее точного выполнения задания, являются вычислительные машины. К управляющим устройствам естественно предъявить не только требование свести к нулевому уровню рассогласование между заданными и текущими значениями выходных параметров объекта, но и сделать это как можно быстрее – такие системы называют ***оптимальными по быстродействию***.

Задача создания оптимального по быстродействию алгоритма управления очень сложна даже в простейших случаях. Как правило, управляемые процессы описываются системой нелинейных дифференциальных уравнений достаточно высокого порядка. Но пусть для простого примера замкнутая система управления в целом (объект и управляющее устройство или регулятор) описываются линейным дифференциальным уравнением второго порядка с постоянными коэффициентами. При малых значениях коэффициента демпфирования (при первой производной в дифференциальном уравнении) переходный процесс будет иметь резко колебательный характер (с большим перерегулированием) и время компенсации (например, ступенчатого возмущения по нагрузке) будет большим. Если сделать коэффициент демпфирования слишком большим, то движение к заданному значению будет апериодическим и время регулирования будет также большим. Выбором коэффициента демпфирования можно уменьшить время регулирования до некоторого значения, но простейшие соображения подсказывают, что можно достичь лучшего результата, если пе-

рейти в нелинейный режим – при больших рассогласованиях сделаем коэффициент малым, а при достаточном уменьшении рассогласования резко его увеличим: начальная фаза процесса обеспечит движение к нулевому рассогласованию с максимальной скоростью, а завершающая будет соответствовать плавному апериодическому подходу к заданию. В результате время регулирования значительно сократится.



Переходный процесс: 1 – при малом демпфировании, 2 – при сильном демпфировании, 3 – при переключении значения коэффициента демпфирования.

Системы, способные изменять свои параметры для достижения лучшего качества переходных процессов носят название ***систем с переменной структурой***. Исследование нелинейных систем очень трудоемко, но ожидаемые результаты того стоят.

В большинстве реальных случаев объект управления задан и его характеристики изменять нельзя – в этом случае задача разработки системы управления сводится к созданию такого реализуемого управляющего алгоритма, который обеспечивает наилучшее в определенном смысле управление объектом. Структура управляющего алгоритма зависит от характеристик объекта и предъявляемых к про-

цессу требований, объема доступной информации о процессе, ограничений на ресурсы управления (на управляющие воздействия) и на все переменные состояния объекта – координаты, скорости, ускорения и пр. Математическая модель управляемого объекта в общем случае может быть представлена системой дифференциальных уравнений вида:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{z}, t).$$

Здесь $\boldsymbol{x}$, $\boldsymbol{u}$, $\boldsymbol{z}$ – вектор-функции от времени t выходных параметров, управляющих воздействий, измеряемых возмущений соответственно, $\boldsymbol{f}$ – вектор-функция, связывающая их взаимной зависимостью. В зависимости от того, какая группа из этих функций должна быть найдена в результате решения системы уравнений, различают следующие задачи:

- ✓ задачу анализа (система решается относительно $\boldsymbol{x}$);
- ✓ задачу идентификации объекта управления (определяется $\boldsymbol{f}$);
- ✓ задачу определения алгоритма управления (система решается относительно $\boldsymbol{u}$ при заданных требования к нему и наложенной системе ограничений).

В этом разделе мы обсудим постановку и некоторые методы решения последней задачи.

Предъявляемые к управляемому объекту требования содержат ***цель управления*** – на языке математики это всегда желание достичь экстремума некоторой величины Q – критерия оптимальности, значение которого в общем случае зависит как от выходной величины, так и от управляющего воздействия и может явно зависеть от времени:

$$Q(\boldsymbol{x}, \boldsymbol{u}, t) = min.$$

Q – это функционал, число, зависящее от вида функций $\boldsymbol{x}$, $\boldsymbol{u}$. Например, в частном случае $Q = \int_0^T [x(t)]^2 dt$, где T –

фиксированная величина.

В качестве критерия Q могут быть выбраны различные технические или экономические показатели – производительность, качество, затраты сырья или энергоресурсов: выбор за прикладной областью, а не за теорией управления.

В настоящее время продолжается построение единой общей теории оптимальных систем управления, включающей как формулировку общих задач, так и методы их решения. Рассмотрим общие задачи этой теории.

Пусть заданы: оператор объекта $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{z}, t)$, цель управления Q, ограничения на управления $\boldsymbol{u}\in\Omega(\boldsymbol{u})$ и/или на координаты объекта $\boldsymbol{x}\in\Omega(\boldsymbol{x})$. Зададимся также классом кусочно-непрерывных функций $\boldsymbol{u}(t)$ с конечным числом точек разрыва первого рода на конечном интервале. Задача состоит в том, чтобы при этих заданных условиях найти такой алгоритм (стратегию) управляющего устройства, при котором критерий оптимальности принимает минимальное значение. Такой алгоритм называют оптимальным. В частном случае систем с полной информацией об объекте, когда функции $\boldsymbol{z}$ и $\boldsymbol{x}$ регулярны и могут быть включены в состав оператора объекта и критерия оптимальности, общее выражение для алгоритма оптимальной системы имеет вид:

$$\boldsymbol{u}(t)=\boldsymbol{K}[\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), t] \ (t_0\le\tau\le t).$$

Если текущее значение вектора $\boldsymbol{x}$ полностью характеризует все будущее поведение объекта независимо от предыстории при $\tau\le t$, объект характеризуется заданными дифференциальными уравнениями и не содержит запаздываний, то $\boldsymbol{u}(t)$ в данный момент является функцией только значения $\boldsymbol{x}$ в тот же момент времени $\boldsymbol{u}(t)=\boldsymbol{K}[\boldsymbol{x}(t), t]$. Если, наконец, объект стационарен (уравнения движения не содержат явно время t), то оптимальный алгоритм ищется в виде функции $\boldsymbol{u}(t)=\boldsymbol{K}[\boldsymbol{x}(t)]$.

Изложенная постановка задачи оптимального управления является основной. Но существует и другая задача – об определении оптимальных процессов $\boldsymbol{u}(t)$, $\boldsymbol{x}(t)$ при задан-

ных начальных условиях $\boldsymbol{x}_0$; это не главная задача, она выдвигается обычно в виде промежуточной, чтобы от нее перейти к основной – определению оптимального алгоритма. Исключая, например, из зависимости $\boldsymbol{u}(t)$, $\boldsymbol{x}(t)$, при некоторых дополнительных условиях можно найти зависимость $\boldsymbol{u}(\boldsymbol{x})$, то есть алгоритм оптимального управляющего устройства.

Постановку задачи определения оптимального процесса, например, для систем с полной информацией об объекте можно сформулировать так. Пусть задано векторное уравнение движения объекта

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t),$$

где $\boldsymbol{f}_i$ – непрерывные и дифференцируемые по своим аргументам функции, $\boldsymbol{x}(t_0)=\boldsymbol{x}_0$.

Допустимыми будем считать управления, удовлетворяющие имеющимся ограничениям $\boldsymbol{u}\in\Omega(\boldsymbol{u})$. В задаче об оптимальном процессе требуется найти такое допустимое $\boldsymbol{u}(t)$ и соответствующее ему движение $\boldsymbol{x}(t)$ объекта, чтобы траектория изображающей точки в фазовом пространстве системы, переходящей из начального состояния $\boldsymbol{x}_0$ в состояние $\boldsymbol{x}_T$ обеспечивала минимум некоторому функционалу $Q=\int_0^T G[\boldsymbol{x}(t),\boldsymbol{u}(t),t]dt$, где G – конечная, обычно положительная скалярная функция $\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$ и t. Явную зависимость от времени формально можно устранить вводом дополнительной координаты $\boldsymbol{x}_{n+1}$, $\boldsymbol{x}_{n+1}(t=0)=0$, $\frac{d\boldsymbol{x}_{n+1}}{dt}=1$. Так как при этом $\boldsymbol{x}_{n+1}=t$, то вместо t можно везде писать $\boldsymbol{x}_{n+1}$ и новая система не будет содержать явно время, но размерность вектора ее выходных координат увеличится на 1. В частном случае задачи с фиксированной конечной точкой $\boldsymbol{x}_T$, но не фиксированным временем ее достижения T, положив в выражении для Q $G=1$, получим $Q=T$ и требование $Q=min$

превращается в $T=min$, то есть получаем ***задачу о максимальном быстродействии***, сыгравшую большую роль в формировании общей теории оптимальных систем.

11. Динамическое программирование как математический метод решения задач оптимального управления

В 1950-х годах Р. Беллман с сотрудниками разработали новый общий метод решения вариационных задач (задач на минимизацию функционалов) и назвали его ***динамическим программированием***. С тех пор этот метод широко используется для решения многих классов задач оптимального управления динамическими системами. Основные положения метода таковы. Пусть решению подлежит задача управления объектом, описываемым системой уравнений

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}), \quad (*)$$

где $\boldsymbol{x}$ – n-мерный вектор с координатами $x_1, x_2, \ldots, x_n$, $\boldsymbol{u}$ – r-мерный вектор с координатами $u_1, u_2, \ldots, u_r$, $\boldsymbol{u} \in \Omega(\boldsymbol{u})$ и требуется найти $\boldsymbol{u}$, обеспечивающий минимум интеграла

$$Q = \int_0^T G[\boldsymbol{x}(t), \boldsymbol{u}(t)]dt, \quad (**)$$

где T пока будем считать фиксированным. Случай с явной зависимостью G и $\boldsymbol{f}$ от времени можно свести к выражениям вида (*) и (**).

В основе метода динамического программирования лежит принцип оптимальности, сформулированный Беллманом для широкого круга систем, будущее поведение которых определяется их состоянием в настоящем и не зависит от характера их «предыстории».

Пусть в n-мерном фазовом пространстве есть оптимальная траектория с начальными $\boldsymbol{x}_0$ и конечными $\boldsymbol{x}_T$ значениями вектора $\boldsymbol{x}$ при $t=t_0$ и $t=T>t_0$. Возьмем на траектории какую-либо точку x_1, соответствующую $t=t_1$, $t_0<t_1<T$, разбивающую траекторию на 2 участка. При этом второму участку соответствует часть интеграла (**), соответствующая нижнему пределу t_1.

Принцип оптимальности можно сформулировать так:

Второй участок оптимальной траектории тоже является оптимальной траекторией.

Это означает, что независимо от того, как система пришла к состоянию x_1, ее оптимальное движению в последующем будет соответствовать второй участок. Этот общий принцип оптимальности – необходимое условие оптимального процесса – справедлив как для непрерывных, так и для дискретных систем. Несмотря на кажущуюся тривиальность принципа оптимальности, из него, рассуждая методически, можно вывести нетривиальные условия для оптимальной траектории.

Другая формулировка принципа оптимальности может выглядеть так:

Оптимальная стратегия не зависит от предыстории системы и определяется лишь ее состоянием в рассматриваемый момент времени.

Подход Беллмана к решению задачи управления рассмотрим на простейшем примере управляемого объекта, движение которого описывается уравнением 1-го порядка $\frac{dx}{dt}=f(x, u)$, начальное условие $x(t_0=0)=x_0$ и требуется найти $u(t)$, минимизирующее функционал

$$Q=\int_{t_0}^{T} G_1(x,u)dt+\varphi_1[x(T)],$$

где T для простоты фиксировано. Для упрощения рассуждений и в порядке неизбежного этапа подготовки задачи к решению на ЭВМ дискретизируем задачу: разобьем интервал $(0, T)$ на N равных участков малой длины h и будем рассматривать дискретные значения $x=x(k)$ и $u=u(k)$ ($k=0, 1, \ldots, N$) в моменты времени $t=0, h, 2h, \ldots, (N-1)h, Nh=T$. Тогда дифференциальное уравнение объекта можно приближенно заменить уравнением в конечных разностях

$$\frac{x(k+1)-x(k)}{h}=f_1[x(k),\,u(k)],$$

или

$$x(k+1)=x(k)+f[x(k),\,u(k)],$$

где

$$f[x(k),\,u(k)]=hf_1[x(k),\,u(k)].$$

Интеграл в критерии оптимальности приближенно заменяется суммой

$$Q=\sum_{n=0}^{N-1}G\big[x(k),u(k)\big]+\varphi[x(N)],$$

где

$$G[x(k),\,u(k)]=G_1[x(k),\,u(k)]h,$$
$$\varphi[x(N)]=\varphi_1[x(Nh)]=\varphi_1[x(T)].$$

Задача теперь состоит в определении дискретных значений u, т.е. величин $u(0)$, $u(1)$, …, $u(N-1)$, минимизирующих вышеприведенную сумму – при заданных конечно-разностном уравнении объекта, ограничениях на управления и начальных условиях требуется найти минимум сложной функции многих переменных. Метод динамического программирования дает возможность свести эту задачу к *последовательности минимизаций функций одной переменной*.

Для решения задачи применяется попятное движение от конца процесса, т.е. от момента $t=T$ к его началу. Рассмотрим момент $t=(N-1)h$, предполагая, что все значения $u(i)$ ($i=0, 1, 2, \ldots, N-2$), кроме последнего $u(N-1)$, уже както были получены и есть некоторое значение $x(N-1)$, соответствующее моменту $t=(N-1)h$. В соответствии с принципом оптимальности, воздействие $u(N-1)$ не зависит от предыстории процесса и определяется только состоянием $x(N-1)$ и целью управления. Искомое $u(N-1)$ влияет только на те члены суммы в Q, которые относятся к последнему участку. Пусть сумма этих членов

$$Q_{N-1}=G[x(N-1),\,u(N-1)]+\varphi[x(N)].$$

Из конечно-разностного уравнения системы получим $x(N)=x(N–1)+f[x(N–1),\ u(N–1)]$, которое тоже зависит от $u(N–1)$.

Найдем допустимое значение $u(N–1)$, минимизирующее $Q_{N\text{-}1}$.

Пусть

$$S_{N\text{-}1}[x(N–1)]=\min_{u(N-1)\in\Omega(u)} Q_{N\text{-}1}=$$

$$=\min_{u(N-1)\in\Omega(u)} \{G[x(N–1),\ u(N–1)]+\varphi[x(N)]\}=$$

$$=\min_{u(N-1)\in\Omega(u)} \{G[x(N–1),\ u(N–1)]+\varphi[x(N–1)]+f[x(N–1),\ u(N–1)]\}.$$

Для определения $S_{N\text{-}1}$ нужно производить минимизацию только по одному $u(N–1)$. После выполнения этой процедуры получим $S_{N\text{-}1}$ в виде функции от $x(N–1)$ – последовательность ее значений придется сохранить.

Теперь перейдем к предпоследнему участку времени и будем рассматривать два участка – последний и предпоследний вместе; заметим при этом, что выбор $u(N–2)$ и $u(N–1)$ повлияет только на те слагаемые суммы в Q, которые входят в состав выражения

$$Q_{N\text{-}2}=G[x(N–2),\ u(N–2)]+\{G[x(N–1),\ u(N–1)]+\varphi[x(N)]\}.$$

Величину $x(N–2)$ в начальный момент предпоследнего интервала, полученную из «предыстории», будем считать заданной. Из принципа оптимальности снова следует, что оптимальное управление на рассматриваемом участке времени определяется только значением $x(N–2)$ и целью управления. Найдем $S_{N\text{-}2}$ как минимум $Q_{N\text{-}2}$ по $u(N–2)$ и $u(N–1)$. Но минимум по $u(N–1)$ слагаемого в фигурной скобке уже был найден раньше *для каждого значения* $x(N–1)$, которое само зависит от $u(N–2)$. Кроме того, при минимизации $Q_{N\text{-}1}$ попутно было найдено и соответствующее оптимальное значение $u(N–1)$ – обозначим его через $u^*(N–1)$. Если учесть также, что первое слагаемое $Q_{N\text{-}2}$ не зависит от $u(N–2)$, то мы можем записать:

$$S_{N\text{-}2}[x(N–2)]=\min_{u(N-2)\in\Omega(u)} Q_{N\text{-}2}=$$

$$= \min_{u(N-2)\in\Omega(u)} \{G[x(N-2), u(N-2)] + S_{N-1}[x(N-1)]\} =$$

$$= \min_{u(N-2)\in\Omega(u)} \{G[x(N-2), u(N-2)] +$$

$$+ S_{N-1}[x(N-2)] + f[x(N-2), u(N-2)]\}.$$

Здесь минимизация также проводится только по одному переменному $u(N-2)$; при этом находим $u^*(N-2)$ как оптимальное значение $u(N-2)$ и величину S_{N-2} как минимум функции Q_{N-2}. Обе эти функции – от аргумента $x(N-2)$. Массив значений S_{N-2} сохраняется, а массив $S_{N-1}[x(N-1)]$ можно освободить. Найденное оптимальное значение $u^*(N-2)$ минимизирует все выражение в фигурной скобке формулы S_{N-2}, а не только слагаемое $G[x(N-2), u(N-2)]$: оптимальная стратегия учитывает конечную цель, т.е. минимизацию всего выражения в фигурных скобках, зависящего от $u(N-j)$.

Продолжая попятное движение к началу промежутка $(0, T)$, на третьем от конца участке будем рассматривать ту часть суммы Q, которая зависит от $u(N-3)$:

$$Q_{N-3} = G[x(N-3), u(N-3)] +$$

$$+ \{G[x(N-2), u(N-2)] + G[x(N-1), u(N-1)] + \varphi[x(N)]\}.$$

Используя рекуррентное конечно-разностное уравнение, получим:

$$x(N-2) = x(N-3) + f[x(N-3), u(N-3)]$$

и минимум в фигурной скобке для Q_{N-3} равен $S_{N-2}[x(N-2)]$. Поэтому минимум S_{N-3} в выражении Q_{N-3} равен:

$$S_{N-3}[x(N-3)] = \min_{u(N-3)\in\Omega(u)} \{G[x(N-3), u(N-3)] + S_{N-2}[x(N-2)]\} =$$

$$= \min_{u(N-3)\in\Omega(u)} \{G[x(N-3), u(N-3)] +$$

$$+ S_{N-2}[x(N-3)] + f[x(N-3), u(N-3)]\}.$$

У нас уже достаточно материала для индуктивного получения рекуррентной формулы для определения S_{N-k}:

$$S_{N-k}[x(N-k)] = \min_{u(N-k)\in\Omega(u)} \{G[x(N-k), u(N-k)] +$$

$$+ S_{N-k+1}[x(N-k)] + f[x(N-k), u(N-k)]\}.$$

Параллельно в процессе минимизации правой части

этой формулы определяется оптимальное значение $u^*(N–k)=u^*[x(N–k)]$, минимизирующее выражение в фигурной скобке для $S_{N-k}[x(N–k)]$. В конце концов мы придем к $u^*(0)$: к управлению в начальный момент времени – то есть то, что было необходимо, так как текущий момент времени можно считать начальным, а все последующие – будущими. Одновременно получим и S_0 – минимум критерия Q при оптимальном управлении.

В большинстве случаев аналитическое выполнение описанной процедуры минимизации невозможно и ее следует рассматривать как алгоритм вычислений на ЭВМ.

Процесс решения для одного уравнения можно перенести на объект любого порядка n с любым числом управляющих воздействий u_l (l=1, …, r) – для этого достаточно заменить скаляры x, u, f векторами $\boldsymbol{x}$, $\boldsymbol{u}$, $\boldsymbol{f}$. Эти векторы придется ввести для k-го момента времени $t=kh$:

$$\boldsymbol{x}(k)=\{x_1(k), \ldots, x_n(k)\}$$
$$\boldsymbol{u}(k)=\{u_1(k), \ldots, u_r(k)\}$$

где $u_j(N–k)$ – j-е управление, $x_j(N–k)$ – j-я координата в момент $t=(N–k)h$.

Заменим векторное дифференциальное уравнение векторным уравнением в конечных разностях

$$\boldsymbol{x}(k+1)=\boldsymbol{x}(k)+\boldsymbol{f}[\boldsymbol{x}(k), \boldsymbol{u}(k)],$$

а интеграл критерия оптимальности суммой и аналогичные рассуждения приведут нас к следующему решению:

$$S_{N-k}[\boldsymbol{x}(N–k)]=\min_{u(N-k)\in\Omega(u)}\{G[\boldsymbol{x}(N–k), \boldsymbol{u}(N–k)]+$$
$$+S_{N-k+1}[\boldsymbol{x}(N–k)]+\boldsymbol{f}[\boldsymbol{x}(N–k), \boldsymbol{u}(N–k)]\}.$$

Теперь на каждом этапе придется искать минимум функции r переменных $u_1(N–k)$, …, $u_r(N–k)$, а оптимальные скаляр S_{N-k} и вектор $\boldsymbol{u}^*(N–k)$ стали функциями вектора $\boldsymbol{x}(N–k)$, т.е. функциями n переменных $x_1(N–k)$, …, $x_n(N–k)$.

Таким образом, вместо привычных в анализе формул мы получили процедуру получения решения, выражаемого в форме таблиц; по ним, при желании, можно построить графики.

В большинстве практических случаев решение по методу динамического программирования достаточно громоздко – на каждом этапе вычислений требуется запоминать в табличной форме две функции n переменных $S_{N-k}(\boldsymbol{x})$, $S_{N-k+1}(\boldsymbol{x})$, что при больших n требует большого расхода памяти и может понадобиться использование аппроксимаций.

Отметим, что приведенная методика без принципиальных изменений переносится и на оптимальные системы со случайными процессами, но мы этот вариант рассматривать не будем. При ряде допущений метод динамического программирования может быть применен и для непрерывных систем, но все же основная область его практического применения – дискретные или приведенные к ним системы.

12. Лабораторный практикум по компьютерному моделированию линейных стационарных динамических систем операторным методом

12.1. Введение

Использование операторного метода предполагает интенсивную работу с комплексными полиномами, методами определения комплексных корней полиномиальных уравнений, с такими математическими объектами, как матрицы и векторы. Поэтому необходима предварительная разработка программной поддержки операций и процедур полиномиальной, векторной и матричной алгебры – другими словами, необходимо создание математических классов векторов, полиномов, матриц, с которыми программирующий пользователь мог бы обращаться так же просто, как со стандартными типами целых, вещественных чисел и др., не выписывая множества циклических, условных и других операторов.

После выполнения такой предварительной работы решение линейных дифуравнений с постоянными коэффициентами становится значительно более простым делом. Современные версии языка C++ с мощной поддержкой объектно-ориентированного стиля программирования, возможностью переопределения (перегрузки) функций и стандартных операций для пользовательских классов позволяют выполнить такую разработку достаточно эффективно – в учебном процессе это позволило бы приобрести и закрепить навыки объектно-ориентированного программирования попутно с решением прикладных задач по моделированию. Возможно выполнение такой работы и в современных версиях языка Паскаль, все еще занимающего место в системе образования в силу ее большой инерционности (хотя и с существенно меньшим комфортом в разработке и по-

следующем использовании). Для вузов, использующих Паскаль, это тем более важно, что в возможности его математических библиотек крайне ограничены (начиная с отсутствия поддержки комплексной арифметики).

Поэтому мы предлагаем 3 уровня сложности лабораторных работ, которые могут быть использованы в зависимости от имеющегося по учебному плану времени и уровня предварительной подготовки слушателей в области программирования и вычислительной математики.

Уровень повышенной сложности предполагает предварительную разработку упомянутых математических классов, разработку программ решения задаваемых в интерактивном режиме дифференциальных уравнений, расчет частотных характеристик и анализ результатов в процессе многочисленных имитаций.

Средний уровень сложности предполагает предоставление студентам готовых исходных кодов векторных, матричных, полиномиальных классов со всеми необходимыми встроенными методами и им предлагается создать на их базе программный продукт для решения линейных дифуравнений с постоянными коэффициентами операторным методом и затем выполнить работы по имитации систем различного уровня сложности. Исходные тексты реализации указанных классов на языках C++ и Паскаль приведены в приложении.

Нижний уровень сложности (ознакомительный) предполагает только имитационное моделирование линейных разомкнутых и замкнутых систем на готовых программах с построением графиков переходных процессов, частотных характеристик и годографов на фазовой плоскости, анализом влияния параметров системы на указанные характеристики. Исходные тексты класса–решателя дифуравнений также приведены в приложении.

Ниже приведены задания ознакомительного уровня, а задания более высоких уровней сложности (требования к

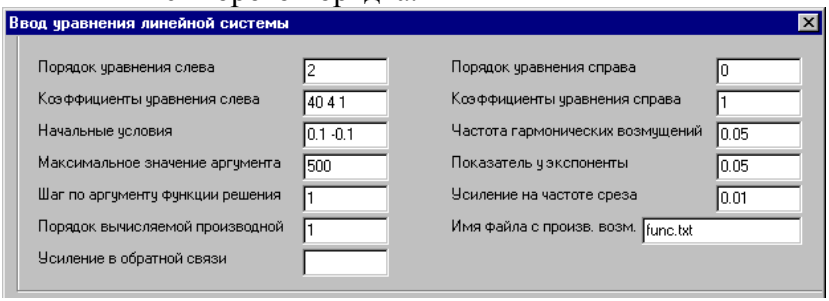
функциональным возможностям программных продуктов) ясны из анализа приведенных в приложении исходных текстов программ.

12.2. Лабораторная работа №1

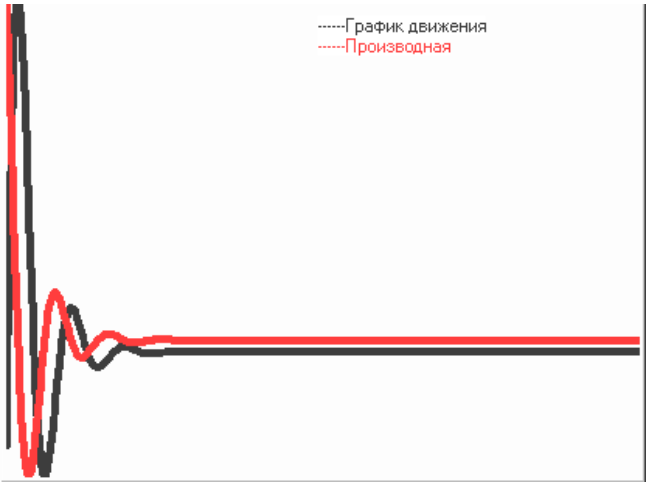
Тема: Исследование свободного движения разомкнутых линейных динамических систем при ненулевых начальных условиях.

Задание:

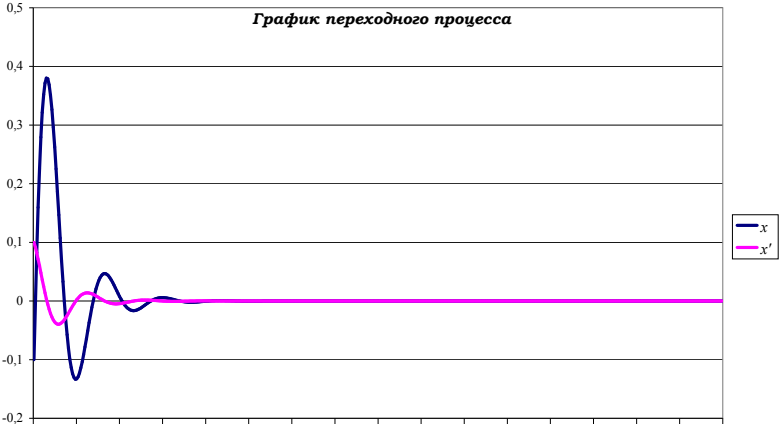
1) Вызвать на выполнение исполняемый файл первой работы. В открывшемся меню выбрать пункт «Ввод данных о системе» и в появившемся окне диалога заполнить предлагаемые строки ввода сведениями о порядке уравнения, его коэффициентах, начальных условиях и пр. Для начала вводите уравнение не выше второго порядка.



2) Решить вручную заданное дифуравнение операторным методом.
3) По корням характеристического уравнения определить, устойчива ли система, и обосновать сделанный вывод.
4) С помощью программы построить график переходного процесса и сравнить его с результатами ручных вычислений.



5) Предоставляемая программа позволяет получить только качественную картину переходных процессов. Для получения количественных оценок в тарированных координатных осях используйте выводимый программой файл решения и мастер построения диаграмм Microsoft Excel – для этого вам придется перенести файл в таблицу, возможно, предварительно преобразовав его при наличии нестандартных разделителей.



	1	32	64	95	126	157	189	220	251	282	314	345	376	407	439	470
x	-0,1	-0,133	0,004	0,006	-3E-04	-2E-04	1,21E-	1,08E-	-	-	3,32E-	2,01E-	-	-	8,24E-	3,70E-
x'	0,1	0,002	-0,004	-1E-04	2E-04	1,33E-	-	-	3,78E-	-	-	2,44E-	7,23E-	-	-	1,13E-

6) Определить и объяснить влияние на характер переходного процесса значений начальных условий и коэффициентов дифференциального уравнения.
7) Составьте письменный отчет о выполненной работе.

12.3. Лабораторная работа №2

Тема: Исследование свободного движения линейных динамических систем с жесткой обратной связью.

Задание:

1) Вызвать на выполнение исполняемый файл второй работы. В открывшемся меню выбрать пункт «Ввод данных о системе» и в появившемся окне диалога заполнить предлагаемые строки ввода сведениями о порядке уравнения, его коэффициентах, начальных условиях и коэффициенте усиления в контуре жесткой обратной связи. Для начала вводите уравнение второго порядка.
2) С помощью программы построить графики переходных процессов для различных входных воздействий при различных значениях усиления в контуре обратной связи.
3) Определить и объяснить влияние на характер переходного процесса, скорость его затухания, значения усиления в обратной связи.
4) Составьте письменный отчет о выполненной работе.

12.4. Лабораторная работа №3

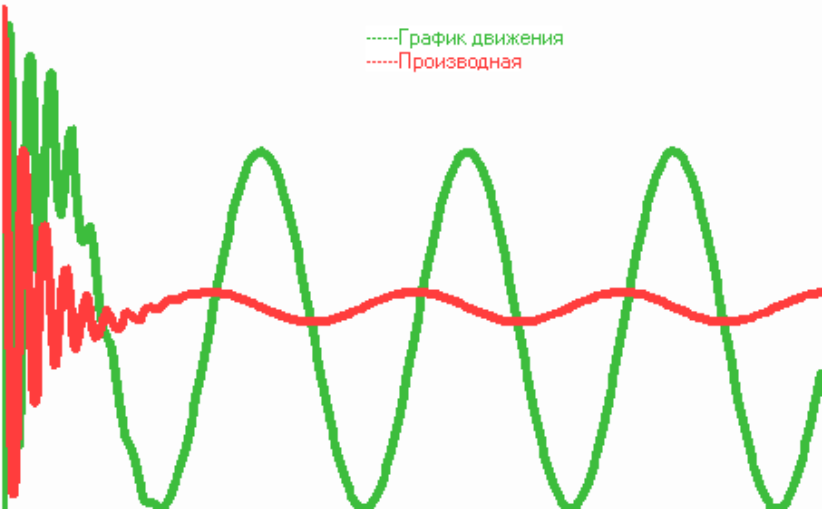
Тема: Исследование вынужденного движения линейных динамических систем при заданных начальных условиях при **стандартных** возмущениях.

Задание:

1) Вызвать на выполнение исполняемый файл третьей работы. В открывшемся меню выбрать пункт «Ввод данных о системе» и в появившемся окне диалога заполнить предлагаемые строки ввода сведениями о

порядке уравнения, его коэффициентах, начальных условиях и пр. Для начала вводите уравнение второго порядка.

2) Решить вручную заданное дифуравнение операторным методом при импульсном, ступенчатом, синусоидальном и экспоненциальном возмущениях.
3) С помощью программы построить графики переходных процессов для различных входных воздействий и сравнить их с результатами ручных вычислений. Построить те же графики в Microsoft Excel.



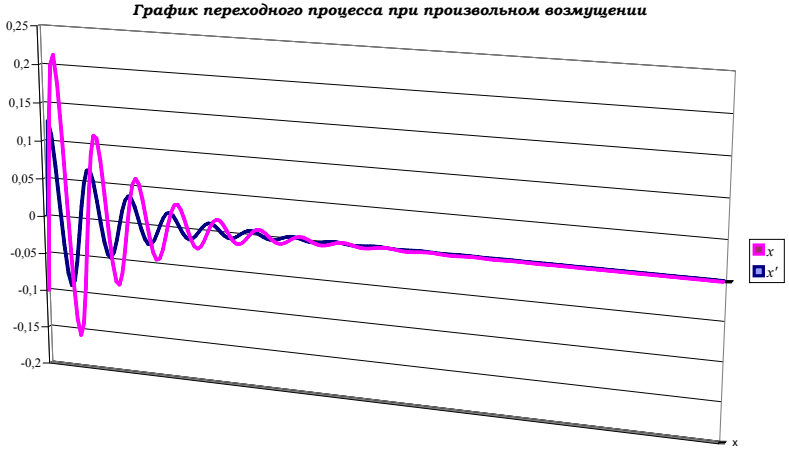
4) Определить и объяснить влияние на характер переходного процесса значений начальных условий, коэффициентов дифференциального уравнения и возмущающих воздействий.
5) Составьте письменный отчет о выполненной работе.

12.5. Лабораторная работа №4

Тема: Исследование вынужденного движения линейных динамических систем при заданных начальных условиях и **произвольных** возмущениях.

Задание:

1) Составьте и отладьте программу, генерирующую выбранную вами произвольную возмущающую функцию и записывающую ее с заданным шагом в дисковый файл.
2) Объясните использование интеграла свертки и его дискретного аналога для вычисления реакции системы на произвольное возмущение и выполните расчет этой реакции, указав имя созданного вашей программой файла в окне ввода данных.
3) Постройте графики реакции на произвольное возмущение в предоставленной программе для качественного анализа и в Microsoft Excel для количественной оценки его характеристик.



4) Составьте отчет.

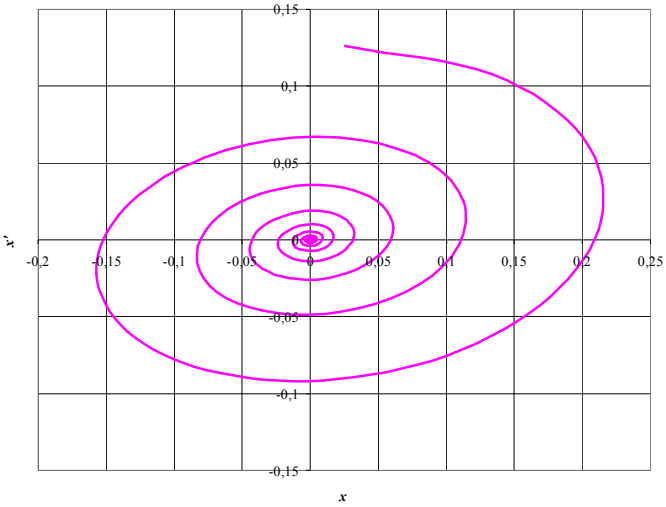
12.6. Лабораторная работа №5

Тема: Качественный анализ линейных систем с помощью годографа фазового вектора (фазового портрета).

Задание:

1) Дайте определения фазового пространства, фазового вектора, фазового портрета системы.

2) Задайте уравнение объекта и вид входного воздействия и постройте годограф фазового вектора. Объясните его связь с уравнением объекта, входным возмущением, начальными условиями.



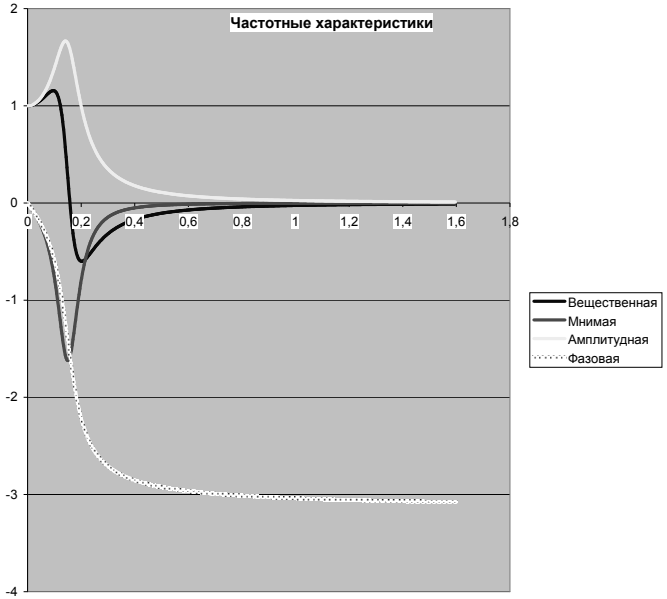
3) Повторите п. 2 при различных параметрах уравнения и различных возмущениях при нулевых и ненулевых начальных условиях.
4) Предложите и опишите метод вычисления производной реакции объекта на произвольное возмущение и построения соответствующего фазового портрета.
5) Составьте отчет.

12.7. Лабораторная работа №6

Тема: Частотный анализ линейных систем.
Задание:

1) Перечислите виды частотных характеристик и изложите методы их расчета.
2) Задайте уравнение объекта, вычислите его собственную частоту и постройте с помощью предоставленной программы частотные характеристики,

варьируя значения коэффициентов уравнения.



3) Объясните влияние коэффициентов уравнения на характер всех частотных характеристик. Опишите, как по частотным характеристикам можно судить о динамических свойствах объекта.
4) Составьте отчет.

12.8. Лабораторная работа №7

Тема: Комплексный анализ линейных систем.
Задание:

1) Повторите все предыдущие задания, определите и опишите взаимосвязь различных характеристик динамических объектов.
2) Предложите и опишите в отчете метод расчета переходных процессов в линейной динамической системе при переменных коэффициентах уравнения (зависящих от времени).
3) Составьте отчет.

13. Программная реализация операторного метода анализа динамики линейных систем

Програмная реализация на C++ приведена для обобщенного варианта лабораторной работы №7; все остальные – её частные случаи. Программная реализация на Object Pascal была выполнена Александром Владимировичем Епишиным и Дмитрием Станиславовичем Федоренко (приводится с любезного согласия авторов).

13.1. Исходные тексты программы на языке C++, выполненные в среде C++ Builder 3

13.1.1. Класс линейных дифуравнений с постоянными коэффициентами

```
//DIFEQU7.H

#include <windows.h>
#include <alloc.h>
#include "equation.h"

typedef vector<dcomplex> cdvector;
/*Шаблон структуры для сведений о корнях характери-
стического полинома */

  struct CA{
  dcomplex ROOT;         //значение корня
  dcomplex A; //коэффициент для корня в оригинале
  int  J;                //кратность
  int  O;};              //кому кратен
dmatrix SolMtr; //Матрица решения и его производных
dvector tc;               //Вектор аргумента решений
dvector userfunc; /*Вектор для произвольного возму-
щения */
dmatrix FreqChMtr; /*Матрица для частотных характе-
ристик*/
double w;               //Частота среза
double Cufz;            //Усиление на частоте среза
```

```
/*Класс обыкновенных линейных дифференциальных
уравнений (ОЛДУ), содержащий методы работы, базиру-
ющиеся на операционном исчислении. */

class DifferentialEquation
{ //приватные данные
 long rngl,rngr;          //Порядок уравнения
 dvector nv; /*Вектор коэффициентов и начальных
условий */
 double tend, tstep;   /*Конечное время решения и
шаг по времени */
 int PointSolCnt;

 //Полиномы левой, правой части и начальных условий
 cpolynom PL,PR,PN;
/*Знаменатель и числитель изображения решения в ви-
де полиномов */
 cpolynom PSOL,QSOL;

  CA * R;           //указатель на структуру
  long RootCount;  /*Количество корней общее - при
наличии правой части уравнения оно не совпадает с
его порядком слева */
  long rcount;   //Количество различных корней
  int cod; double omega, alpha;/*Сведения о стан-
дартных возмущениях */
  double kos;
 public:        //общедоступные члены класса
DifferentialEquation(dvector CFNL, dvector CFNR,
dvector NU, int COD, double OMEGA, double ALPHA,
double TEND, double TSTEP, double KOS);

  //Деструктор
  ~DifferentialEquation() {if(R!=NULL) farfree(R);}


/*Функция для вычисления корней характеристического
полинома */
  void GetRoot();

/*Функция формирования числителя изображения произ-
водных функции решения */
```

```
cpolynom Qsol(int der);/*аргумент - порядок произ-
водной */

/*Функция, вычисляющая коэффициенты оригинала для
всех корней; результат помещается в массив структур
R*/
  void GetCoeffOrigin(void);
/*Функция, возвращающая значение выхода при задан-
ном значении аргумента */
  double GetValue(double t);
  void Sol();
  void FreqChar(void);
};

/*Нам понадобятся простые служебные функции - вы-
числения факториала и определения знака числа */
dcomplex fact(long x)
{ long ret=1; for(long i=1;i<=x;i++) ret*=i; return
dcomplex(double(ret),0);}


inline long sign(double x)
{ return (x>0)?1:((x<0)?-1:0);}

 //Определение методов класса

 //Конструктор по данным пользователя
DifferentialEquation::DifferentialEquation(dvector
CFNL, dvector CFNR, dvector NU, int COD, double
OMEGA, double ALPHA, double TEND, double TSTEP,
double KOS) : cod(COD), omega(OMEGA), alpha(ALPHA),
tend(TEND),tstep(TSTEP), kos(KOS)
{R=NULL;
//Определяем порядки уравнения слева
int i;
rngl=CFNL.getm()-1;rngr=CFNR.getm()-1;
cpolynom plt=cpolynom(rngl+1),
prt=cpolynom(rngr+1),qsolt,psolt;

/*Инициализация полиномов с преобразованием коэффи-
циентов в комплексные и реверсированием их последо-
вательности - мы предполагаем, что вводятся коэффи-
```

```
циенты и начальные условия начиная со старшей про-
изводной */
for(i=0;i<(rngr+1);i++) prt[i]=dcomplex(CFNR[i],0);
prt=prt.reverse();
PR=prt; QSOL=prt;

for(i=0;i<(rngl+1);i++) plt[i]=dcomplex(CFNL[i],0);
plt=plt.reverse();

cpolynom UOS; UOS[0]=dcomplex(kos,0.0);
if(kos!=0) plt+=(UOS*prt);
PL=plt; PSOL=plt;

//Создадим также вектор начальных условий
nv=dvector(rngl);
for(i=0;i<rngl;i++)     nv[i]=NU[rngl-i-1];

/*Формирование полинома начальных условий (числите-
ля изображения решения при свободном движении. Если
будет ненулевая правая часть, то ее изображение
надо прибавить к  результату вычисления PN) */

  //p-полином первой степени p+0
  cpolynom p(2); p[0]=dcomplex(0.0,0.0);
  p[1]=dcomplex(1.0,0.0);
  cpolynom D=PL,tpn;
  for( i=0;i<rngl;i++)
  {
  D=D/p;//делим полином на p, понижая степень на 1
  /*прибавляем произведение значения производной i-
го порядка в начальной точке на полиномиальный ко-
эффициент */
  for(int j=0;j<rngl;j++) D[i]*=nv[i];
  tpn=nv[i]*D;
  PN+=tpn;
   }

 /*Сформируем числитель QSOL и знаменатель PSOL
изображения решения - это полиномиальная дробь */
   cpolynom rone;rone[0]=dcomplex(1.0,0.0);/* ве-
ществ. полиномиальная единица */
    //Если возмущения нет
   if(cod==0 ) {QSOL=PN; PSOL=PL;}
```

```
   /*Если это единичный импульс или произвольная
функция*/
   if(cod==1 || cod==6)  {QSOL=PR+PN;PSOL=PL;}

   //Если на входе ступенька
   if(cod==2)  {QSOL=PN*p+PR; PSOL=PL*p;}

   //Если на входе - синусоида
   if(cod==3)
 {QSOL=PN*((p^2)+((omega*rone)^2))+PR*(omega*rone);
   PSOL=PL*((p^2)+((omega*rone)^2)); }

   //Если  косинусоида
   if(cod==4)
   { QSOL=PN*((p^2)+((omega*rone)^2))+(PR*p);
     PSOL=PL*((p^2)+((omega*rone)^2)); }

   //Если экспонента
   if(cod==5){QSOL=PN*(p+(alpha*rone))+PR;
   PSOL=PL*(p+(alpha*rone));}

   PointSolCnt=floor(tend/tstep); /*Количество дис-
кретных точек */

   /*Если объект конструируется для расчета частот-
ных характеристик *
 //if(cod==7) return; досрочно покидаем конструктор

   /*Определим теперь общее количество корней ха-
рактеристического полинома */
   RootCount=PSOL.getm()-1;
    /*После определения количества корней выделим
память для хранения сведений о них */
   R=(CA*)farcalloc(RootCount,sizeof(CA));
   memset(R,0,RootCount*sizeof(CA));

  //Конструируем хранители решений
  double j;
   tc=dvector(PointSolCnt);    //Вектор аргумента
 for(i=0,j=0.0;i<PointSolCnt;i++,j+=tstep) tc[i]=j;
  /*Матрица решений на RootCount строк - для функ-
ции и ее производных */
```

```
  SolMtr=dmatrix(RootCount+1,PointSolCnt);
  /*Сразу занесем в матрицу решений начальные зна-
чения функции и ее производных */
  if(RootCount>0)
  for(i=0;i<RootCount;i++)  SolMtr[i][0]=nv[i];

  GetRoot(); //Вычисляем корни x-го полинома

 }//Конец конструктора

   /*Функция для вычисления корней характеристиче-
ского полинома*/
  void DifferentialEquation::GetRoot()
   {
int i,j;

    //ищем корни характеристического полинома
    cvector polyroot=newton(PSOL);

 //Ограничим точность вычисления корней вблизи нуля
  for(i=0;i<RootCount;i++)
{
if(fabs(real(polyroot[i]))<1e-7)
polyroot[i]=dcomplex(0,imag(polyroot[i]));
if(fabs(imag(polyroot[i]))<1e-7)
polyroot[i]=dcomplex(real(polyroot[i]),0);
}

 //Заносим в структуру значения корней
 for(i=0;i<RootCount;i++) R[i].ROOT=polyroot[i];
 /*определяем кратность каждого корня и заносим в
структуру признаком -1 пометим корни уже проверен-
ные на кратность. J=1 будет корень первой кратности
корню 0 и т.д. */
 int repeat;
 rcount=RootCount;/*Вначале предполагаем, что все
корни различны */
  for(i=0;i<RootCount;i++)
  { repeat=1;
    if(R[i].J!=-1) //корень еще не встречался
    {
      R[i].J=1; //количество повторений i-го корня
      for(j=i+1;j<RootCount;j++)
```

```
        if((R[j].J!=-1)&&(R[i].ROOT==R[j].ROOT))
        {
          repeat++;
          rcount--;R[j].J=-1;R[j].O=i;
          /*устанавливаем признак того, что этот
корень уже учтён */
        }
      R[i].J=repeat;//кратность корня
    }
   }
   }


/*Подпрограмма определения числителя изображения
производной решения */
cpolynom DifferentialEquation::Qsol(int der)
{
cpolynom Q=QSOL,p(2),sum;
  //доформируем числитель изображения
  p[0]=dcomplex(0,0);p[1]=dcomplex(1.0,0.0);
  if(der>0)
  {
  for(int i=0;i<der;i++)
  {sum+=Q[RootCount-i-1]*p^(der-i-1); }
   Q=Q*(p^der)-PSOL*sum;
  }
  return Q;
}

/*Функция, вычисляющая коэффициенты оригинала для
всех корней  результат помещается в массив структур
R*/

void DifferentialEquation::GetCoeffOrigin()
{
dcomplex ap,tp;*/Для значений числителя и знамена-
теля при подстановке значения корня */
/*Числитель может оказаться полиномом или просто
числом */
int qi=QSOL.getm();//Выясняем это
int pi=PSOL.getm();
//Если это число
```

```
if(qi==1) ap=QSOL[0];/*Его и используем в качестве
значения числителя*/
if(pi==1) tp=PSOL[0];
if(RootCount==1)//Единственный некратный корень
{
/*Вычисляем значение производной знаменателя изоб-
ражения решения при подстановке в нее значения
единственного корня */
if(pi>1) tp=derive(PSOL,1)(R[0].ROOT);
if(qi>1) ap=QSOL(R[0].ROOT);//Если числитель - по-
лином, подставляем в него значение корня */
R[0].A=ap/tp;
return;
}

//Если корней больше одного
long i,j,k,l;
cpolynom CH=QSOL,ZN=PSOL,RCH,RZN;
cpolynom pw=2;
cpolynom A;
//dcomplex ch;

  for(k=0;k<RootCount;k++)
  {
if(R[k].J==1)            //Для некратного корня
{ //Формируем полином  pw вида p+0
 cpolynom pw(2);pw[0]=-
R[k].ROOT;pw[1]=dcomplex(1.0,0.0);
 if(qi>1) ap=QSOL(R[k].ROOT);
 if(pi>1) tp=(PSOL/pw)(R[k].ROOT);
 R[k].A=ap/tp;
}

//Если корень кратный
if(R[k].J>1)
{
cpolynom pw(2);pw[0]=-R[k].ROOT;
pw[1]=dcomplex(1.0,0.0);
cpolynom pw1=(pw^R[k].J);
//До  кратности этого корня
 for(j=1;j<=R[k].J;)
 {
     if(j==1)
```

```
      {
      if(qi>1) ap=QSOL(R[k].ROOT);
      if(pi>1) tp=(PSOL/pw1)(R[k].ROOT);
      R[k].A=ap/tp;j++;}
      else
      {
for(i=k+1;i<RootCount;i++)
{
//Если нашли кратный текущему
if((R[i].O==k)/*&&(R[i].ROOT==dcomplex(0,0))&&(R[i]
.J==-1)*/)
{
cpolynom tmp=PSOL/pw1;//Восстанавливаем знаменатель
cpolynom
chisl=QSOL,znamen=PSOL/pw1,dchisl,dznamen,m1,m2;
//Дифференцируем дробь изображения решения j-1 раз
for(l=1;l<=(j-1);l++)
{dchisl=derive(chisl,1);dznamen=derive(znamen,1);
 m1=dchisl*znamen;
 m2=dznamen*chisl;
 chisl=m1-m2;
 znamen^=2;
}

R[i].A=chisl(R[k].ROOT)/(znamen(R[k].ROOT)*fact(j-
1));
 j++;
}}}}}}}


/*Подпрограмма, возвращающая значение функции реше-
ния ОЛДУ или ее производной заданного порядка при
заданном t */
double DifferentialEquation::GetValue(double dt)
{ int k,i,index;
double d;
dcomplex result(0,0),t=dcomplex(dt,0);
if(RootCount>0)
{
  for(k=0;k<rcount;k++)
  {
 index=0;
 if(R[k].J==1)//Если корень простой
```

```
 result+=R[k].A*exp(R[k].ROOT*t);
  if(R[k].J>1)//Если корень кратный
  {
  result+=R[k].A*exp(R[k].ROOT*t);index++;
for(i=k+1;i<rcount;i++)
{
if((R[i].O==k)&&(R[i].J==-1))
{
 double dpw=R[k].J-index;

result+=R[k].A*pow(t,dpw)*exp(R[k].ROOT*t)/fact(R[k
].J-index);
index++;}
  }}}
 d=real(result);
return d;
}
return 0;
  }

void DifferentialEquation::Sol()
{int i,k;
  cpolynom Q=QSOL;
  if(cod==6)
  {
  GetCoeffOrigin();
  /*Вычисляем реакцию  на импульс и пишем в 0-ю
строку SolMtr */
  for(k=1;k<PointSolCnt;k++)
SolMtr[0][k]=GetValue(tc[k]);
  /*Реакцию на произвольное возмущение разместим в
1-й строке SolMtr */
  for(i=1;i<PointSolCnt;i++)
  {SolMtr[1][i]=0;
   for(k=0;k<i;k++)SolMtr[1][i]+=SolMtr[0][i-
k]*userfunc[i]*tstep;
  }
  }
  else
  for(i=0;i<RootCount;i++)
  {
  QSOL=Qsol(i);
  GetCoeffOrigin();
```

```
  for(k=1;k<PointSolCnt;k++)
SolMtr[i][k]=GetValue(tc[k]);
  QSOL=Q;
  }
 ofstream os("solmtr.txt");os<<SolMtr;
 }

    void DifferentialEquation::FreqChar(void)
   {
   FreqChMtr=dmatrix(5,PointSolCnt);
   dcomplex jone=dcomplex(0.0,1.0);//мнимая единица
   //QSOL=PR;PSOL=PL;
   //
   double KU0=fabs(real(PR[0]/PL[0])); /*Усиление
на нулевой частоте */
   double KUZ=Cufz*KU0;
   double wstep=1.0,QW,PW,KU=KU0,RQ,IQ,RP,IP;
   if(PR.getm()==1) QW=real(PR[0]);
   //Подбираем частоту среза
   for(w=wstep;;w+=wstep)
   {
   if(PR.getm()>1)
    QW=sqrt(real(PR(w*jone))*real(PR(w*jone))+
            imag(PR(w*jone))*imag(PR(w*jone)));
    PW=sqrt(real(PL(w*jone))*real(PL(w*jone))+
            imag(PL(w*jone))*imag(PL(w*jone)));
    KU=QW/P       //Усиление на текущей частоте
    if((KU< 1.1*KUZ) && (KU>0.9*KUZ))break;
    if((KU<0.9*KUZ) && (wstep>0)) wstep=-0.1*wstep;
    if((KU>1.1*KUZ) && (wstep<0)) wstep=-0.1*wstep;
   }
    /*Заполним вектор частоты в матрице - мы разме-
стим его в 0-й строке */
    wstep=w/PointSolCnt;
    int i;
    for(i=0;i<PointSolCnt;i++)
FreqChMtr[0][i]=i*wstep;
    //Вычисляем  частотные характеристики
    if(PR.getm()==1) {RQ=real(PR[0]);IQ=0;}
    for(i=0;i<PointSolCnt;i++)
    {if(PR.getm()>1)
```

```
{RQ=real(PR(FreqChMtr[0][i]*jone));IQ=imag(PR(FreqC
hMtr[0][i]*jone));}

RP=real(PL(FreqChMtr[0][i]*jone));IP=imag(PL(FreqCh
Mtr[0][i]*jone));
     FreqChMtr[1][i]=(RQ*RP+IQ*IP)/(RP*RP+IP*IP);
/*Вещественная частотная */
     FreqChMtr[2][i]=(IQ*RP-RQ*IP)/(RP*RP+IP*IP);
/*Мнимая частотная */
FreqChMtr[3][i]=sqrt(FreqChMtr[1][i]*FreqChMtr[1][i
]+FreqChMtr[2][i]*FreqChMtr[2][i]);  //Амплитудная
     if(FreqChMtr[1][i]>0)
     FreqChMtr[4][i]=
atan(FreqChMtr[2][i]/FreqChMtr[1][i]); //Фазовая
     if(FreqChMtr[1][i]<=0)
     FreqChMtr[4][i]=-M_PI+
atan(FreqChMtr[2][i]/FreqChMtr[1][i]);//Фазовая;
    }
    ofstream os1("freqchar.txt");os1<<FreqChMtr;
    }
```

13.1.2. Форма основной программы

```
//LDEU7.DFM

object F1: TF1
  Left = 50
  Top = 118
  Width = 691
  Height = 344
  Caption = 'Лаб. раб. №7. Комплексный анализ  ди-
намики линейных систем. '
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  Menu = M0
  PixelsPerInch = 96
  TextHeight = 13
  object M0: TMainMenu
    Left = 360
```

```
Top = 16
object M1: TMenuItem
  Caption = 'Ввод коэффициентов уравнения'
  OnClick = M1Click
end
object M2: TMenuItem
  Caption = 'Вывод графика  движения'
  object N3: TMenuItem
    Caption = 'Свободное движение'
    OnClick = M20Click
  end
  object M21: TMenuItem
    Caption = 'Импульс'
    OnClick = M21Click
  end
  object N2: TMenuItem
    Caption = 'Ступенька'
    OnClick = M22Click
  end
  object M23: TMenuItem
    Caption = 'Синусоида'
    OnClick = M23Click
  end
  object M24: TMenuItem
    Caption = 'Косинусоида'
    OnClick = M24Click
  end
  object M25: TMenuItem
    Caption = 'Экспонента'
    OnClick = M25Click
  end
  object N1: TMenuItem
    Caption = 'Произвольное возмущение'
    OnClick = M26Click
  end
end
object M3: TMenuItem
  Caption = 'Частотные характеристики'
  object M31: TMenuItem
    Caption = 'Вещественная'
    OnClick = M31Click
  end
  object M32: TMenuItem
```

```
      Caption = 'Мнимая'
      OnClick = M32Click
    end
    object M33: TMenuItem
      Caption = 'Амплитудная'
      OnClick = M33Click
    end
    object M34: TMenuItem
      Caption = 'Фазовая'
      OnClick = M34Click
    end
    object M35: TMenuItem
      Caption = 'Амплитудно-фазовая'
      OnClick = M35Click
    end
  end
  object M4: TMenuItem
    Caption = 'Фазовый портрет'
    object M41: TMenuItem
      Caption = 'При свободном движении'
      OnClick = M41Click
    end
    object M42: TMenuItem
      Caption = 'При импульсе на входе'
      OnClick = M42Click
    end
    object M43: TMenuItem
      Caption = 'При ступеньке на входе'
      OnClick = M43Click
    end
    object M44: TMenuItem
      Caption = 'При синусоиде на входе'
      OnClick = M44Click
    end
    object M45: TMenuItem
      Caption = 'При косинусоиде'
      OnClick = M45Click
    end
    object M46: TMenuItem
      Caption = 'При экспоненте на входе'
      OnClick = M46Click
    end
    object M47: TMenuItem
```

```
        Caption = 'При произвольной функции'
        OnClick = M47Click
      end
    end
    object M5: TMenuItem
      Caption = 'Очистка области вывода'
      OnClick = M5Click
    end
  end
end
```

13.1.3. Модуль основной программы

```
//LDEU7.CPP

//---------------------------------------------
#include <vcl.h>
#include <conio.h>
#pragma hdrstop

#include "LDEU7.h"
#include "OKCANCL7.h"
#include "difequ7.h"
//--------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TF1 *F1;

int sc;                  //Количество точек решения
char Userfile[80]; /*Для имени файла с произвольным
возмущением */
int DeriveNumber=0;/*Порядок вычисляемой производ-
ной */
 /*Переменные для хранения размеров клиентской об-
ласти окна вывода */
int cxClient,cyClient;

int Cod=0,               //Код возмущения
Cod1=0, /*Дополнительный код для фаз. портрета или
ч.х. */
Rngl,Rngr;               //Порядок уравнения слева
dvector Cfnl,Cfnr,Nu;
```

```
double Tend,Tstep,  /*Конечное время решения и шаг
по времени */
Omega,Alpha;  //Параметры стандартных  возмущений
double Kos; /*Коэффициент усиления жесткой обратной
связи */
BOOL isdata=0;   //Признак ввода исходных данных
/*Прототип подпрограммы для пересчета натуральных
значений аргументов и функций */
//координаты точек в окне
void getCoord(dvector, dvector);

__fastcall TF1::TF1(TComponent* Owner):
TForm(Owner)
{
}

void __fastcall TF1::M1Click(TObject *Sender)
{
/*Выводим диалоговое окно для приема данных от
пользователя */
 Dlg1->ShowModal();
}

const RBUF=100,RTMP=10; /*Размеры буферов приема
данных */
char* buf, *tmp;         //Адреса буферов
HPEN* hpen;              // Перья
struct PT                // Координаты точек
{
 int x;
 int y;
};
PT* pt;                  //Массив точек
POINT *ptPnt;
 void __fastcall TDlg1::OKBtnClick(TObject *Sender)
{  int i;
   buf=new char[RBUF+2];
   buf=new char[RTMP+2];
//Частота гармонических возмущений
E9->GetTextBuf(buf,RBUF);
Omega=atof(buf);
//Показатель степени экспоненциального воздействия
E10->GetTextBuf(buf,RBUF);
```

```
Alpha=atof(buf);
//Коэффициент усиления на частоте среза
E11->GetTextBuf(buf,RBUF);
Cufz=atof(buf);

//Коэффициент усиления в контуре обратной связи
E13->GetTextBuf(buf,RBUF);
Kos=atof(buf);

/*Определяем введенные значения  конечного времени
и шага */
E4->GetTextBuf(buf,RBUF);
Tend=atof(buf);
if(Tend<=0)//Отказываемся выполнять невыполнимое
{
Application-> MessageBox("Ошибочное конечное значе-
ние аргумента", "Ошибка ввода",
MB_OK|MB_ICONERROR);
delete[] buf;delete[] tmp;return;
}

E5->GetTextBuf(buf,RBUF);
Tstep=atof(buf);
if(Tstep<=0)//Отказываемся выполнять невыполнимое
{
Application-> MessageBox("Ошибочен шаг по аргумен-
ту","Ошибка ввода", MB_OK|MB_ICONERROR);
delete[] buf;delete[] tmp;return;
}
//Количество точек в решении
sc=floor(Tend/Tstep);
if(sc<=50){ Application->MessageBox("Мало шагов ре-
шения","Ошибка ввода", MB_OK|MB_ICONEXCLAMATION);
return;}

/*После получения количества точек определяются
размеры массивов для хранения информации */
userfunc=dvector(sc);
//Имя файла с произвольным возмущением
E12->GetTextBuf(buf,RBUF);
strcpy(Userfile,buf);
//Перепишем порядок уравнения из E1->TextBuf в buf
E1->GetTextBuf(buf,RBUF);
```

```
Rngl=atoi(buf);
if(Rngl<=0)
{
Application-> MessageBox("Не введен или 0 порядок
уравнения", "Ошибка ввода",MB_OK|MB_ICONERROR);
delete[] buf;delete[] tmp; return;
}
/*Создаем вектор коэффициентов уравнения, вектор
начальных условий и определяем попутно размерность
введенных векторов */
Cfnl=dvector(Rngl+1);Nu=dvector(Rngl);
double cfntmp;
E2->GetTextBuf(buf,RBUF);
//Разборка строки
tmp=strtok(buf," ");
for(i=0;tmp!=NULL;)
{
cfntmp=atof(tmp);
if(i==0 && cfntmp==0){tmp=strtok(NULL,"
");continue;}
else { Cfnl[i]=cfntmp;tmp=strtok(NULL," "); i++; }
}
if(i!=(Rngl+1))/*Отказываемся выполнять невыполни-
мое */
{
Application-> MessageBox("Количество коэффициентов
слева не соответствует порядку уравнения","Ошибка
ввода",MB_OK|MB_ICONERROR);
delete[] buf;delete[] tmp;return;
}
E3->GetTextBuf(buf,RBUF);
tmp=strtok(buf," ");
for(i=0;tmp!=NULL;i++) {Nu[i]=atof(tmp);
tmp=strtok(NULL," ");}
if(i!=Rngl)//Отказываемся выполнять невыполнимое
{
Application-> MessageBox("Количество начальных
условий не соответствует порядку уравнения сле-
ва","Ошибка ввода",MB_OK|MB_ICONERROR);
delete[] buf;delete[] tmp;return;
}
// Перепишем порядок уравнения  справа
E7->GetTextBuf(buf,RBUF);
```

```
Rngr=atoi(buf);
if(Rngr<0 || Rngr>Rngl)
{
Application-> MessageBox("Порядок справа больше чем
слева", "Ошибка ввода",MB_OK|MB_ICONERROR);
delete[] buf;delete[] tmp;  return;
}
/*Создаем вектор коэффициентов уравнения, вектор
начальных условий и определяем попутно размерность
введенных векторов */
Cfnr=dvector(Rngr+1);
E8->GetTextBuf(buf,RBUF);
//Разборка строки
tmp=strtok(buf," ");
for(i=0;tmp!=NULL;)
{
cfntmp=atof(tmp);
if(i==0 && cfntmp==0){tmp=strtok(NULL,"
");continue;}
else { Cfnr[i]=cfntmp;tmp=strtok(NULL," "); i++; }
}
if(i!=(Rngr+1))/*Отказываемся выполнять невыполни-
мое*/
 {
Application-> MessageBox("Количество коэффициентов
справа не соответствует порядку уравнения","Ошибка
ввода",MB_OK|MB_ICONERROR);
delete[] buf;delete[] tmp;return;
}
/*Порядок производной, график которой надо отрисо-
вать рядом с функцией решения */
E6->GetTextBuf(buf,RBUF);
DeriveNumber=atoi(buf);
if(DeriveNumber<0 || DeriveNumber>(Rngl-1))
{
Application-> MessageBox("Не корректен порядок про-
изводной", "Ошибка ввода",MB_OK|MB_ICONERROR);
delete[] buf;delete[] tmp; return;
}
delete[] buf;delete[] tmp;
isdata=1;
}
```

```
/* Для пересчета натуральных единиц в пиксельные
координаты для построения графиков переходных про-
цессов, фазовых портретов и частотных характеристик
напишем подпрограмму getCoord - в качестве аргумен-
тов ей понадобятся ссылка массивы значений функции
и аргумента. Прежде всего подпрограмма должна вы-
числить диапазон изменения значений функции и коли-
чество пикселов на одну натуральную единицу по обе-
им осям. */

struct OFF{long x,y;}off; /*Для смещения координат-
ных осей */

/*Подпрограмма для пересчета натуральных значений
аргументов и функций координаты точек в окне */
void getCoord(dvector y, dvector x)
{int i, sc=x.getm();

/*Для графического отображения функций решения
определим массив структур типа POINT, в который по-
том перенесем пересчитанные в координаты значения
отображаемой функции и значения ее аргумента */
 pt=new PT[sc];
 ptPnt=new POINT[sc];
  /*Определим наибольшее, наименьшее значение функ-
ции, диапазон изменения- хотя было бы неплохо полу-
чать эти значения с помощью методов класса vector -
но мы их не написали */

 double fymax,fymin,fxmax,fxmin;
 fymax=fymin=y[0];fxmax=fxmin=x[0];/*Для начала
пусть так */
 for(i=0;i<sc;i++)
 {if(y[i]>fymax)fymax=y[i];
if(y[i]<fymin)fymin=y[i];
  if(x[i]>fxmax)fxmax=x[i];
if(x[i]<fxmin)fxmin=x[i];
 }
 //Отцентрируем все данные относительно минимумов
 for(i=0;i<sc;i++)
 {x[i]-=fxmin;y[i]-=fymin;}

 //Теперь заполним массив структур с координатами
```

```
 long px,py;

 if(fxmax!=fxmin && fymax!=fymin)
 {for(i=0;i<sc;i++)
 {px=x[i]*(double)F1->ClientWidth/(fxmax-fxmin);
  py=F1->ClientHeight-y[i]*(double)F1->
ClientHeight/(fymax-fymin);
  pt[i].x=px;
  pt[i].y=py;
  ptPnt[i]=Point(px,py);
  }
 }

 if((fxmax-fxmin)!=0)
 off.x=fxmin*((double)F1->ClientWidth/(fxmax-
fxmin));
 if((fymax-fymin)!=0)
 off.y=fymin*((double)F1->ClientHeight/(fymax-
fymin));
 }


void __fastcall TF1::M5Click(TObject *Sender)
{
Invalidate();
}

//Функция для рисования переходных процессов
void ProcessPaint(int fnc,TCanvas * Canvas)
{
  //Если данные не вводились
  if(!isdata)
  {Application->MessageBox("Вы не ввели параметры
системы","Отсутствие данных",
MB_OK|MB_ICONEXCLAMATION); return;}
 //Создаем объект - уравнение
  DifferentialEquation Dequ(Cfnl,Cfnr, Nu,fnc,
Omega,Alpha,Tend, Tstep, Kos);
  //Вызываем метод решения,получая матрицу решения
  Dequ.Sol();
//Рисуем сначала функцию  решения
  char* s1="------График движения",
      * s2="------Производная";
```

```
  //Canvas->Pen->Color=clGreen;
  randomize();
  Canvas->Pen->Color = (Graphics::TColor)
random(65535);
  Canvas->Font->Color=Canvas->Pen->Color;
  Canvas->TextOut(200,Canvas->TextHeight(s1),s1);
  if(fnc==6) getCoord(SolMtr[1],tc);
  else getCoord(SolMtr[0],tc);
  Canvas-> Pen->Width=5;
  Canvas-> PolyBezier(ptPnt,3*(((sc-4)/3)+1));
//Затем рисуем производную решения
if(fnc!=6){
  Canvas->Pen->Color=(Graphics::TColor)
random(65535);
  Canvas->Font->Color=Canvas->Pen->Color;
  Canvas->TextOut(200,2*Canvas->TextHeight(s2),s2);
  getCoord(SolMtr[DeriveNumber],tc);
  Canvas->Pen->Width=5;
  Canvas->PolyBezier(ptPnt,3*(((sc-4)/3)+1));
  }
  delete[] pt; delete[] ptPnt;

}

void __fastcall TF1::M21Click(TObject *Sender)
{
int fnc=1;
ProcessPaint(fnc,Canvas);
}

void __fastcall TF1::M22Click(TObject *Sender)
{
Cod=2;
ProcessPaint(Cod,Canvas);
}

void __fastcall TF1::M23Click(TObject *Sender)
{
 if(Omega==0.0)
 {Application->MessageBox("Не задана частота сину-
соиды","Ошибка в данных",
MB_OK|MB_ICONEXCLAMATION); return;}
```

```
Cod=3;
ProcessPaint(Cod,Canvas);
}

void __fastcall TF1::M24Click(TObject *Sender)
{
 if(Omega==0.0)
 {Application->MessageBox("Не задана частота коси-
нусоиды","Ошибка в данных",
 MB_OK|MB_ICONEXCLAMATION);return;}
Cod=4;
ProcessPaint(Cod,Canvas);
}

void __fastcall TF1::M25Click(TObject *Sender)
{
 if(Alpha==0.0)
 {Application->MessageBox("Не задан показатель экс-
поненты","Ошибка в данных",
 MB_OK|MB_ICONEXCLAMATION);return;}
Cod=5;
ProcessPaint(Cod,Canvas);
}

void __fastcall TF1::M26Click(TObject *Sender)
{
if(!strlen(Userfile))
 {Application->MessageBox("Для произвольного возму-
щения не задано имя файла ", "Ошибка в данных",
MB_OK|MB_ICONEXCLAMATION);return;
 }
ifstream uf(Userfile);
/*Если файл открыт - читаем из него  в вектор
userfunc */
if(uf) {uf>>userfunc;uf.close();}
else
 {Application->MessageBox("Неудача при открытии
файла с произвольным возмущением.\n\Вычисляем по
имитации","",MB_OK|MB_ICONEXCLAMATION);
 for(int i=0;i<sc;i++)
 userfunc[i]= exp(-0.02*i*Tstep)*cos(i*Tstep*0.05);
 }
Cod=6;
```

```
ProcessPaint(Cod,Canvas);
}

void __fastcall TF1::M20Click(TObject *Sender)
{
Cod=0;
ProcessPaint(Cod,Canvas);
}

//Функция для рисования частотных характеристик
void PaintFreqChar(int cod,TCanvas* Canvas)
{
  //Если данные не вводились
  if(!isdata)
  {Application->MessageBox("Вы не ввели параметры
системы","Отсутствие данных",
MB_OK|MB_ICONEXCLAMATION);return;}
  if(Cufz<=0.0)
 {Application->MessageBox("Для частотных характери-
стик не задана ненулевая частота среза ", "Ошибка в
данных", MB_OK|MB_ICONEXCLAMATION);return;}
  int fnc=7;
//Создаем объект - уравнение
  DifferentialEquation Dequ(Cfnl,Cfnr,Nu,fnc,
Omega,Alpha,Tend, Tstep, Kos);
  //Вызываем метод решения, получая матрицу решения
  Dequ.FreqChar();
 if(cod>=31 && cod<=34)
  {
  getCoord(FreqChMtr[(cod-30)],FreqChMtr[0]);
  Canvas->Pen->Width=5;
  Canvas->Pen->Color = (Graphics::TColor)
random(65535);

  for(int i=1;i<sc;i++)
  {
  Canvas->MoveTo(ptPnt[i-1].x,ptPnt[i].y);
  Canvas->LineTo(ptPnt[i].x,ptPnt[i+1].y);
  }
  //Canvas-> PolyBezier(ptPnt,3*(((sc-4)/3)+1)+1);
  }
  if(cod==35)
  {
```

```
  getCoord(FreqChMtr[2],FreqChMtr[1]);
  Canvas->Pen->Color = (Graphics::TColor)
random(65535);
  Canvas->Pen->Width=5;
  for(int i=1;i<sc;i++)
  {
  Canvas->MoveTo(ptPnt[i-1].x,ptPnt[i].y);
  Canvas->LineTo(ptPnt[i].x,ptPnt[i+1].y);
  }
  //Canvas->PolyBezier(ptPnt,3*(((sc-4)/3)+1));
  }
  delete[] pt; delete[] ptPnt;
}
void __fastcall TF1::M31Click(TObject *Sender)
{
Cod1=31;
PaintFreqChar(Cod1,Canvas);
}


void __fastcall TF1::M32Click(TObject *Sender)
{
Cod1=32;
PaintFreqChar(Cod1,Canvas);
}


void __fastcall TF1::M33Click(TObject *Sender)
{
Cod1=33;
PaintFreqChar(Cod1,Canvas);
}


void __fastcall TF1::M34Click(TObject *Sender)
{
Cod1=34;PaintFreqChar(Cod1,Canvas);
}


void __fastcall TF1::M35Click(TObject *Sender)
{
Cod1=35;PaintFreqChar(Cod1,Canvas);
```

```
}

//Функция для рисования фазовых портретов
void PaintPortret(int cod,TCanvas* Canvas)
{
//Если данные не вводились
  if(!isdata)
  {Application->MessageBox("Вы не ввели параметры
системы","Отсутствие данных",
MB_OK|MB_ICONEXCLAMATION);return;}
int fnc=cod-40;
 //Создаем объект - уравнение
  DifferentialEquation Dequ(Cfnl,Cfnr, Nu,fnc,
Omega,Alpha,Tend, Tstep,Kos);
  //Вызываем метод решения,получая матрицу решения
  Dequ.Sol();
  getCoord(SolMtr[DeriveNumber],
SolMtr[DeriveNumber-1]);
  Canvas->Pen->Width=5;
  Canvas->Pen->Color = (Graphics::TColor)
random(65535);
  Canvas->PolyBezier(ptPnt,3*(((sc-4)/3)+1));
  delete[] pt; delete[] ptPnt;
}


void __fastcall TF1::M41Click(TObject *Sender)
{
Cod1=40;
PaintPortret(Cod1,Canvas);
}


void __fastcall TF1::M42Click(TObject *Sender)
{
Cod1=41;PaintPortret(Cod1,Canvas);
}


void __fastcall TF1::M43Click(TObject *Sender)
{
Cod1=42;PaintPortret(Cod1,Canvas);
}
```

```
void __fastcall TF1::M44Click(TObject *Sender)
{
Cod1=43;PaintPortret(Cod1,Canvas);
}


void __fastcall TF1::M45Click(TObject *Sender)
{
Cod1=44;PaintPortret(Cod1,Canvas);
}

void __fastcall TF1::M46Click(TObject *Sender)
{
Cod1=45;PaintPortret(Cod1,Canvas);
}

void __fastcall TF1::M47Click(TObject *Sender)
{
Cod1=46;PaintPortret(Cod1,Canvas);
}
```

13.1.4. Форма ввода данных

```
//OKCANCL.DFM


object Dlg1: TDlg1
  Left = 85
  Top = 115
  BorderStyle = bsDialog
  Caption = 'Ввод уравнения линейной системы'
  ClientHeight = 402
  ClientWidth = 616
  ParentFont = True
  Position = poScreenCenter
  PixelsPerInch = 96
  TextHeight = 13
  object Dlg1: TBevel
    Left = 8
    Top = -8
    Width = 745
```

```
  Height = 201
  Shape = bsFrame
end
object OKBtn: TButton
  Left = 255
  Top = 372
  Width = 75
  Height = 25
  Caption = 'OK'
  Default = True
  Font.Charset = RUSSIAN_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Courier New'
  Font.Style = []
  ModalResult = 1
  ParentFont = False
  TabOrder = 15
  OnClick = OKBtnClick
end
object ST1: TStaticText
  Left = 24
  Top = 16
  Width = 137
  Height = 17
  Caption = 'Порядок уравнения слева'
  TabOrder = 14
end
object ST2: TStaticText
  Left = 24
  Top = 40
  Width = 171
  Height = 17
  Caption = 'Коэффициенты уравнения слева'
  TabOrder = 16
end
object E2: TEdit
  Left = 224
  Top = 40
  Width = 65
  Height = 21
  TabOrder = 2
  Text = '40 4 1'
```

```
  end
  object ST3: TStaticText
    Left = 24
    Top = 64
    Width = 105
    Height = 17
    Caption = 'Начальные условия'
    TabOrder = 17
  end
  object E3: TEdit
    Left = 224
    Top = 64
    Width = 65
    Height = 21
    TabOrder = 3
    Text = '0.1 -0.1'
  end
  object Memo1: TMemo
    Left = 8
    Top = 200
    Width = 745
    Height = 169
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clNavy
    Font.Height = -11
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    Lines.Strings = (
      'Лаб. раб. №7. Задание и комментарий.')
    ParentFont = False
    ReadOnly = True
    TabOrder = 0
  end
  object E1: TEdit
    Left = 224
    Top = 16
    Width = 65
    Height = 21
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -11
    Font.Name = 'MS Sans Serif'
    Font.Style = []
```

```
  ParentFont = False
  TabOrder = 1
  Text = '2'
end
object St4: TStaticText
  Left = 24
  Top = 88
  Width = 187
  Height = 17
  Caption = 'Максимальное значение аргумента'
  TabOrder = 18
end
object ST5: TStaticText
  Left = 24
  Top = 112
  Width = 187
  Height = 17
  Caption = 'Шаг по аргументу функции решения'
  TabOrder = 19
end
object E4: TEdit
  Left = 224
  Top = 88
  Width = 65
  Height = 21
  TabOrder = 4
  Text = '500'
end
object E5: TEdit
  Left = 224
  Top = 112
  Width = 65
  Height = 21
  TabOrder = 5
  Text = '1'
end
object E6: TEdit
  Left = 224
  Top = 136
  Width = 65
  Height = 21
  TabOrder = 6
  Text = '1'
```

```
  end
  object ST6: TStaticText
    Left = 24
    Top = 136
    Width = 189
    Height = 17
    Caption = 'Порядок вычисляемой производной'
    TabOrder = 20
  end
  object E7: TEdit
    Left = 536
    Top = 16
    Width = 65
    Height = 21
    TabOrder = 7
    Text = '0'
  end
  object E8: TEdit
    Left = 536
    Top = 40
    Width = 65
    Height = 21
    TabOrder = 8
    Text = '1'
  end
  object E9: TEdit
    Left = 536
    Top = 64
    Width = 65
    Height = 21
    TabOrder = 9
    Text = '0.05'
  end
  object E10: TEdit
    Left = 536
    Top = 88
    Width = 65
    Height = 21
    TabOrder = 10
    Text = '0.05'
  end
  object E11: TEdit
    Left = 536
```

```
  Top = 112
  Width = 65
  Height = 21
  TabOrder = 11
  Text = '0.01'
end
object ST7: TStaticText
  Left = 336
  Top = 16
  Width = 143
  Height = 17
  Caption = 'Порядок уравнения справа'
  TabOrder = 21
end
object ST8: TStaticText
  Left = 336
  Top = 40
  Width = 177
  Height = 17
  Caption = 'Коэффициенты уравнения справа'
  TabOrder = 22
end
object ST9: TStaticText
  Left = 336
  Top = 64
  Width = 193
  Height = 17
  Caption = 'Частота гармонических возмущений'
  TabOrder = 23
end
object ST10: TStaticText
  Left = 336
  Top = 88
  Width = 137
  Height = 17
  Caption = 'Показатель у экспоненты'
  TabOrder = 24
end
object ST11: TStaticText
  Left = 336
  Top = 112
  Width = 144
  Height = 17
```

```
    Caption = 'Усиление на частоте среза'
    TabOrder = 25
  end
  object E12: TEdit
    Left = 480
    Top = 136
    Width = 121
    Height = 21
    TabOrder = 12
    Text = 'func.txt'
  end
  object ST12: TStaticText
    Left = 336
    Top = 136
    Width = 144
    Height = 17
    Caption = 'Имя файла с произв. возм.'
    TabOrder = 26
  end
  object E13: TEdit
    Left = 224
    Top = 160
    Width = 65
    Height = 21
    TabOrder = 13
  end
  object ST13: TStaticText
    Left = 24
    Top = 160
    Width = 146
    Height = 17
    Caption = 'Усиление в обратной связи'
    TabOrder = 27
  end
end
```

13.1.5. Заголовочный файл модуля ввода данных

```
//OKCANCL.H

#ifndef OCBH
#define OCBH
```

```
#include <System.hpp>
#include <Windows.hpp>
#include <SysUtils.hpp>
#include <Classes.hpp>
#include <Graphics.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Controls.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>

class TDlg1 : public TForm
{
__published:
      TButton *OKBtn;
        TBevel *Dlg1;
        TStaticText *ST1;
        TStaticText *ST2;
        TEdit *E2;
        TStaticText *ST3;
        TEdit *E3;
        TMemo *Memo1;
        TEdit *E1;
        TStaticText *St4;
        TStaticText *ST5;
        TEdit *E4;
        TEdit *E5;
        TEdit *E6;
        TStaticText *ST6;
        TEdit *E7;
        TEdit *E8;
        TEdit *E9;
        TEdit *E10;
        TEdit *E11;
        TStaticText *ST7;
        TStaticText *ST8;
        TStaticText *ST9;
        TStaticText *ST10;
        TStaticText *ST11;
        TEdit *E12;
        TStaticText *ST12;
        TEdit *E13;
        TStaticText *ST13;
```

```
        void __fastcall OKBtnClick(TObject
*Sender);


private:
public:
      virtual __fastcall TDlg1(TComponent* AOwner);
};

extern PACKAGE TDlg1 *Dlg1;

#endif
```

13.1.6. Модуль ввода данных

```
//OKCANCL.CPP


#include <vcl.h>
#include <stdlib.h>
#pragma hdrstop

#include "OKCANCL7.h"

#pragma resource "*.dfm"
TDlg1 *Dlg1;


__fastcall TDlg1::TDlg1(TComponent* AOwner)
      : TForm(AOwner)
{

}
```

13.1.7. Заголовочный файл инициализационного модуля

```
//LDEU7.H


#ifndef LDEUH
#define LDEUH

#include <Classes.hpp>
```

```
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>

class TF1 : public TForm
{
__published:      // IDE-managed Components
        TMainMenu *M0;
        TMenuItem *M1;
        TMenuItem *M2;
        TMenuItem *M5;
        TMenuItem *M21;
        TMenuItem *N2;
        TMenuItem *M23;
        TMenuItem *M24;
        TMenuItem *M25;
        TMenuItem *N1;
        TMenuItem *N3;
        TMenuItem *M3;
        TMenuItem *M31;
        TMenuItem *M32;
        TMenuItem *M33;
        TMenuItem *M34;
        TMenuItem *M35;
        TMenuItem *M4;
        TMenuItem *M41;
        TMenuItem *M42;
        TMenuItem *M43;
        TMenuItem *M44;
        TMenuItem *M45;
        TMenuItem *M46;
        TMenuItem *M47;
        void __fastcall M1Click(TObject *Sender);

        void __fastcall M5Click(TObject *Sender);
        void __fastcall M21Click(TObject *Sender);
        void __fastcall M22Click(TObject *Sender);
        void __fastcall M23Click(TObject *Sender);
        void __fastcall M24Click(TObject *Sender);
        void __fastcall M25Click(TObject *Sender);
        void __fastcall M26Click(TObject *Sender);
        void __fastcall M20Click(TObject *Sender);
```

```
        void __fastcall M31Click(TObject *Sender);
        void __fastcall M32Click(TObject *Sender);
        void __fastcall M33Click(TObject *Sender);
        void __fastcall M34Click(TObject *Sender);
        void __fastcall M35Click(TObject *Sender);

        void __fastcall M41Click(TObject *Sender);
        void __fastcall M42Click(TObject *Sender);
        void __fastcall M43Click(TObject *Sender);
        void __fastcall M44Click(TObject *Sender);
        void __fastcall M45Click(TObject *Sender);
        void __fastcall M46Click(TObject *Sender);
        void __fastcall M47Click(TObject *Sender);
private:    // User declarations
public:           // User declarations
        __fastcall TF1(TComponent* Owner);
};

extern PACKAGE TF1 *F1;

#endif
```

13.1.8. Инициализационный модуль

```
//LDEP7.CPP


#include <vcl.h>
#pragma hdrstop

USEFORM("LDEU7.cpp", F1);
USEFORM("OKCANCL7.cpp", Dlg1);
//---------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                Application->Initialize();
Application->CreateForm(__classid(TF1), &F1);
Application->CreateForm(__classid(TDlg1), &Dlg1);
                Application->Run();
        }
        catch (Exception &exception)
```

```
        {
Application->ShowException(&exception);
        }
        return 0;
}
```

13.1.9. Файл проекта

```
#LDEP7.BPR


!if !$d(BCB)
BCB = $(MAKEDIR)\..
!endif

# ---------------------------
# IDE SECTION
# ---------------------------
# The following section of the project makefile is
# managed by the BCB IDE.
# It is recommended to use the IDE to change any of
# the values in this
# section.
# ---------------------------

VERSION = BCB.03
# ---------------------------
PROJECT = LDEP7.exe
OBJFILES = LDEP7.obj LDEU7.obj OKCANCL7.obj
RESDEPEN = $(RESFILES) LDEU7.dfm OKCANCL7.dfm
LIBFILES =
LIBRARIES =
SPARELIBS = VCL35.lib
PACKAGES = vclx35.bpi VCL35.bpi vcldb35.bpi
vcldbx35.bpi bcbsmp35.bpi dclocx35.bpi \
  Qrpt35.bpi teeui35.bpi teedb35.bpi tee35.bpi
ibsmp35.bpi dss35.bpi NMFast35.bpi \
  inetdb35.bpi inet35.bpi VclMid35.bpi
DEFFILE =
# ---------------------------
PATHCPP = .;
PATHASM = .;
PATHPAS = .;
```

```
PATHRC = .;
DEBUGLIBPATH = $(BCB)\lib\debug
RELEASELIBPATH = $(BCB)\lib\release
# ---------------------------
CFLAG1 = -Od -Hc -w -r- -k -y -v -vi- -c -tWC
CFLAG2 = -DUSEPACKAGES -
I..\objrepos;$(BCB)\include;$(BCB)\include\vcl -
H="c:\Program
Files\Borland\CBuilder3\\lib\vcl35.csm"
CFLAG3 = -Jgd -Tkh30000 -5
PFLAGS = -DUSEPACKAGES \
  -
U..\examples\controls\source;..\objrepos;$(BCB)\lib
;$(BCB)\lib\obj;$(DEBUGLIBPATH) \
  -I..\objrepos;$(BCB)\include;$(BCB)\include\vcl -
$Y -$W -$O- -v -JPHN -M
RFLAGS = -DUSEPACKAGES -
i..\objrepos;$(BCB)\include;$(BCB)\include\vcl
AFLAGS = /i..\objrepos /i$(BCB)\include
/i$(BCB)\include\vcl /dUSEPACKAGES /mx /w2 /zi
LFLAGS = -
L..\examples\controls\source;..\objrepos;$(BCB)\lib
;$(BCB)\lib\obj;$(DEBUGLIBPATH) \
  -S:0x1000000 -Sc:0x20000 -aa -Tpe -x -Gn -v
IFLAGS =
# ---------------------------
ALLOBJ = c0w32.obj $(PACKAGES) $(OBJFILES)
ALLRES = $(RESFILES)
ALLLIB = $(LIBFILES)  import32.lib cw32mt.lib
# ---------------------------
!ifdef IDEOPTIONS

[Version Info]
IncludeVerInfo=0
AutoIncBuild=0
MajorVer=1
MinorVer=0
Release=0
Build=0
Debug=0
PreRelease=0
Special=0
Private=0
```

```
DLL=0
Locale=1049
CodePage=1251

[Version Info Keys]
CompanyName=
FileDescription=Executable (Console)
FileVersion=1.0.0.0
InternalName=
LegalCopyright=
LegalTrademarks=
OriginalFilename=
ProductName=
ProductVersion=1.0.0.0
Comments=

[HistoryLists\hlIncludePath]
Count=4
Item0=..\objrepos;$(BCB)\include;$(BCB)\include\vcl
Item1=C:\PROGRA~1\BORLAND\CBUILD~1\include;C:\PROGR
A~1\BORLAND\CBUILD~1\include\vcl
Item2=C:\PROGRA~1\BORLAND\CBUILD~1\include
Item3=$(BCB)\include

[HistoryLists\hlLibraryPath]
Count=5
Item0=..\examples\controls\source;..\objrepos;$(BCB
)\lib;$(BCB)\lib\obj
Item1=..\objrepos;$(BCB)\lib;$(BCB)\lib\obj
Item2=C:\PROGRA~1\BORLAND\CBUILD~1\lib\obj;C:\PROGR
A~1\BORLAND\CBUILD~1\lib
Item3=C:\PROGRA~1\BORLAND\CBUILD~1\lib;C:\PROGRA~1\
BORLAND\CBUILD~1\Projects
Item4=$(BCB)\lib;C:\PROGRA~1\BORLAND\CBUILD~1\Proje
cts

[HistoryLists\hlDebugSourcePath]
Count=1
Item0=C:\PROGRA~1\BORLAND\CBUILD~1\Projects

[HistoryLists\hlConditionals]
Count=5
Item0=USEPACKAGES;_RTLDLL
```

```
Item1=USEPACKAGES
Item2=_RTLDLL;USEPACKAGES
Item3=_NO_VCL;USEPACKAGES
Item4=_NO_VCL

[HistoryLists\hlIntOutputDir]
Count=1
Item0=C:\PROGRA~1\BORLAND\CBUILD~1\Projects

[Debugging]
DebugSourceDirs=

[Parameters]
RunParams=
HostApplication=

!endif

# ---------------------------
# MAKE SECTION
# ---------------------------
# This section of the project file is not used by
# the BCB IDE.  It is for the benefit of building
# from the command-line using the MAKE utility.
# ---------------------------

.autodepend
# ---------------------------
!if !$d(BCC32)
BCC32 = bcc32
!endif

!if !$d(DCC32)
DCC32 = dcc32
!endif

!if !$d(TASM32)
TASM32 = tasm32
!endif

!if !$d(LINKER)
LINKER = ilink32
!endif
```

```
!if !$d(BRCC32)
BRCC32 = brcc32
!endif
# ---------------------------
!if $d(PATHCPP)
.PATH.CPP = $(PATHCPP)
.PATH.C   = $(PATHCPP)
!endif

!if $d(PATHPAS)
.PATH.PAS = $(PATHPAS)
!endif

!if $d(PATHASM)
.PATH.ASM = $(PATHASM)
!endif

!if $d(PATHRC)
.PATH.RC  = $(PATHRC)
!endif
# ---------------------------
$(PROJECT): $(OBJFILES) $(RESDEPEN) $(DEFFILE)
    $(BCB)\BIN\$(LINKER) @&&!
    $(LFLAGS) +
    $(ALLOBJ), +
    $(PROJECT),, +
    $(ALLLIB), +
    $(DEFFILE), +
    $(ALLRES)
!
# ---------------------------
.pas.hpp:
    $(BCB)\BIN\$(DCC32) $(PFLAGS) {$< }

.pas.obj:
    $(BCB)\BIN\$(DCC32) $(PFLAGS) {$< }

.cpp.obj:
    $(BCB)\BIN\$(BCC32) $(CFLAG1) $(CFLAG2)
$(CFLAG3) -n$(@D) {$< }

.c.obj:
```

```
    $(BCB)\BIN\$(BCC32) $(CFLAG1) $(CFLAG2)
$(CFLAG3) -n$(@D) {$< }

.asm.obj:
    $(BCB)\BIN\$(TASM32) $(AFLAGS) $<, $@

.rc.res:
    $(BCB)\BIN\$(BRCC32) $(RFLAGS) -fo$@ $<
# ---------------------------
```

13.2. Исходные тексты программы на языке Object Pascal, выполненной в среде Delphi 4

13.2.1. Форма изменения размеров пера

```
//PENW.DFM


object Form4: TForm4
  Left = 267
  Top = 272
  Width = 196
  Height = 129
  Caption = 'Толщина'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  Position = poScreenCenter
  PixelsPerInch = 96
  TextHeight = 13
  object BitBtn1: TBitBtn
    Left = 64
    Top = 72
    Width = 57
    Height = 25
    TabOrder = 0
    OnClick = BitBtn1Click
    Kind = bkOK
```

```
    Margin = 3
  end
  object SpinEdit1: TSpinEdit
    Left = 32
    Top = 32
    Width = 121
    Height = 22
    MaxValue = 10
    MinValue = 0
    TabOrder = 1
    Value = 0
  end
end
```

13.2.2. Модуль изменения размеров пера

```
//PENW.PAS


unit PenW;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs,
  StdCtrls, Spin, Buttons;

type
  TForm4 = class(TForm)
    BitBtn1: TBitBtn;
    SpinEdit1: TSpinEdit;
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form4: TForm4;
  WidP:integer;
implementation
```

```
{$R *.DFM}

procedure TForm4.BitBtn1Click(Sender: TObject);
begin
WidP:=SpinEdit1.Value;
end;

end.
```

13.2.3. Форма ввода данных

```
//INPUTD.DFM


object Form2: TForm2
  Left = 198
  Top = 113
  BorderStyle = bsToolWindow
  Caption = 'Ввод данных'
  ClientHeight = 411
  ClientWidth = 519
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  Position = poScreenCenter
  PixelsPerInch = 96
  TextHeight = 14
  object Bevel1: TBevel
    Left = 0
    Top = 0
    Width = 303
    Height = 411
    Align = alLeft
  end
  object StaticText1: TStaticText
    Left = 26
    Top = 9
    Width = 268
```

```
  Height = 18
  AutoSize = False
  BorderStyle = sbsSunken
  Caption = 'Порядок левой части уравнения'
  TabOrder = 0
end
object StaticText2: TStaticText
  Left = 26
  Top = 34
  Width = 268
  Height = 19
  AutoSize = False
  BorderStyle = sbsSunken
  Caption = 'Коэффициенты левой части уравнения'
  TabOrder = 1
end
object StaticText3: TStaticText
  Left = 26
  Top = 60
  Width = 268
  Height = 19
  AutoSize = False
  BorderStyle = sbsSunken
  Caption = 'Начальные условия'
  TabOrder = 2
end
object StaticText4: TStaticText
  Left = 26
  Top = 86
  Width = 268
  Height = 18
  AutoSize = False
  BorderStyle = sbsSunken
  Caption = 'Порядок правой части уравнения'
  TabOrder = 3
end
object StaticText5: TStaticText
  Left = 26
  Top = 112
  Width = 268
  Height = 18
  AutoSize = False
  BorderStyle = sbsSunken
```

```
    Caption = 'Коэффициенты правой части уравнения'
    TabOrder = 4
  end
  object StaticText6: TStaticText
    Left = 26
    Top = 138
    Width = 268
    Height = 18
    AutoSize = False
    BorderStyle = sbsSingle
    Caption = 'ЧИСЛА РАЗДЕЛЯЙТЕ ТОЛЬКО ПРОБЕЛАМИ'
    TabOrder = 5
  end
  object StaticText7: TStaticText
    Left = 26
    Top = 164
    Width = 268
    Height = 18
    AutoSize = False
    BorderStyle = sbsSunken
    Caption = 'Частота гармонических возмущений'
    TabOrder = 6
  end
  object StaticText8: TStaticText
    Left = 26
    Top = 190
    Width = 268
    Height = 18
    AutoSize = False
    BorderStyle = sbsSunken
    Caption = 'Показатель экспоненциального воздей-
ствия'
    TabOrder = 7
  end
  object StaticText10: TStaticText
    Left = 26
    Top = 218
    Width = 268
    Height = 18
    AutoSize = False
    BorderStyle = sbsSunken
    Caption = 'Усиление на частоте среза'
    TabOrder = 8
```

```
end
object StaticText11: TStaticText
  Left = 26
  Top = 243
  Width = 268
  Height = 19
  AutoSize = False
  BorderStyle = sbsSunken
  Caption = 'Конечное значение времени'
  TabOrder = 9
end
object Edit1: TEdit
  Left = 310
  Top = 9
  Width = 199
  Height = 22
  TabOrder = 10
  Text = '2'
end
object Edit2: TEdit
  Left = 310
  Top = 34
  Width = 199
  Height = 22
  TabOrder = 11
  Text = '40 4 1'
end
object Edit3: TEdit
  Left = 310
  Top = 60
  Width = 199
  Height = 22
  TabOrder = 12
  Text = '0,1 0,3'
end
object Edit4: TEdit
  Left = 310
  Top = 86
  Width = 199
  Height = 22
  TabOrder = 13
  Text = '0'
end
```

```
  object Edit5: TEdit
    Left = 310
    Top = 112
    Width = 199
    Height = 22
    TabOrder = 14
    Text = '1'
  end
  object Edit6: TEdit
    Left = 310
    Top = 164
    Width = 199
    Height = 22
    TabOrder = 15
    Text = '0,05'
  end
  object Edit7: TEdit
    Left = 310
    Top = 190
    Width = 199
    Height = 22
    TabOrder = 16
    Text = '0,05'
  end
  object Edit9: TEdit
    Left = 310
    Top = 215
    Width = 199
    Height = 22
    TabOrder = 17
    Text = '0,01'
  end
  object kzTime: TEdit
    Left = 310
    Top = 241
    Width = 199
    Height = 22
    TabOrder = 18
    Text = '200'
  end
  object BitBtn1: TBitBtn
    Left = 362
    Top = 378
```

```
  Width = 81
  Height = 26
  TabOrder = 19
  OnClick = BitBtn1Click
  Kind = bkOK
end
object StaticText9: TStaticText
  Left = 26
  Top = 274
  Width = 268
  Height = 18
  AutoSize = False
  BorderStyle = sbsSunken
  Caption = 'Шаг по времени'
  TabOrder = 20
end
object StaticText12: TStaticText
  Left = 26
  Top = 303
  Width = 268
  Height = 18
  AutoSize = False
  BorderStyle = sbsSunken
  Caption = 'Коэффициент обратной связи'
  TabOrder = 21
end
object StepTime: TEdit
  Left = 308
  Top = 269
  Width = 199
  Height = 22
  TabOrder = 22
  Text = '1'
end
object Koefz: TEdit
  Left = 308
  Top = 299
  Width = 199
  Height = 22
  TabOrder = 23
  Text = '0'
end
object StaticText13: TStaticText
```

```
    Left = 26
    Top = 335
    Width = 268
    Height = 18
    AutoSize = False
    BorderStyle = sbsSunken
    Caption = 'Порядок отображаемой производной'
    TabOrder = 24
  end
  object Edit8: TEdit
    Left = 308
    Top = 331
    Width = 199
    Height = 22
    TabOrder = 25
    Text = '1'
  end
end
```

13.2.4. Модуль ввода данных

```
//INPUTD.PAS


unit InputD;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, Buttons,T_Vector,math;
type
  TForm2 = class(TForm)
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    StaticText3: TStaticText;
    StaticText4: TStaticText;
    StaticText5: TStaticText;
    StaticText6: TStaticText;
    StaticText7: TStaticText;
    StaticText8: TStaticText;
    StaticText10: TStaticText;
```

```
    StaticText11: TStaticText;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit6: TEdit;
    Edit7: TEdit;
    Edit9: TEdit;
    kzTime: TEdit;
    BitBtn1: TBitBtn;
    Bevel1: TBevel;
    StaticText9: TStaticText;
    StaticText12: TStaticText;
    StepTime: TEdit;
    Koefz: TEdit;
    StaticText13: TStaticText;
    Edit8: TEdit;
    procedure BitBtn1Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;
  Procedure DelSpace (var s:string); Far;
var
  Form2: TForm2;
  vrng,vsc,vrngr,errc,vDerNum:integer;
  vcfn,vcfnr,vnusl:TVector;
  vfreq,vcufz,vsexp:real;
  vKos,vTend,vStepTime:real;
implementation

{$R *.DFM}

Procedure DelSpace (var s:string);
var i:integer;
begin
i:=1;
repeat
if s[i] = ' ' then begin delete(s,i,1);
i:=i-1;
```

```
end;
i:=i+1;
until i>length(s);
end;

procedure TForm2.BitBtn1Click(Sender: TObject);
var i,j,k:integer;
    temp:string;

begin
errc := 0;
temp:=edit1.Text;
DelSpace(temp);
edit1.Text:=temp;

temp:=edit4.Text;
DelSpace(temp);
edit4.Text:=temp;

temp:=StepTime.Text;
DelSpace(temp);
StepTime.Text:=temp;

temp:=kzTime.Text;
DelSpace(temp);
kzTime.Text:=temp;

temp:=Koefz.Text;
DelSpace(temp);
Koefz.Text:=temp;

temp:=edit8.Text;
DelSpace(temp);
edit8.Text:=temp;


vrng:=strtoint(edit1.Text);
vrngr:=strtoint(edit4.Text);
vTend:=strtoint(kzTime.Text);
vStepTime:=strtoint(StepTime.Text);
vKos:=strtoint(Koefz.text);
vDerNum:=strtoint(edit8.text);
vsc:=floor(vTend/vSteptime);
```

```
vcfn:=TVector.VCreate(vrng+1);
vcfnr:=TVector.VCreate(vrngr+1);
vnusl:=TVector.VCreate(vrng);

if vsc<=50 then begin
   MessageBox('Мало шагов решения');
   inc(errc);
   exit;
   end;

if vrng<=0 then begin
   MessageBox('Нулевой порядок уравнения');
   inc(errc);exit;
   end;

if vrngr<0  then begin
   MessageBox('Некорректен код порядка справа');
   inc(errc);exit;
   end;

if vTend<=0 then begin
   MessageBox('Ошибочно конечное значение аргумен-
та');
   inc(errc);exit;
   end;

if vStepTime<=0 then begin
   MessageBox('Ошибочен шаг по аргументу');
   inc(errc);exit;
   end;

if vrngr>vrng then begin
   MessageBox('Порядок справа больше чем слева');
   inc(errc);exit;
   end;

if (vDerNum<0) or (vDerNum>(vrng-1)) then begin
   MessageBox('Не корректен порядок производной');
   inc(errc);exit;
   end;

j:=1;
edit2.Text:=edit2.Text+' '+#33+' ';
```

```
i:=0;
while copy(edit2.Text,j,1) = ' ' do j:=j+1;
repeat
k:=j;
while copy(edit2.Text,j,1)<>' ' do j:=j+1;
if copy(edit2.Text,k,j-k) = #33 then break;
vcfn.Vector[i]:=strtofloat(copy(edit2.Text,k,j-k));
while copy(edit2.Text,j,1) = ' ' do j:=j+1;
inc(i);
until false;
temp:=edit2.text;
delete (temp,j-1,length(temp)-j+2);
edit2.text:=temp;
if ((i-1) <> vrng) or (vcfn.Vector[0]=0) then begin
   MessageBox('Кол-во коэффициентов не соответству-
ет '+#13+'порядку уравнения слева');
   inc(errc);exit;
   end;
if vrngr >= 0 then begin
j:=1;
edit5.Text:=edit5.Text+' '+#33+' ';
i:=0;
while copy(edit2.Text,j,1) = ' ' do j:=j+1;
repeat
k:=j;
while copy(edit5.Text,j,1)<>' ' do j:=j+1;
if copy(edit5.Text,k,j-k) = #33 then break;
vcfnr.Vector[i]:=strtofloat(copy(edit5.Text,k,j-
k));
while copy(edit5.Text,j,1) = ' ' do j:=j+1;
inc(i);
until false;
temp:=edit5.text;
delete (temp,j-1,length(temp)-j+2);
edit5.text:=temp;
if ((i-1) <> vrngr) or (vcfnr.Vector[0]=0) then
begin
   MessageBox('Кол-во коэффициентов не соответству-
ет '+#13+'порядку уравнения справа');
   inc(errc);exit;
   end;
end;
```

```
j:=1;
edit3.Text:=edit3.Text+' '+#33+' ';
i:=0;
while copy(edit2.Text,j,1) = ' ' do j:=j+1;
repeat
k:=j;
while copy(edit3.Text,j,1)<>' ' do j:=j+1;
if copy(edit3.Text,k,j-k) = #33 then break;
vnusl.Vector[i]:=strtofloat(copy(edit3.Text,k,j-
k));
while copy(edit3.Text,j,1) = ' ' do j:=j+1;
inc(i);
until i=vrng;
temp:=edit3.text;
delete (temp,j-1,length(temp)-j+2);
edit3.text:=temp;
if i <> vrng then begin
   MessageBox('Количество начальных условий не со-
ответствует '+#13+'порядку уравнения слева');
   inc(errc);exit;
   end;

temp:=edit6.Text;
DelSpace(temp);
edit6.Text:=temp;

temp:=edit7.Text;
DelSpace(temp);
edit7.Text:=temp;

temp:=edit9.Text;
DelSpace(temp);
edit9.Text:=temp;

vfreq:=strtofloat(edit6.Text);
vsexp:=strtofloat(edit7.Text);
vcufz:=strtofloat(edit9.Text);

if errc = 0 then form2.close;
end;

end.
```

13.2.5. Форма основной программы

```
//MAIN.DFM


object Form1: TForm1
  Left = 29
  Top = 112
  Width = 726
  Height = 478
  Caption = 'Численное решение ОЛДУ'
  Color = clBtnFace
  Font.Charset = RUSSIAN_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Fixedsys'
  Font.Style = [fsBold]
  Menu = MainMenu1
  OldCreateOrder = False
  Position = poScreenCenter
  Visible = True
  OnClose = FormClose
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 16
  object Bevel1: TBevel
    Left = 0
    Top = 0
    Width = 718
    Height = 2
    Align = alTop
    ParentShowHint = False
    Shape = bsTopLine
    ShowHint = False
  end
  object Image1: TImage
    Left = 0
    Top = 2
    Width = 718
    Height = 430
    Align = alClient
    AutoSize = True
    Stretch = True
```

```
  end
  object MainMenu1: TMainMenu
    Left = 112
    Top = 80
    object inputD: TMenuItem
      Caption = 'Ввод данных'
      OnClick = inputDClick
    end
    object N1: TMenuItem
      Caption = 'Решение уравнения'
      object mLiberty: TMenuItem
        Caption = 'Свободное движение'
        OnClick = mLibertyClick
      end
      object mImpuls: TMenuItem
        Caption = 'Реакция на импульс'
        OnClick = mImpulsClick
      end
      object mStupen: TMenuItem
        Caption = 'Реакция на ступенчатое воздей-
ствие'
        OnClick = mStupenClick
      end
      object msinus: TMenuItem
        Caption = 'Реакция на синусоиду'
        OnClick = msinusClick
      end
      object mcosinus: TMenuItem
        Caption = 'Реакция на косинусоиду'
        OnClick = mcosinusClick
      end
      object exponenta: TMenuItem
        Caption = 'Реакция на экспоненту'
        OnClick = exponentaClick
      end
      object proizvoln: TMenuItem
        Caption = 'Реакция на произвольное воздей-
ствие'
        OnClick = proizvolnClick
      end
    end
    object N9: TMenuItem
      Caption = 'Частотные характеристики'
```

```
    object Rchar: TMenuItem
      Caption = 'Вещественная'
      OnClick = RcharClick
    end
    object ImChar: TMenuItem
      Caption = 'Мнимая'
      OnClick = ImCharClick
    end
    object ampChar: TMenuItem
      Caption = 'Амплитудная'
      OnClick = ampCharClick
    end
    object freqchar: TMenuItem
      Caption = 'Фазовая'
      OnClick = freqcharClick
    end
    object amphfaze: TMenuItem
      Caption = 'Амплитудно-фазовая'
      OnClick = amphfazeClick
    end
  end
  object mportret: TMenuItem
    Caption = 'Фазовые портреты'
    object plib: TMenuItem
      Caption = 'При свободном движении'
      OnClick = plibClick
    end
    object pimpuls: TMenuItem
      Caption = 'При импульсе на входе'
      OnClick = pimpulsClick
    end
    object pstup: TMenuItem
      Caption = 'При ступеньке на входе'
      OnClick = pstupClick
    end
    object psin: TMenuItem
      Caption = 'При синусоиде на входе'
      OnClick = psinClick
    end
    object pcos: TMenuItem
      Caption = 'При косинусоиде на входе'
      OnClick = pcosClick
    end
```

```
      object pexp: TMenuItem
        Caption = 'При экспоненте на входе'
        OnClick = pexpClick
      end
      object puser: TMenuItem
        Caption = 'При произвольной функции'
        OnClick = puserClick
      end
    end
    object Clear: TMenuItem
      Caption = 'Очистка области вывода'
      OnClick = ClearClick
    end
    object Help1: TMenuItem
      Caption = 'Помощь'
      object Contents1: TMenuItem
        Caption = 'Содержание'
        OnClick = Contents1Click
      end
      object About1: TMenuItem
        Caption = 'Об авторах...'
        OnClick = About1Click
      end
    end
    object N2: TMenuItem
      Caption = 'Опции..'
      object mRead: TMenuItem
        Caption = 'Чтение из файла'
        OnClick = mReadClick
      end
      object mSave: TMenuItem
        Caption = 'Запись результатов'
      end
      object Pen1: TMenuItem
        Caption = 'Карандаш'
        object Pen: TMenuItem
          Caption = 'Толшина'
          OnClick = PenClick
        end
      end
    end
  end
  object OpenDialog1: TOpenDialog
```

```
    Filter = '*.txt|*.txt'
    Options = [ofHideReadOnly, ofPathMustExist,
ofFileMustExist, ofCreatePrompt, ofEnableSizing]
    Left = 264
    Top = 88
  end
end
```

13.2.6. Модуль основной программы

```
//MAIN.PAS


unit main;

interface

uses
  {Подключаем стандартные модули Delphi 4}
  Windows, Messages, SysUtils,
  Classes, Graphics, Controls, Forms, Dialogs,
ComCtrls, ExtCtrls,Menus,math,

  {Подключаем модули для работы с мат-ми объектами
и ввод данных}

About,T_Vector,T_CVector,PenW,T_Complex,T_Polynom,T
_Matrix,T_CMatrix,InputD;

type

 {Сведения о корнях характеристического полинома
сгруппируем в структуре, а для всех корней придется
разместить в памяти массив таких структур}

    Ca=record
       root,              //значение корня
       a:TComplex; {коэффициент для корня в ориги-
нале}
       o,               //кому кратен
       j:integer;       //кратность
       end;
```

```
   R_type = array of ca;
{Класс обыкновенных линейных дифференциальных урав-
нений (ОЛДУ), содержащий методы работы, базирующие-
ся на операционном исчислении.}

  TDifferentialEquation = class(TObject)

  rngl,rngr:integer;
  nv:TVector;
 coeff_count,{Количество коэффициентов в уравнении}
  cod,
  rootcount,           //Количество корней всего
  rcount:integer;   //Количество различных корней
  Omega,Alpha,Kos,
  TStep,Tend:real;
  PointSolCnt:integer;
  pr,pn,pl,Psol,Qsol:TPolynom;
  r: R_Type;

  private

  protected

  public
  Constructor DECreate(CFNL,CFNR,NU:TVector;
COD:integer; OMEGA,ALPHA,TEND,TSTEP,KOS:real);

 {Функция, вычисляющая коэффициенты оригинала для
всех корней результат помещается в массив структур
R}
  procedure GetCoeffOrigin;virtual;

  {Функция для вычисления корней характеристическо-
го полинома }
  procedure GetRoot;virtual;

  Function QSoltn (der:integer) :TPolynom ;
Virtual;

  {Функция, возвращающая значение выхода при задан-
ном значении аргумента }
  function GetValue(dt:real):extended;virtual;
```

```
  Procedure Sol; Virtual;
  Procedure FreqChar; Virtual;

  published
  end;

  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    inputD: TMenuItem;
    N1: TMenuItem;
    mLiberty: TMenuItem;
    mImpuls: TMenuItem;
    mStupen: TMenuItem;
    msinus: TMenuItem;
    mcosinus: TMenuItem;
    exponenta: TMenuItem;
    proizvoln: TMenuItem;
    N9: TMenuItem;
    ampChar: TMenuItem;
    freqchar: TMenuItem;
    amphfaze: TMenuItem;
    mportret: TMenuItem;
    Clear: TMenuItem;
    Bevel1: TBevel;
    Help1: TMenuItem;
    About1: TMenuItem;
    Contents1: TMenuItem;
    N2: TMenuItem;
    mRead: TMenuItem;
    mSave: TMenuItem;
    OpenDialog1: TOpenDialog;
    Pen1: TMenuItem;
    Pen: TMenuItem;
    Image1: TImage;
    Rchar: TMenuItem;
    ImChar: TMenuItem;
    plib: TMenuItem;
    pimpuls: TMenuItem;
    pstup: TMenuItem;
    psin: TMenuItem;
    pcos: TMenuItem;
    pexp: TMenuItem;
    puser: TMenuItem;
```

```
            {Реакции на нажатия элементов меню}
    procedure inputDClick(Sender: TObject);
    procedure About1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var
Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure mStupenClick(Sender: TObject);
    procedure exponentaClick(Sender: TObject);
    procedure ClearClick(Sender: TObject);
    procedure mLibertyClick(Sender: TObject);
    procedure mImpulsClick(Sender: TObject);
    procedure msinusClick(Sender: TObject);
    procedure mcosinusClick(Sender: TObject);
    procedure proizvolnClick(Sender: TObject);
    procedure freqcharClick(Sender: TObject);
    procedure amphfazeClick(Sender: TObject);
    procedure Contents1Click(Sender: TObject);
    procedure mReadClick(Sender: TObject);
    procedure mSaveClick(Sender: TObject);
    procedure PenClick(Sender: TObject);
    procedure RcharClick(Sender: TObject);
    procedure ImCharClick(Sender: TObject);
    procedure ampCharClick(Sender: TObject);
    procedure plibClick(Sender: TObject);
    procedure pimpulsClick(Sender: TObject);
    procedure pstupClick(Sender: TObject);
    procedure psinClick(Sender: TObject);
    procedure pexpClick(Sender: TObject);
    procedure pcosClick(Sender: TObject);
    procedure puserClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

{Для графического отображения функций решения опре-
делим массив структур типа TPOINT, в который потом
перенесем пересчитанные в координаты значения отоб-
ражаемой функции и значения ее аргумента }
  ptt = array of TPoint;
```

```
 {Нам понадобятся простые служебные функции - вы-
числения факториала и определения знака числа}
  function Fact(x:integer):longint;far;
  function Sign(x:real):integer; far;


 {Для рисования кривых нам понадобится массив
структур типа POINT с вертикальной и горизонтальной
координатами точек кривой. В нашем же распоряжении
пока есть только вещественный массив значений отоб-
ражаемой функции (их нужно преобразовать в ордина-
ты) и подразумеваемый массив последовательности це-
лых 0,1,2,...,sc-1, которые надо пересчитать в абс-
циссы рисуемых точек. Для такого пересчета напишем
подпрограмму GetCoord . Прежде всего подпрограмма
должна вычислить диапазон изменения значений функ-
ции и количество пикселов на одну натуральную еди-
ницу по обеим осям.}
  function GetCoord(const y,x:TVector):ptt;far;

  {Оптимизация структуры R}
  Procedure R_Optimize(var x:R_Type); Far;
  Procedure ProcessPaint (const fnc:integer); Far;
  Procedure PaintFreqChar (const cod:integer); Far;
  Procedure PaintPortret (const cod:integer); Far;
const
cod : integer = 0;
codl : integer =0;
var
  isData:boolean;//Флаг-признак ввода данных
  W, //Частота среза
  SolMtr,FreqChMtr:TMatrix;{Матрицы решения и ча-
стотных характеристик }
  fxmax,fxmin,fymax,fymin:extended; {MAX и MIN зна-
чения x и y }
  sc:integer;{Для количества дискретных точек реше-
ния}
  pt:ptt;
  Form1: TForm1;
  DeriveNumber:integer;{Порядок отображаемой произ-
водной}
  rngl,rngr:integer;{Для порядка уравнения в левой
и правой части}
```

```
  cufz, {Для кратности снижения амплитуды на часто-
те среза}
  alpha,omega:real;  {Для частоты гармонических
возмущений и степени экспоненциального}
  tc:TVector; {Массив для значений аргументов функ-
ций}
  cfnl,cfnr,nu:TVector;

{Для задания произвольной входной функции тоже от-
ведем вещественный массив значений - заполнить его
можно например из файла. Результаты решения тоже
удобнее всего для последующего анализа сохранять в
дисковых файлах}
  Userfunc:TVector;

  userfile:TFileName;      //Имя файла пользователя
  de:TDifferentialEquation;
  coords:TPoint;        //Координаты осей графика
  penwid:integer;          //Толщина карандаша
  Tstep,TEnd,Kos:real;{Конечное время, шаг по вре-
мени и коэффициент обратной связи}

  implementation

{$R *.DFM}
var
 abc,ordn:array [1..2] of TPoint; //Оси


{Переприсваивание переменных из модуля ввода
данных}
procedure TForm1.inputDClick(Sender: TObject);
label 1;
begin
isData:=false;
1 : form2.ShowModal;
if errc <> 0 then goto 1;
isData:=true;
rngl:=vrng;
rngr:=vrngr;
cfnl:=vcfn;
cfnr:=vcfnr;
omega:=vfreq;
```

```
alpha:=vsexp;
cufz:=vcufz;
nu:=vnusl;
Tstep:=vStepTime;
TEnd:=vTend;
Kos:=VKos;
DeriveNumber:=vDerNum;
sc:=Floor(TEnd/TStep);
end;

Constructor
TDifferentialEquation.DECreate(CFNL,CFNR,NU:TVector
;COD:integer;OMEGA,ALPHA,TEND,TSTEP,KOS:real);
var i,j:integer;
    Plt,Prt,Qsolt,Psolt:TPolynom;
    UOS,p,d,tpn:Tpolynom;
    rone:TPolynom;
    jstep:real;
begin
inherited Create;
QSol:=TPolynom.PCreate(0);
PSol:=TPolynom.PCreate(0);
rngl:=CFNL.len-1;
rngr:=CFNR.len-1;
pn:=TPolynom.PCreate(0);
Plt:=TPolynom.PCreate(rngl+1);
Prt:=TPolynom.PCreate(rngr+1);
QSolt:=TPolynom.PCreate(0);
PSolt:=TPolynom.PCreate(0);
for i:=0 to rngr do
Prt.Vector[i]:=Complex(cfnr.vector[i],0);
Prt.Revers;
Pr:=Prt;
QSol:=Prt;
for i:=0 to rngl do
Plt.Vector[i]:=Complex(cfnl.vector[i],0);
Plt.Revers;
UOS:=TPolynom.PCreate(1);
UOS.Vector[0]:=Complex(Kos,0);
if Kos<>0 then Plt:=PSumma(Plt,PMult(UOS,Prt));
Pl:=Plt;
PSol:=Plt;
nv:=TVector.VCreate(rngl);
```

```
for i:=0 to rngl-1 do nv.vector[i]:=NU.Vector[rngl-
i-1];
p:=TPolynom.PCreate(2);
d:=TPolynom.PCreate(0);
d:=pl;
tpn:=TPolynom.PCreate(0);
p.Vector[0]:=Complex(0,0);
p.Vector[1]:=Complex(1,0);
for i:=0 to rngl-1 do begin
d:=PIntOfDiv(d,p);
  tpn:=D.PMultOnDig(complex(nv.vector[i],0));
  pn:=PSumma(pn,tpn);
end;
{Сформируем числитель QSOL и знаменатель PSOL изоб-
ражения решения - это полиномиальная дробь}
rone:=TPolynom.PCreate(1);
rone.vector[0]:=complex(1.0,0.0);{веществ. полино-
миальная единица}
case cod of
 0:begin QSol:=pn;PSol:=Pl; end;
 1,6:begin Qsol:=PSumma(pr,pn);PSol:=Pl; end;
 2:begin QSol:=PSumma(PMult(pn,p),pr);
PSol:=PMult(pl,p); end;
 3:begin

QSol:=PSumma(PMult(pn,PSumma(p.PPow(2),rone.PMultOn
Dig(Complex(omega,0)).PPow(2))),
PMult(pr,rone.PMultOnDig(Complex(omega,0))));

PSol:=PMult(pl,PSumma(p.PPow(2),rone.PMultOnDig(Com
plex(omega,0)).PPow(2)));
   end;
 4:begin

QSol:=PSumma(PMult(pn,PSumma(p.PPow(2),rone.PMultOn
Dig(Complex(omega,0)).PPow(2))), PMult(pr,p));

PSol:=PMult(pl,PSumma(p.PPow(2),rone.PMultOnDig(Com
plex(omega,0)).PPow(2)));
   end;
 5:begin
```

```
QSol:=PSumma(PMult(pn,PSumma(p,rone.PMultOnDig(Comp
lex(alpha,0)))),pr);

PSol:=PMult(pl,PSumma(p,rone.PMultOnDig(Complex(alp
ha,0))));
   end;
 7: exit;
end;
psol.optimize;
PointSolCnt:=Floor(TEnd/TStep);
RootCount:=pSol.len-1;
setlength(r,rootcount);
tc:=TVector.VCreate(pointSolCnt);
jstep:=0;
for i:=0 to PointSolCnt-1 do begin
tc.vector[i]:=jstep;
jstep:=jstep+tstep;
end;
SolMtr:=Tmatrix.MCreate(RootCount+1,PointSolCnt);
if RootCount>0 then
  for i:=0 to rngl-1 do
SolMtr.Matrix[i,0]:=nv.vector[i];
GetRoot;
end;

function GetCoord(const y,x:TVector):ptt;
var i:integer;
    res:ptt;
    x1,y1:TVector; { Создаём копии чтобы не изме-
нять оригинал}
begin
x1:=TVector.VCreate(x.len);
y1:=TVector.VCreate(y.len);
for i:=0 to sc-1 do begin { Копируем данные}
x1.Vector[i]:=x.Vector[i];
y1.Vector[i]:=y.Vector[i];
end;
{Определим наибольшее, наименьшее значение функции,
диапазон  изменения}
fymax:=y1.Vector[0];
fymin:=fymax;
fxmax:=x1.vector[0];
```

```
fxmin:=fxmax;
for i :=0  to sc-1  do
begin
if y1.vector[i]>fymax then fymax:=y1.vector[i];
if y1.vector[i]<fymin then fymin:=y1.vector[i];
if x1.vector[i]>fxmax then fxmax:=x1.vector[i];
if x1.vector[i]<fxmin then fxmin:=x1.vector[i];
end;
for i:=0 to sc-1 do begin  {Отцентрируем все данные
относительно минимумов}
x1.Vector[i]:=x1.vector[i]-fxmin;
y1.Vector[i]:=y1.vector[i]-fymin;
end;
setlength(res,sc);
if (fxmax<>fxmin) and (fymax<>fymin) then
begin //Теперь заполним массив структур res
for i:=0 to sc-1 do
res[i].y:=round(form1.image1.clientheight-2-
y1.vector[i]*(form1.image1.clientheight-2)/(fymax-
fymin));
for i:=0 to sc-1 do
res[i].x:=round(x1.vector[i]*(form1.image1.clientwi
dth-2)/(fxmax-fxmin));
result:=res;
if fxmax<>fxmin then
coords.x:=abs(round(fxmin*(form1.image1.clientwidth
-2)/(fxmax-fxmin)));
if fymin<>fymax then
coords.y:=Abs(round(fymin*(form1.image1.clientheigh
t-2)/(fymax-fymin)));
//Ось ординат
abc[1].x:=coords.x+1;
abc[1].y:=0;
abc[2].x:=coords.x+1;
abc[2].y:=form1.image1.ClientHeight-2;
// Ось абсцисс
ordn[1].x:=0;
ordn[1].y:=form1.image1.clientheight-coords.y-2;
ordn[2].x:=form1.image1.clientWidth;
ordn[2].y:=form1.image1.clientheight-coords.y-2;
end;
x1.Free; // Убираем мусор
y1.Free;
```

```
end;

procedure R_Optimize(var x:R_Type);
var i:integer;
begin
for i:=low(x) to high(x) do begin  {Ограничим зна-
чения полей записи}
if abs(x[i].a.re)<=1e-7 then x[i].a.re:=0;
if abs(x[i].a.im)<=1e-7 then x[i].a.im:=0;
if abs(x[i].root.re)<=1e-7 then x[i].root.re:=0;
if abs(x[i].root.im)<=1e-7 then x[i].root.im:=0;
end;
end;

procedure TForm1.About1Click(Sender: TObject);
begin
form3.ShowModal;
end;

function Fact(x:integer):longint; {Вычисление фак-
ториала числа}
var res,i:longint;
begin
 res:=1;
 for i:=1 to x do res:=res*i;
 result:=res;
end;

function Sign(x:real):integer; {Знак числа}
var res:integer;
begin
res:=0;
if x<>0 then res:=round(abs(x)/x);
result:=res;
end;

{Функция для вычисления корней характеристического
полинома}
procedure TDifferentialEquation.GetRoot;
var i,j,rep:integer;
    Polyroot:TCVector;
begin
polyroot:=TCVector.CVCreate(0);
```

```
polyroot.Vector:=newton(PSol); {ищем корни характе-
ристического полинома}
Polyroot.vector:=CVOptimize(Polyroot.vector);
for i:=0 to rootcount-1 do
r[i].root:=polyroot.vector[i];{Заносим в структуру
значения корней}
{определяем кратность каждого корня и заносим в
структуру}
 rcount:=rootcount; {Вначале предполагаем, что все
корни различны}
for i:=0 to rootcount-1 do begin
 rep:=1;
 if r[i].j<>-1 then begin//корень еще не встречался
  r[i].j:=1;    //количество повторений i-го корня
  for j:=i+1 to rootcount-1 do
  if (r[j].j<>1) and Cequiv(r[i].root,r[j].root)
then begin
    inc(rep);
    dec(rcount);
    r[j].j:=-1;{устанавливаем признак того, что
этот корень уже учтён}
    r[j].o:=i;
  end;
  r[i].j:=rep; //кратность корня
 end;
end;
//R_Optimize(r);
end;

{Подпрограмма определения числителя изображения
производной решения}
function TDifferentialEquation.QSoltn(der:integer):
TPolynom;
var q,p,sum:TPolynom;
    i:integer;
begin
q:=TPolynom.PCreate(rootcount);
p:=TPolynom.PCreate(2);
p.vector[0]:=Complex(0,0);
p.vector[1]:=Complex(1,0);
sum:=TPolynom.PCreate(0);
for i:=0 to Qsol.len-1 do
q.Vector[i]:=QSol.vector[i];
```

```
if der > 0  then begin
for i:=0 to der-1 do
   sum:=PSumma(sum,p.PPow(der-i-
1).PmultOnDig(q.vector[rootcount-i-1]));
q:=PRazn(PMult(q,p.ppow(der)),PMult(PSol,sum));
end;
result:=q;
end;


{Функция, вычисляющая коэффициенты оригинала для
всех корней, результат помещается в массив структур
R}
procedure TDifferentialEquation.GetCoeffOrigin;
var i,j,k,l,rd,pi,qi:integer;
    ap,tp,zz:TComplex;
    pw,pw1,ch,zn,rch,rzn,a:TPolynom;
    chisl,znamen,dchisl,dznamen,m1,m2,tmp:TPolynom;
begin
qi:=QSol.len; pi:=PSol.len;
if qi = 1 then ap:=QSol.vector[0];
if pi = 1 then tp:=PSol.vector[0];
if rootcount = 1 then begin
 if pi > 1 then
tp:=PSol.Derive(1).GetFun(r[0].root);
 if qi > 1 then ap:=QSol.GetFun(r[0].root);
 r[0].a:=CDiv(ap,tp);
 exit;
end;

pw:=TPOLynom.PCreate(2);
ch:=TPOLynom.PCreate(0);
zn:=TPOLynom.PCreate(0);
rch:=TPOLynom.PCreate(0);
rzn:=TPOLynom.PCreate(0);
ch:=QSol; zn:=PSol;
a:=TPOLynom.PCreate(0);

for k:=0 to rootcount-1 do begin
 if r[k].j=1 then begin
  pw.Vector[0]:=CMultOnDig(r[k].root,-1);
  pw.Vector[1]:=complex(1,0);
  if qi > 1 then ap:=QSol.GetFun(r[k].root);
```

```
  if pi > 1 then
tp:=PIntOfDiv(PSol,pw).GetFun(r[k].root);
  r[k].a:=Cdiv(ap,tp);
  end;
 if r[k].j>1 then begin
   pw1:=TPolynom.PCreate(0);
   pw.Vector[0]:=CMultOnDig(r[k].root,-1);
   pw.Vector[1]:=complex(1,0);
   pw1:=pw.PPow(r[k].j);
   j:=1;
   while j <= r[k].j do begin
    if j = 1 then begin
      if qi > 1 then ap:=QSol.GetFun(r[k].root);
      if pi > 1 then
tp:=PIntOfDiv(PSol,pw1).GetFun(r[k].root);
      r[k].a:=CDiv(ap,tp);
      inc(j);
    end
    else
    for i:=k+1 to rootcount-1 do begin
     if r[i].o = k then begin
      tmp:=TPolynom.Pcreate(0);
      chisl:=TPolynom.Pcreate(0);
      znamen:=TPolynom.Pcreate(0);
      dchisl:=TPolynom.Pcreate(0);
      dznamen:=TPolynom.Pcreate(0);
      m1:=TPolynom.Pcreate(0);
      m2:=TPolynom.Pcreate(0);
      tmp:=PIntOfDiv(PSol,pw1);
      chisl:=QSol;
      znamen:=PintOfDiv(PSol,pw1);
      for l:=1 to j-1 do begin
       dchisl:=chisl.Derive(1);
       dznamen:=znamen.Derive(1);
       m1:=PMult(dchisl,znamen);
       m2:=PMult(dznamen,chisl);
       chisl:=PRazn(m1,m2);
       znamen:=znamen.ppow(2);
      end;

r[i].a:=CDIv(chisl.getfun(r[k].root),CMultOnDig(zna
men.getfun(r[k].root),fact(j-1)));
      inc(j);
```

```
    end;
   end;
  end;
 end;
end;
end;

{Функция, возвращающая значение решения ОЛДУ при
заданном t}
function
TDifferentialEquation.GetValue(dt:real):extended;
var i,index,k:integer;
    res,t:TComplex;
    dpw,d:extended;
begin
res:=Complex(0,0);
t:=Complex(dt,0);
if rootcount > 0 then begin
for k:=0 to rcount-1 do begin
 index:=0;
 if r[k].j=1 then //Если корень простой

res:=CSumma(res,CMult(r[k].a,Cexp(Cmult(r[k].root,t
))));
 if r[k].j>1 then begin //Если корень кратный

res:=Csumma(res,CMult(r[k].a,Cexp(Cmult(r[k].root,t
))));
    inc(index);
    for i:=k+1 to rcount-1 do begin
        if (r[i].o=k) and (r[i].j=-1) then begin
           dpw:=r[k].j-index;

res:=CSumma(res,CMultOnDig(CMult(r[k].a,CMult(Cpow(
t,dpw),CExp(CMult(r[k].root,t)))),1/fact(r[k].j-
index)));
           inc(index);
        end;
    end;
 end;
end;
d:=res.re;
result:=d;
```

```
exit;
end;
result:=0;
end;

procedure TDifferentialEquation.Sol;
var i,k:integer;
    q:TPolynom;
begin
q:=TPolynom.PCreate(0);
q:=QSol;
{if (cod=0) and (q.len=1) then begin
 for i:=0 to SolMtr.Row-1  do
  for k:=0 to SolMtr.Col-1 do
solMtr.matrix[i,k]:=0;
 exit;
end; }
if cod = 6 then begin
 GetCoeffOrigin;
 For k:=1 to PointSolCnt-1 do
SolMtr.Matrix[0,k]:=GetValue(tc.Vector[k]);
 for i:=1 to PointSolCnt-1 do begin
   SolMtr.Matrix[1,i]:=0;
   for k:=0 to i-1 do
SolMtr.Matrix[1,i]:=SolMtr.Matrix[1,i]+SolMtr.matri
x[0,i-k]*UserFunc.Vector[i]*TStep;
 end;
end
 else
for i:=0 to rootcount-1 do begin
 QSol:=QSoltn(i);
 GetCoeffOrigin;
 for k:=1 to PointSolCnt-1 do
SolMtr.Matrix[i,k]:=GetValue(tc.vector[k]);
 QSol:=q;
end;
//Вывод матрицы в поток
end;

procedure TDifferentialEquation.FreqChar;
var
    jone:TComplex;
    i:integer;
```

```
    w,kuz,WStep,QW,PW,KU,KUO,RQ,IQ,RP,IP:real;
begin
   jone:=complex(0.0,1.0);// мнимая единица
   KUO:=abs(CDiv(pr.Vector[0],pl.vector[0]).re);
//Усиление на нулевой частоте
   KUZ:=Cufz*KUO;
   wstep:=1.0;
   ku:=kuo;
   if  pr.len=1 then QW:= pr.Vector[0].re;
   //Подбираем частоту среза
    w:=wstep;
   repeat
   if  pr.Len>1 then
    QW:=CNorm(pr.GetFun(CMultOnDig(jone,w)));
    pW:=CNorm(pl.GetFun(CMultOnDig(jone,w)));
    KU:=QW/pW;    //Усиление на текущей частоте
    if (KU < 1.1*KUZ) and (KU > 0.9*KUZ) then
break;
    if (KU < 0.9*KUZ) And (wstep > 0) then
wstep:=-0.1*wstep;
    if (KU > 1.1*KUZ) And (wstep < 0) then
wstep:=-0.1*wstep;
    w:=w+wstep;
    until False;
{Заполним вектор частоты в матрице - мы разместим
его в 0-й строке}
    wstep:=w/PointSolCnt;
    FreqChMtr:=TMatrix.MCreate(5,PointSolCnt);
    for i:=0 to PointSolCnt-1 do
FreqChMtr.Matrix[0,i]:=i*wstep;
    if pr.len = 1 then begin
RQ:=PR.vector[0].re;IQ:=0; end;
    for i:=0 to PointSolCnt-1 do begin
       if pr.len > 1 then begin
RQ:=pr.GetFun(CMultOnDig(jone,FreqChMtr.Matrix[0,i]
)).re;
IQ:=pr.GetFun(CMultOnDig(jone,FreqChMtr.Matrix[0,i]
)).im;
       end;
RP:=pl.GetFun(CMultOnDig(jone,FreqChMtr.Matrix[0,i]
)).re;
IP:=pl.GetFun(CMultOnDig(jone,FreqChMtr.Matrix[0,i]
)).im;
```

```
FreqChMtr.Matrix[1,i]:=(RQ*RP+IQ*IP)/(RP*RP+IP*IP);
       FreqChMtr.Matrix[2,i]:=(IQ*RP-
RQ*IP)/(RP*RP+IP*IP);
FreqChMtr.Matrix[3,i]:=Sqrt(FreqChMtr.Matrix[1,i]*F
reqChMtr.Matrix[1,i]+SQR(FreqChMtr.Matrix[2,i]));
FreqChMtr.Matrix[4,i]:=ArcTan2(FreqChMtr.Matrix[2,i
],FreqChMtr.Matrix[1,i]);
    end;
// Вывод матрицы.
end;

procedure ProcessPaint(const fnc:integer);
var temp:TVector;
begin
if not isData then begin MessageBox('Не введены
данные ');exit;end;
de:=TDifferentialEquation.DECreate(cfnl,cfnr,nu,
fnc,omega,alpha,tend,tstep,kos);
de.Sol;
randomize;
temp:=TVector.VCreate(sc);
temp.vector:=Solmtr.matrix[1];
with Form1.Image1.Canvas do begin
 Pen.Color:=RGB(random(255),random(255),
random(255));
 Font.Color:=Pen.Color;
 TextOut(form1.Image1.Width div 2-40,10,'-------
график функции');
 if fnc = 6 then pt:=GetCoord(temp,tc)
  else begin
  temp.vector:=Solmtr.matrix[0];
  pt:=GetCoord(temp,tc);
  end;
 Polyline(pt);
 Pen.Color:=RGB(random(255),random(255),
random(255));
 if (fnc <> 6) and (DeriveNumber <>0) then begin
 temp.vector:=Solmtr.matrix[deriveNumber];
 pt:=GetCoord(temp,tc);
 Pen.Color:=0;
 Polyline(pt);
 Font.Color:=0;
```

```
 TextOut(form1.Image1.Width div 2-40,30,'-------
производная');
 end;
end;
end;

procedure PaintFreqChar(const cod:integer);
var t1,t2:TVector;
begin
if not isData then begin MessageBox('Не введены
данные ');exit;end;
t1:=TVector.Vcreate(sc);
t2:=TVector.Vcreate(sc);
Randomize;
Form1.Image1.Canvas.Pen.Color:=RGB(random(255),
random(255),random(255));
if cufz <= 0 then begin
   MessageBox('Для частотных характеристик не зада-
на ненулевая частота среза');
   exit;
   end;
de:=TDifferentialEquation.DECreate(cfnl,cfnr,nu,
cod,omega,alpha,tend,tstep,kos);
de.FreqChar;
if (cod >= 31) and (cod <= 34) then begin
  t1.vector:=FreqChMtr.matrix[cod-30];
  t2.vector:=FreqChMtr.Matrix[0];
  pt:=GetCoord(t1,t2);
  Form1.Image1.Canvas.Polyline(pt);
end;
if cod = 35 then begin
  t1.vector:=FreqChMtr.matrix[2];
  t2.vector:=FreqChMtr.Matrix[1];
  pt:=GetCoord(t1,t2);
  Form1.Image1.Canvas.Polyline(pt);
end;
end;

procedure PaintPortret(const cod : integer);
var fnc:integer;
    t1,t2:TVector;
begin
```

```
if not isData then begin MessageBox('Не введены
данные ');exit;end;
t1:=TVector.Vcreate(sc);
t2:=TVector.Vcreate(sc);
Randomize;
Form1.Image1.Canvas.Pen.Color:=RGB(random(255),
random(255),random(255));
fnc:=cod-40;
de:=TDifferentialEquation.DECreate(cfnl,cfnr,nu,
fnc,omega,alpha,tend,tstep,kos);
de.Sol;
t1.vector:=SolMtr.matrix[0];
t2.vector:=SolMtr.Matrix[deriveNumber];
pt:=GetCoord(t1,t2);
Form1.Image1.Canvas.Polyline(pt);
end;

// Освобождаем память от мусора.
{ Процедура Free - стандартный метод для всех объ-
ектов.(для подробной информации см. Help)}
procedure TForm1.FormClose(Sender: TObject; var
Action: TCloseAction);
begin
Action:=caFree; {Признак ,что приложение должно
быть выгружено из памяти}
Application.Terminate;{Выгрузка приложения из памя-
ти}
end;

 //Создание главной формы
procedure TForm1.FormCreate(Sender: TObject);
begin
penWid:=2; //По умолчанию толщина карандаша 0
 Image1.Canvas.pen.Width:=PenWid;{Устанавливаем
начальную толщину}
 Image1.Canvas.Brush.Color:=clBtnFace;
 Image1.Canvas.Pen.Color:=clBtnFace;
 Image1.Canvas.Rectangle(0,0, image1.clientwidth+5,
image1.clientheight+5);{Очищаем экран (на всякий
случай)}
end;

//Ступенчатое воздействие
```

```
procedure TForm1.mStupenClick(Sender:
TObject);//Ступенька
begin
ProcessPaint(2);
end;

//Реакция на экспоненту
procedure TForm1.exponentaClick(Sender: TObject);
begin
ProcessPaint(5);
end;

//Очистка области вывода
procedure TForm1.ClearClick(Sender: TObject);
begin
 Image1.Canvas.Pen.Width:=PenWid;
 Image1.Canvas.Pen.Color:=clBtnFace;
 Image1.Canvas.Rectangle(0,0, image1.clientwidth+5,
image1.clientheight+5);
 Image1.Canvas.Pen.Color:=clBlack;
end;

//Свободное движение
procedure TForm1.mLibertyClick(Sender: TObject);
begin
ProcessPaint(0);
end;

//Реакция на импульс
procedure TForm1.mImpulsClick(Sender: TObject);
begin
ProcessPaint(1);
end;

//Реакция на синус
procedure TForm1.msinusClick(Sender: TObject);
begin
if omega = 0 then begin
  MessageBox('Не задана частота синусоиды.');
  exit;
  end;
ProcessPaint(3);
end;
```

```
//Реакция на косинус
procedure TForm1.mcosinusClick(Sender: TObject);
begin
if omega = 0 then begin
  MessageBox('Не задана частота косинусоиды.');
  exit;
  end;
ProcessPaint(4);
end;

//Реакция на произвольное воздействие
procedure TForm1.proizvolnClick(Sender: TObject);
var i:integer;
begin
UserFunc:=TVector.VCreate(sc);
for i:=0 to sc-1 do UserFunc.Vector[i]:=
exp(-0.02*i*tstep)*cos(i*tstep*0.05);
ProcessPaint(6);
end;


//Фазо-частотная хар-ка
procedure TForm1.freqcharClick(Sender: TObject);
begin
PaintFreqChar(34);
end;

//Амплитудно-фазовая
procedure TForm1.amphfazeClick(Sender: TObject);
begin
PaintFreqChar(35);
end;

//Help contents
procedure TForm1.Contents1Click(Sender: TObject);
begin
 application.HelpFile:='oldu.hlp';{Указываем какой
файл справки мы подключаем в нашем проекте}
 Application.HelpCommand(HELP_FINDER, 0);{Выполняем
инструкцию из WinHelp 4.0}
end;
```

```
// Чтение из файла userfile.txt
// Чтение из файла userfile.txt
procedure TForm1.mReadClick(Sender: TObject);
var s:string;i:integer;ch:char;
begin
UserFunc:=TVector.VCreate(sc);
if opendialog1.Execute then begin
userfile:=opendialog1.FileName;
AssignFile(input,userfile);
reset(input);s:='';i:=0;
while not eof(input) do begin
 read(input,ch);
 if ch=' ' then begin
  UserFunc.vector[i]:=strtofloat(s);
  i:=i+1;
  s:='';
 end else s:=s+ch;
end;
end;
end;

procedure FSave(const a:TMatrix; F:TFilename);
var i,j,k:integer;s:string;
begin
if a=nil then exit;
assignFile(output,f);
rewrite(output);
for i:=0 to a.Row-1 do begin
j:=0;
while j<a.Col do begin
 s:=floattostr(a.matrix[i,j]);
 for k:=1 to length(s) do
write(output,copy(s,k,1));
 write(output,' ');
 j:=j+1;
end;
write(output,#13);
end;
closeFile(output);
end;
```

```
procedure TForm1.mSaveClick(Sender: TObject);
begin
FSave(SolMtr,'SolMtr.txt');
FSave(FreqChMtr,'FreqChMtr.txt');
end;

//Изменение толщины карандаша
procedure TForm1.PenClick(Sender: TObject);
begin
form4.ShowModal;
penWid:=WidP;{Получаем значение толщины карандаша (
ограничения на толщину : 0 < penWid < 10 )}
end;

//Вещественная частотная характеристика
procedure TForm1.RcharClick(Sender: TObject);
begin
PaintFreqChar(31);
end;

//Мнимая
procedure TForm1.ImCharClick(Sender: TObject);
begin
PaintFreqChar(32);
end;

//Амплитудная
procedure TForm1.ampCharClick(Sender: TObject);
begin
PaintFreqChar(33);
end;


//Фазовый портрет при свободном движении
procedure TForm1.plibClick(Sender: TObject);
begin
PaintPortret(40);
end;

//При импульсе на входе
procedure TForm1.pimpulsClick(Sender: TObject);
begin
PaintPortret(41);
```

```
end;

//При ступеньке на входе
procedure TForm1.pstupClick(Sender: TObject);
begin
PaintPortret(42);
end;

//При синусоиде на входе
procedure TForm1.psinClick(Sender: TObject);
begin
PaintPortret(43);
end;

//При экспоненте на входе
procedure TForm1.pexpClick(Sender: TObject);
begin
PaintPortret(45);
end;

//При косинусоиде на входе
procedure TForm1.pcosClick(Sender: TObject);
begin
PaintPortret(44);
end;

//При произвольном движении
procedure TForm1.puserClick(Sender: TObject);
begin
PaintPortret(46);
end;
{Конец файла.}
end.
```

13.2.7. Форма сведений о программе

```
//ABOUT.DFM

object Form3: TForm3
  Left = 90
  Top = 142
  BorderStyle = bsToolWindow
  Caption = 'Об авторах'
```

```
ClientHeight = 262
ClientWidth = 668
Color = clBtnFace
Font.Charset = DEFAULT_CHARSET
Font.Color = clWindowText
Font.Height = -11
Font.Name = 'MS Sans Serif'
Font.Style = []
OldCreateOrder = False
Position = poScreenCenter
PixelsPerInch = 96
TextHeight = 13
object Bevel1: TBevel
  Left = 224
  Top = 8
  Width = 441
  Height = 209
end
object Image1: TImage
  Left = 8
  Top = 8
  Width = 209
  Height = 241
  Picture.Data = {}
  Stretch = True
end
object Label1: TLabel
  Left = 232
  Top = 8
  Width = 322
  Height = 24
  Caption = 'Научный руководитель проекта '
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clRed
  Font.Height = -21
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  ParentFont = False
  Transparent = True
end
object Label2: TLabel
  Left = 488
  Top = 40
```

```
    Width = 122
    Height = 24
    Caption = ' А.П. Полищук'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clBlue
    Font.Height = -21
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    Transparent = True
  end
  object Label3: TLabel
    Left = 232
    Top = 64
    Width = 154
    Height = 24
    Caption = 'Программисты:'
    Color = clNone
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -21
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentColor = False
    ParentFont = False
    Transparent = True
  end
  object Label4: TLabel
    Left = 344
    Top = 96
    Width = 268
    Height = 20
    Caption = ' Федоренко Д.С. , Епишин А.В. '
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clBlue
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
  end
  object Label5: TLabel
    Left = 232
    Top = 136
```

```
    Width = 334
    Height = 20
    Caption = 'Программа создана в недрах КГПИ в
1998г.'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = 8454016
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    Transparent = True
  end
  object Label6: TLabel
    Left = 232
    Top = 168
    Width = 424
    Height = 16
    Caption = 'Все права защищены , по вопросам
приобретения коммерческой'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clPurple
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    Transparent = True
  end
  object Label7: TLabel
    Left = 232
    Top = 192
    Width = 284
    Height = 16
    Caption = 'версии обращаться по тел. 66-98-76
,66-56-96'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clPurple
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    Transparent = True
  end
  object BitBtn1: TBitBtn
```

```
    Left = 320
    Top = 232
    Width = 75
    Height = 25
    Caption = '&Ok'
    TabOrder = 0
    OnClick = BitBtn1Click
    Kind = bkYes
  end
end
```

13.2.8. Модуль сведений о программе

```
//ABOUT.PAS


unit About;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls, Buttons,
ExtCtrls, jpeg;

type
  TForm3 = class(TForm)
    Image1: TImage;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Bevel1: TBevel;
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form3: TForm3;

implementation

{$R *.DFM}

procedure TForm3.BitBtn1Click(Sender: TObject);
begin
form3.close;
end;

end.
```

13.2.9. Файл конфигурации

```
//LAPLAS.CFG


-$A+
-$B-
-$C+
-$D+
-$E-
-$F-
-$G+
-$H+
-$I+
-$J+
-$K-
-$L+
-$M-
-$N+
-$O+
-$P+
-$Q-
-$R-
-$S-
-$T-
-$U-
-$V+
-$W-
```

```
-$X+
-$YD
-$Z1
-cg
-
AWinTypes=Windows;WinProcs=Windows;DbiTypes=BDE;Dbi
Procs=BDE;DbiErrs=BDE;
-H+
-W+
-M
-$M16384,16777216
-K$04000000
```

13.2.10. Файл проекта

```
//LAPLAS.DPR


program Laplas;

uses
  Forms,
  main in 'main.pas' {Form1},
  InputD in 'InputD.pas' {Form2},
  About in 'About.pas' {Form3},
  PenW in 'PenW.pas' {Form4};


begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm3, Form3);
  Application.CreateForm(TForm4, Form4);
  Application.Run;
end.
```

14. Приложения

14.1. Математические классы на языке C++

14.1.1. Базовый класс параметризованных векторов

```
//VECTOR.H


#ifndef __VECTOR_H
#define __VECTOR_H
#ifndef __COMPLEX_H
#include <complex.h>
#endif
#ifndef __FSTREAM_H
#include <fstream.h>
#endif
#ifndef __IOMANIP_H
#include <iomanip.h>
#endif
#ifndef __STDLIB_H
#include <stdlib.h>
#endif
#ifndef __MATH_H
#include <math.h>
#endif
#ifndef __EXCEPT_H
#include <except.h>
#endif
#ifndef __CSTRING_H
#include <cstring.h>
#endif

/*параметризованный класс для работы с векторными
объектами */
template <class YourOwnFloatType>/*подставьте свой
тип */
class vector
{ //приватные данные

  virtual void In(istream &);
```

```
  virtual void Out(ostream &);
  protected:
   long m;//размерность (длина) вектора
  YourOwnFloatType* vec;/*указатель на элементы
вектора*/
  /*виртуальные функции чтения из потока и записи в
поток; их необходимо переопределить в производных
классах для обеспечения возможности использования
единых перегруженных операторов << и >> */
 public://общедоступные данные и функции
  vector(char *);/*загрузка вектора из файла :
dimesion data1 data2 ...*/
  vector();/*создание пустого вектора единичной
размерности */
  vector(long);/*создание пустого вектора заданной
размерности */
  vector(long,YourOwnFloatType *);/*создание векто-
ра заданной размерности, заполняемого данными из
массива */
  vector(vector<YourOwnFloatType> &);/*конструктор
копирования */
  ~vector();//деструктор
  vector<YourOwnFloatType> operator+(vector
<YourOwnFloatType>);//сложение двух векторов
  vector<YourOwnFloatType> operator+=(vector
<YourOwnFloatType>);//сложение с присвоением
  vector<YourOwnFloatType> operator-(vector
<YourOwnFloatType>);//вычитание
  vector<YourOwnFloatType> operator-=(vector
<YourOwnFloatType>);//вычитание с присвоением
  YourOwnFloatType operator*(vector
<YourOwnFloatType>);//скалярное умножение
  friend vector<YourOwnFloatType>
operator*(YourOwnFloatType , vector
<YourOwnFloatType>);//умножение числа на вектор
  vector<YourOwnFloatType> operator*
(YourOwnFloatType );//умножение вектора на число
  void operator*=(YourOwnFloatType );/*умножение
вектора на число с присвоением */
  friend ostream &operator<<(ostream &,
vector<YourOwnFloatType> &);//вывод вектора в поток
  friend istream &operator>>(istream &, vector
<YourOwnFloatType> &);//ввод вектора из потока
```

```
  vector<YourOwnFloatType>
operator=(vector<YourOwnFloatType> );//присвоение
  vector<YourOwnFloatType> operator-();/*унарный
минус */
  vector<YourOwnFloatType> operator+();/*унарный
плюс */
  vector<YourOwnFloatType>
operator~();//нормализация (нормирование)
//(определение направляющих косинусов)
  YourOwnFloatType operator!();//модуль вектора
  long IsEqual(vector<YourOwnFloatType>);/*эта вир-
туальная функция сравнивает текущий вектор с объек-
том, лежащим по адресу, передаваемому через обоб-
щённый указатель */
  friend long operator==(vector<YourOwnFloatType> ,
    vector<YourOwnFloatType> );/*проверка на равен-
ство */
  friend long operator!=(vector<YourOwnFloatType> ,
    vector<YourOwnFloatType> );/*проверка на нера-
венство */
  YourOwnFloatType &operator[](long a);
//индексирование элементов вектора
  long getm() { return m; }//размерность вектора
};

/*
  Основными объектами линейной алгебры являются
вектора и матрицы, которые средствами языка C++ до-
статочно легко превратить в соответствующие классы,
сохранив при этом естественное представление матри-
цы как упорядоченного кортежа арифметических векто-
ров, а вектора - как упорядоченного кортежа объек-
тов любой природы.
*/


/*
  Создавать вектор можно по-разному. Например, если
он находится на внешнем устройстве в формате m d1
d2 ... dm, где m - размерность вектора, a di - его
компоненты, то имеем следующий конструктор:
*/
template <class YourOwnFloatType>
```

```
vector<YourOwnFloatType>::vector(char *f)/*имя фай-
ла*/
{
  long i;

  ifstream fp=f;//пытаемся открыть файл
  if(!fp)//если не удалось
    throw xmsg("Не могу открыть файл "+string(f)+
"\n");
  fp>>m;//вводим размерность
  if(m<=0)//проверка на корректность
    throw xmsg("Размерность вектора некоррект-
на\n");//диагностика
  try
  {
    vec=new YourOwnFloatType[m];/*попытка выделения
памяти */
  }
  catch(xalloc)
  {
    throw xmsg("Не хватает памяти\n");
  }
  for(i=0;i<m&&fp>>vec[i];i++);/*считывание из фай-
ла */
}


/*
  В случае, когда нам известна лишь размерность
вектора, но неизвестны его составляющие, предпола-
гаем, что данный вектор является нулевым:
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType>::vector(long a):m(a)
//размерность вектора
{
  long i;

  if(m<=0)//проверка размерности
    throw xmsg("Размерность вектора некоррект-
на\n");//диагностика
  try
  {
```

```
    vec=new YourOwnFloatType[m];/*попытка выделения
памяти */
  }
  catch(xalloc)
  {
    throw xmsg("Не хватает памяти\n");
  }
  for(i=0;i<m;vec[i++]=0);/*обнуление компонент
вектора */
}


/*
  Наконец, нам могут быть известны как размерность,
так и компоненты вектора:
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType>::vector(long a,
YourOwnFloatType *v):m(a)
/*этот конструктор принимает размер и указатель на
данные */
{
  long i;

  if(m<=0)//проверка размерности
    throw xmsg("Размерность вектора некоррект-
на\n");//диагностика
  try
  {
    vec=new YourOwnFloatType[m];/*попытка выделения
памяти */
  }
  catch(xalloc)
  {
    throw xmsg("Не хватает памяти\n");
  }
  for(i=0;i<m;i++)
    vec[i]=v[i];/*копирование из внешнего массива в
вектор */
}


/*
```

```
  Есть ещё один случай, когда мы ничего не можем
сказать о размерности и компонентах вектора - при
создании массива векторов, то есть матрицы, когда
для оператора new требуется конструктор без пара-
метров или когда размер вектора заранее неизвестен.
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType>::vector():m(1)
{
  try
  {
    vec=new YourOwnFloatType[m];/*попытка выделения
памяти */
  }
  catch(xalloc)
  {
    throw xmsg("Не хватает памяти\n");
  }
  *vec=0;//обнуляем единственный имеющийся элемент
}


/*
  В реальных расчетах могут использоваться вектора
больших размерностей, поэтому размещаются они в
свободной памяти компьютера, а когда необходимость
в них отпадает - уничтожаются.
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType>::~vector()
{
  delete []vec;//уничтожение динамического массива
}


/*
  Необходимость в индексации вектора возникает в
двух случаях:
  - при получении составляющей вектора по её номеру
и
  - при изменении не всего вектора, а только одной
его составляющей.
```

```
  При этом, конечно, следует учитывать возможность
ошибочного задания номера составляющей : допустимый
диапазон значений [0,m).
*/
template <class YourOwnFloatType> YourOwnFloatType
&vector<YourOwnFloatType>::operator[](long a)
{
  if(a>=0&&a<m)//если всё ОК
    return vec[a];
  else/*при выходе за пределы вектора ругаемся как
Паскаль */
  {
    cerr<<"Индекс "<<a<<" вне диапазона вектора\n";
    return vec[0];
  }
}


/*
  Создавая вектор, можно попутно инициализировать
его данными из уже существующего:
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType>::vector(
vector<YourOwnFloatType> &ex) : m(ex.m)
/*это конструктор копирования, принимающий ссылку
на вектор */
{
  try
  {
    vec=new YourOwnFloatType[m];/*попытка выделения
памяти */
  }
  catch(xalloc)
  {
    throw xmsg("Не хватает памяти\n");
  }
  for(long i=0;i<m;i++)
    vec[i]=ex[i];/*здесь при копировании ex исполь-
зуется уже индексация */
}
```

```
/*
  Сложение векторов является алгебраической опера-
цией только тогда, когда вектора одинаковой размер-
ности. Результатом сложения является вектор той же
размерности, что и исходные, компонентами которого
является сумма соответствующих компонент исходных
векторов.
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType> vector<YourOwnFloatType>::
operator+(vector<YourOwnFloatType> f)
{
  if(f.m!=m)//проверка на равенство размерностей
    throw xmsg("Слагаемые векторы имеют различные
длины\n");//диагностика
  vector<YourOwnFloatType> temp(m);/*создаём вре-
менный вектор */
  /*здесь работают операции индексирования для всех
трёх векторов */
  for(long i=0;i<m;i++)
    temp[i]=f[i]+this->vec[i];
  return temp;//возвращаем результирующий вектор
}


//сокращённая операция "сложение с присвоением"
template <class YourOwnFloatType>
vector<YourOwnFloatType> vector<YourOwnFloatType>::
operator+=(vector<YourOwnFloatType> f)
{  *this=(*this)+f;}


/*
  Введём несколько вспомогательных унарных опера-
ций:
  - "минус":
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType>
vector<YourOwnFloatType>::operator-()
{
  vector<YourOwnFloatType> temp(m);/*создаём вре-
менный вектор */
```

```
  /*Если this - это указатель на текущий объект
векторного класса, то *this - это сам текущий объ-
ект класса vector, то есть тот, с которым мы сейчас
работаем. А к любому векторному объекту мы можем
применить операцию индексирования */
  for(long i=0;i<m;i++)
    temp[i]=-(*this)[i];
  //temp[i]=-vec[i];
  //temp.vec[i]=-vec[i];
  //temp.operator[](i)=-operator[](i); etc...
  return temp;//возвращаем результирующий вектор
}


//унарный плюс
template <class YourOwnFloatType>
vector<YourOwnFloatType>
vector<YourOwnFloatType>::operator+()
{
  return *this;//возвращаем самого себя
}


/*
  Операция, которую алгебраической назвать нельзя -
это, скорее, пример очень распространённого тернар-
ного отношения "скалярное произведение двух векто-
ров":
*/
template <class YourOwnFloatType>
YourOwnFloatType
vector<YourOwnFloatType>::operator*(
      vector<YourOwnFloatType> f)
{
  if(f.m!=m)
    throw xmsg("Умножение векторов с несовпадающими
размерами невозможно\n");//диагностика
  YourOwnFloatType temp=0;
  for(long i=0;i<m;i++)
    temp+=f[i]*(this->vec[i]);/*суммируем произве-
дения составляющих векторов */
  return temp;
}
```

```
/*
  Модуль вектора - бинарное отношение, которое ми
определим не совсем стандартно, а именно как квад-
ратный корень скалярного произведения вектора на
самого себя:
*/
template <class YourOwnFloatType>
inline YourOwnFloatType
vector<YourOwnFloatType>::operator!()//modulus
{
  return sqrt((*this)*(*this));
}


/*
- нормирование вектора по модулю
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType>
vector<YourOwnFloatType>::operator~()
{
  vector<YourOwnFloatType> temp(m);
  /*скалярное произведение текущего объекта на са-
мого себя */
  YourOwnFloatType modul=!(*this);
  for(long i=0;i<m;i++)
    temp[i]=(*this)[i]/modul;/*направляющие косину-
сы */
  return temp;
}


/*
  Важным при подсчётах является тернарное отношение
"умножение числа на вектор":
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType> operator*
(YourOwnFloatType ld,vector<YourOwnFloatType> v)
{ long r=v.getm();
  for(long i=0;i<r;i++)
```

```
    v[i]=v[i]*ld;/*скорее, это даже "удлиннение"
вектора */
  return v;
}


/*
  Тернарное отношение "умножение числа на вектор"
является коммутативным, поэтому через него можно
определить и умножение вектора на число:
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType> vector<YourOwnFloatType>::
operator*(YourOwnFloatType ld)
{
  return ld*(*this);/*очень просто - вызвали другую
функцию */
}


//операция сокращённого умножения вектора на число
template <class YourOwnFloatType>
void vector<YourOwnFloatType>::
operator*=(YourOwnFloatType ld)
{
  *this=ld*(*this);
}


/*
  Имея определённые бинарную операцию сложения век-
торов и унарную получения вектора, противоположного
к данному, можно на векторном языке, не обращаясь к
компонентам векторов, определить операцию вычита-
ния:
*/
template <class YourOwnFloatType>
vector<YourOwnFloatType> vector<YourOwnFloatType>::
operator-(vector<YourOwnFloatType> f)
{
  return (*this+(-f));
  //return operator+(-f);
}
```

```
//операция сокращённого вычитания
template <class YourOwnFloatType>
vector<YourOwnFloatType> vector<YourOwnFloatType>::
operator-=(vector<YourOwnFloatType>f)
{
  *this=*this-f;
}


/*
  При переписывании одного вектора в другой возмож-
ны два случая:
  1. если размерность обоих векторов совпадает, то
просто заменяем составляющие первого вектора компо-
нентами второго;
  2. в противном случае безжалостно уничтожаем пер-
вый вектор и создаём снова, используя второй как
строительный материал.
*/

template <class YourOwnFloatType>
vector<YourOwnFloatType> vector<YourOwnFloatType>::
operator=(vector<YourOwnFloatType> x)
{
  if(m!=x.m)//если размеры не совпадают
  {
    delete []vec;/*уничтожаем содержимое текущего
вектора */
    m=x.m;//устанавливаем новый размер
    try
    {
      vec=new YourOwnFloatType[m];/*попытка выделе-
ния памяти */
    }
    catch(xalloc)
    {
      throw xmsg("Не хватает памяти\n");
    }
  }
  for(long i=0;i<m;i++)
```

```
    vec[i]=x[i];/*копируем данные из вектора x в
текущий */
  /*присвоение - это бинарная операция, первым па-
раметром которой является объект, которому присваи-
вают, вторым - объект, который присваивают. При
этом первый объект, в отличие от всех остальных би-
нарных операций, меняется, и он же возвращается в
качестве результата (это бывает необходимым для
операций вида a=b=c;) */
  return *this;
}


/*Эта функция сравнивает текущий вектор с вектором,
лежащим по адресу x. Для каждого класса, производ-
ного от векторного, не имеет смысла переопределять
операторные функции проверки на равенство и нера-
венство - достаточно переопределить эту виртуальную
функцию */
template <class YourOwnFloatType>
long vector<YourOwnFloatType>::IsEqual
(vector<YourOwnFloatType>x)
{
  if(m!=x.getm())//при несовпадении размерностей
    return 0; //констатируем несовпадение векторов
  for(long i=0;i<m;i++)
    if((*this)[i]!=x[i])
      return 0;//если хоть один элемент не совпал
  return 1;
}


/*
  Сравнение векторов является тернарным отношением,
результатом которого является число нуль, если век-
тора не равны и единица в противном случае. Два
вектора будем считать равными, если они имеют оди-
наковые длины и их соответствующие составляющие
совпадают:
*/
template <class YourOwnFloatType>
long operator==(vector<YourOwnFloatType> f,
vector<YourOwnFloatType> s)
```

```
{
  return f.IsEqual(s);
}


/*
  Неравенство векторов определим через равенство и
операцию отрицания:
*/
template <class YourOwnFloatType>
inline long operator!=(vector<YourOwnFloatType> f,
vector<YourOwnFloatType> s)
{
  return !(f==s);
}


/*
  Мощный I/O-механизм C++ позволяет в естественной
форме выводить (вводить) вектора на любое устрой-
ство отображения информации:
*/

/*Для универсализации считывания и записи вектора в
поток снова прибегнем к механизму виртуальных функ-
ций. С этой целью, по аналогии с printOn, определим
две функции - одну для ввода, другую - для вывода*/
template <class YourOwnFloatType>
void vector<YourOwnFloatType>::In(istream &is)
{
  for(long i=0;i<m;i++)
    is>>(*this)[i];
}


template <class YourOwnFloatType>
void vector<YourOwnFloatType>::Out(ostream &os)
{
  for(long i=0;i<m;i++)
  {
    os.precision(6);
    os<<(*this)[i]<<" ";/*компоненты разделяем про-
белами */
```

```
  }
}


//вывод в поток
template <class YourOwnFloatType>
ostream &operator<<(ostream &os,
vector<YourOwnFloatType> &x)
{
  x.Out(os);
  return os;
}


/*
  - ввод из потока
*/
template <class YourOwnFloatType>
istream &operator>>(istream &is,
vector<YourOwnFloatType> &x)
{
  x.In(is);
  return is;/*принимаем и возвращаем ссылку на по-
ток ввода */
}

typedef complex<double> dcomplex;
typedef vector<dcomplex> cvector;
typedef vector<double> dvector;

#endif
```

14.1.2. Параметризованный класс матриц

```
//MATRIX.H


#ifndef __MATRIX_H
#define __MATRIX_H
#ifndef __VECTOR_H
#include "vector.h"
#endif
#ifndef __FSTREAM_H
```

```
#include <fstream.h>
#endif
#ifndef __MATH_H
#include <math.h>
#endif
#ifndef __STDIO_H
#include <stdlib.h>
#endif
#ifndef __EXCEPT_H
#include <except.h>
#endif
#ifndef __CSTRING_H
#include <cstring.h>
#endif
#ifndef __COMPLEX_H
#include <complex.h>
#endif
typedef complex<double> dcomplex;
//параметризованный класс для работы с матричными
объектами
template  <class YourOwnFloatType>
class _RWSTDExportTemplate matrix
{ //приватные данные
  long m,n;/*row, columns - размерность матрицы в
строках и столбцах */
  vector<YourOwnFloatType> *mtr; /*указатель на
вектор данных */
 public://общедоступные члены
  matrix(char *);/*загрузка матрицы из файла в фор-
мате m n d11 d12 ... dmn */
  matrix();//пустая матрица размером 1x1
  matrix(long,long);/*пустая матрица заданного раз-
мера */
  matrix(long,long,YourOwnFloatType *);/*матрица
size1xsize2 из массива */
  matrix(matrix &);//конструктор копирования
  ~matrix();//деструктор
  friend matrix<YourOwnFloatType> operator+
(matrix<YourOwnFloatType> &,
matrix<YourOwnFloatType> &);//сложение
  friend matrix<YourOwnFloatType> operator-
(matrix<YourOwnFloatType> ,
matrix<YourOwnFloatType> );//вычитание
```

```
  friend matrix<YourOwnFloatType>
operator*(matrix<YourOwnFloatType> &,
matrix<YourOwnFloatType> &);//умножение матриц
  friend matrix<YourOwnFloatType>
operator*(YourOwnFloatType , matrix
<YourOwnFloatType> &);//умножение числа на матрицу
  friend matrix<YourOwnFloatType>
operator*(matrix<YourOwnFloatType> &,
YourOwnFloatType );//умножение матрицы на число
  friend ostream &operator<<(ostream &,
matrix<YourOwnFloatType> &);//вывод матрицы в поток
  friend istream &operator>>(istream &, matrix
<YourOwnFloatType> &);//ввод матрицы из потока
  /*первый параметр - квадратная матрица NxN, вто-
рой - Nx1 (матричное уравнение) */
  friend matrix<YourOwnFloatType>
SLAE_Orto(matrix<YourOwnFloatType> &,
matrix<YourOwnFloatType> &);//метод ортогонализации
  friend matrix<YourOwnFloatType> SLAE_Gauss(matrix
<YourOwnFloatType> &, matrix<YourOwnFloatType> &);
//метод Гаусса с выбором главного элемента
  friend YourOwnFloatType det2(matrix
<YourOwnFloatType> ); //аналитический детерминант
  friend YourOwnFloatType det (matrix
<YourOwnFloatType> );  //численный детерминант
  matrix<YourOwnFloatType> minor(long,long);
         /*минор - создание матрицы из имеющейся
без заданных строки и столбца */
  matrix<YourOwnFloatType>
operator=(matrix<YourOwnFloatType> );//присвоение
  matrix<YourOwnFloatType> operator*=(matrix
<YourOwnFloatType> &x);/*набор сокращённых операций
умножения,*/
  matrix<YourOwnFloatType> operator+=
(matrix<YourOwnFloatType> &x);//сложения и
  matrix<YourOwnFloatType> operator-=
(matrix<YourOwnFloatType> &x);//вычитания
  matrix<YourOwnFloatType> operator^(long);
//степень матрицы как операция
  matrix<YourOwnFloatType> operator^=(long);
//сокращённая степень
```

```
  friend matrix<YourOwnFloatType> pow(matrix
<YourOwnFloatType> &,long);/*степень матрицы как
дружественная функция */
  matrix<YourOwnFloatType> operator~();
//транспонирование
  matrix<YourOwnFloatType> operator!();
//аналитическое обращение матрицы
  matrix<YourOwnFloatType> operator*();
//численное обращение матрицы
  YourOwnFloatType operator&();/*численный детерми-
нант унарная операция */
  operator YourOwnFloatType(); /*быстрый детерми-
нант как оператор преобразования к типу-параметру
*/
  matrix<YourOwnFloatType> operator-();/*унарный
минус */
  matrix<YourOwnFloatType> operator+();/*унарный
плюс */
  friend long operator==(matrix<YourOwnFloatType>
&, matrix<YourOwnFloatType> &);/*проверка на равен-
ство */
  friend long operator!=(matrix<YourOwnFloatType>
&, matrix<YourOwnFloatType> &);/*проверка на нера-
венство */
  vector<YourOwnFloatType> &operator[](long a);
//индексирование матрицы
  long getm() { return m; }//число строк
  long getn() { return n; }//число столбцов
};


long sign(long double x);//-1,0,1 - получение знака


double fabs(dcomplex x)
{
  return abs(x);
}


/*
    Напомним, что матрица - это тоже вектор, эле-
ментами которого являются не числовые объекты, а
```

арифметические вектора. В связи с этим между векторным и матричным классами есть много общего, однако иногда можно наблюдать и существенные отличия. Возьмём, скажем, классический файловый метод хранения данных. Если для вектора достаточно было указать одно служебное число - его длину (размерность), то для матрицы их требуется уже два, причём первое из этих чисел будет определять количество векторов в матрице, а второе - размерность каждого из этих векторов. В файловых операциях с векторами нам поможет вышеопределённая операция введения вектора с некоторого устройства:

```
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>::
matrix(char *f)//имя файла с матрицей
{
  long i;

  ifstream fp=f;//пытаемся открыть файл
  if(!fp)
    throw xmsg("Не могу открыть файл
"+string(f)+"\n");
  fp>>m>>n;//размеры матрицы считываем из файла
  if(m<=0||n<=0)
    throw xmsg("Некорректный размер матрицы\n");
  try
  {
    //здесь работает конструктор без параметров
    mtr=new vector<YourOwnFloatType>[m];/*попытка
выделения памяти */
  }
  catch(xalloc)
  {
    throw xmsg("Не хватает памяти\n");
  }
  for(i=0;i<m;i++)/*и только здесь вектора расширяется до нужной размерности */
    mtr[i]=vector<YourOwnFloatType>(n);
  for(i=0;i<m&&fp>>mtr[i];i++);/*при вводе используем перегруженную операцию класса vector */
}
```

```
/*
  зная только размеры матрицы, ми можем считать её
нулевым элементом данного размера и сконструировать
соответствующим образом:
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>::matrix(long a, long b):
m(a),n(b)//число строк и число столбцов
{
  long i;

  if(m<=0||n<=0)
    throw xmsg("Некорректный размер матрицы\n");
  try
  {
    //здесь работает конструктор без параметров
    mtr=new vector<YourOwnFloatType>[m];/*попытка
выделения памяти */
  }
  catch(xalloc)
  {
    throw xmsg("Не хватает памяти\n");
  }
  for(i=0;i<m;i++)/*расширяем вектора до нужного
размера */
  mtr[i]=(vector<YourOwnFloatType>(n));
}


/* Получив о матрице все возможные сведения, имеет
смысл, сконструировав её, инициализировать соответ-
ствующими элементами: */
template <class YourOwnFloatType>
matrix<YourOwnFloatType>::matrix(long a,long b,
YourOwnFloatType *mt):m(a),n(b)
{
  if(m<=0||n<=0)
    throw xmsg("Некорректный размер матрицы\n");
  try
  {
    //здесь работает конструктор без параметров
```

```
    mtr=new vector<YourOwnFloatType>[m];/*попытка
выделения памяти */
  }
  catch(xalloc)
  {
    throw xmsg("Не хватает памяти\n");
  }
  for(long i=0;i<m;i++)
    mtr[i]=vector<YourOwnFloatType>(n);//расширение
  for(long i=0,l=0;i<m;i++)
    for(long j=0;j<n;j++)
      mtr[i][j]=mt[l++];/*сконструировав, копируем
данные из массива в матрицу */
}


/*
  Обратный случай (когда мы не знаем ни размеров,
ни элементов матричных векторов) решается достаточ-
но просто - созданием матрицы из одного единствен-
ного вектора длиною в один элемент по известному
принципу "аби було":
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>::matrix():m(1),n(1)
{
  try
  {
    //здесь работает конструктор без параметров
    mtr=new vector<YourOwnFloatType>[m];/*попытка
выделения памяти -  обнулится он вектором */
  }
  catch(xalloc)
  {
    throw xmsg("Не хватает памяти\n");
  }
}


/*
  уничтожение матрицы автоматически уничтожает все
её векторы:
*/
```

```
template <class YourOwnFloatType>
matrix<YourOwnFloatType>::~matrix()
{
  delete []mtr; /*вызов деструкторов для всех век-
торов */
}


/*
  Как и вектора, матрицы тоже бывает необходимым
проиндексировать; результатом этого действия, есте-
ственно, будет соответствующий вектор:
*/
template <class YourOwnFloatType> vector
<YourOwnFloatType> &matrix<YourOwnFloatType>::
operator[](long a)
{
  static vector<YourOwnFloatType> error;
  if(a>=0&&a<m)
    return mtr[a];/*а дальше можно индексировать
вектор */
  else
  {
    cerr<<"Индекс "<<a<<" вне матрицы\n";
    return mtr[0];
  }
}


/*
  Если в памяти ЭВМ уже есть какая-то матрица, с
неё можно снять копию-слепок:
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>::matrix /*конструктор ко-
пирования */ (matrix<YourOwnFloatType> &ex) :
m(ex.m), n(ex.n)
{
  try
  {
    //здесь работает конструктор без параметров
    mtr=new vector<YourOwnFloatType>[m];/*попытка
выделения памяти */
```

```
  }
  catch(xalloc)
  {
    throw xmsg("Не хватает памяти\n");
  }
  for(long i=0;i<m;i++)
    mtr[i]=ex[i];/*очень громоздкое индексирование
и присваивание векторов */
}


/*
  Сложение матриц одинаковой размерности сводится к
сложению соответствующих векторов:
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType> operator+
(matrix<YourOwnFloatType> &f,
matrix<YourOwnFloatType> &s)
{
  if(f.m!=s.m||f.n!=s.n)
    throw xmsg("Размерности слагаемых матриц не
совпадают\n");
  matrix<YourOwnFloatType> temp(f.m,f.n);
//временная матрица
  for(long i=0;i<f.m;i++)
    temp[i]=f[i]+s[i];//слагаем вектора матриц
  return temp;//возвращаем результат
}


/*
  Как и для векторов, определим унарный "минус":
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::operator-()
{
  matrix<YourOwnFloatType> temp(m,n);
  for(long i=0;i<m;i++)
    temp[i]=-(*this)[i];
  return temp;//возвращаем результат
}
```

```
//унарный плюс
template <class YourOwnFloatType>
matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::operator+()
{
  return *this; /*возвращаем себя без каких-либо
изменений */
}


/*
  В отличие от скалярного произведения векторов,
умножение матриц является бинарной алгебраической
операцией, однако только для отдельных видов мат-
риц, к тому же эта операция не является коммутатив-
ной!
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType> operator*(matrix
<YourOwnFloatType> &f, matrix<YourOwnFloatType> &s)
{
  if(f.n!=s.m)
    throw xmsg("Умножение матриц с данными размера-
ми невозможно\n");
  matrix<YourOwnFloatType> temp(f.m,s.n);
  for(long j=0;j<s.n;j++)
    for(long i=0;i<f.m;i++)
      for(long k=0;k<f.n;k++)
        temp[i][j]+=f[i][k]*s[k][j];
  return temp;
}


/*
  Полезной является унарная операция увеличения
матрицы в некоторое число раз:
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType> operator*
(matrix<YourOwnFloatType> &f,YourOwnFloatType s)
{
```

```
  matrix<YourOwnFloatType> temp=f;
  for(long i=0;i<f.m;i++)
    temp[i]=temp[i]*s; /*здесь используем умножение
вектора на число */
  return temp;
}


/*
    Оператор возведения матрицы в целую степень
можно определить так:
  - любая матрица в нулевой степени является еди-
ничной,
  - положительная степень определяется через произ-
ведение,
  - отрицательная - через обращение матрицы и про-
изведение.
    Последний случай можно красиво реализовать с
использованием рекурсии, как, впрочем, и предыду-
щий.
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::operator^(long l)
{
  matrix<YourOwnFloatType> temp(getm(),getn());
  if(getm()!=getn())
    throw xmsg("Для неквадратных матриц степень не
определена\n");
  if(l==0)
  {
    for(long i=0;i<getm();i++)
      temp[i][i]=1;
    return temp;
  }
  if(l>0)
  {
    temp=(*this);
    for(long i=1;i<l;i++)
      temp*=(*this);
    return temp;
  }
  else
```

```
    return (*(*this))^(-l);
}


//сокращённая степень матрицы
template <class YourOwnFloatType>
matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::operator^=(long l)
{
  return (*this)=(*this)^l;
}


/*
  Степень матрицы в виде функции
*/
template <class YourOwnFloatType>
inline matrix<YourOwnFloatType>
pow(matrix<YourOwnFloatType> &x,long l)
{
  return x^l;
}


/*
  Умножение числа на матрицу реализуем через уже
имеющуюся операцию умножения матрицы на число для
сохранения коммутативности
*/
template <class YourOwnFloatType>
inline matrix<YourOwnFloatType> operator*
(YourOwnFloatType f, matrix<YourOwnFloatType> &s)
{
  return s*f;
}


/*
  Вычитание традиционно реализуем через сложение и
унарный минус:
*/
template <class YourOwnFloatType>
```

```
matrix<YourOwnFloatType> operator-(matrix
<YourOwnFloatType> f, matrix<YourOwnFloatType> s)
{
  matrix<YourOwnFloatType> tms=(-s),
  tm=f+tms;

  return tm;
  //return f+(-s);
}


/*
  При присвоении содержимое матрицы x переписывает-
ся в текущую сразу только тогда, когда их размеры
совпадают. В противном случае приходится уничтожать
текущую матрицу, заново её создавать с новыми раз-
мерами и лишь тогда производить повекторное копиро-
вание
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType> matrix<YourOwnFloatType>::
operator=(matrix<YourOwnFloatType> x)
{
  if(m!=x.m||n!=x.n)
  {
    delete []mtr;//уничтожаем содержимое матрицы
    m=x.m,n=x.n;//устанавливаем новые размеры
    try
    {
      mtr=new vector<YourOwnFloatType>[m];/*попытка
выделения памяти */
    }
    catch(xalloc)
    {
      throw xmsg("Не хватает памяти\n");
    }
  }
  for(long i=0;i<m;i++)
    mtr[i]=x[i];
  return *this;//возвращение себя
}
```

```
/*
  Набор сокращённых операций:
  - умножение
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::operator*=
(matrix<YourOwnFloatType> &x)
{
  matrix<YourOwnFloatType> tm=(*this)*x;
  (*this)=tm;
  return (*this);//=(*this)*x;
}


/*
  - сложение
*/
template <class YourOwnFloatType>
inline matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::operator+=
(matrix<YourOwnFloatType> &x)
{
  return (*this)=(*this)+x;
}


/*
  - вычитание
*/
template <class YourOwnFloatType>
inline matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::operator-=
(matrix<YourOwnFloatType> &x)
{
  return (*this)+=-x;/*используем только что сде-
ланное сокращённое сложение */
}


/*
  Очень часто бывает нужна унарная операция транс-
понирования:
```

```
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::operator~()
{
  matrix<YourOwnFloatType> temp(n,m);
  for(long j=0;j<n;j++)
    for(long i=0;i<m;i++)
      temp[j][i]=mtr[i][j];
  return temp;//переставлены столбцы и строки
}


/*
  Сравнивая матрицы, мы сначала учитываем, одинако-
вой ли они размерности, а потом в случае необходи-
мости сравниваем вектора:
*/
template <class YourOwnFloatType>
long operator==(matrix<YourOwnFloatType> &f,
matrix<YourOwnFloatType> &s)
{
  if(f.m!=s.m||f.n!=s.n)
    return 0;
  for(long i=0;i<f.m;i++)
    if(f[i]!=s[i])
      return 0;
  return 1;
}


/*
  Неравенство - оно и есть НЕ равество
*/
template <class YourOwnFloatType>
inline long operator!=(matrix<YourOwnFloatType> &f,
matrix<YourOwnFloatType> &s)
{
  return !(f==s);//!operator==(f,s);
}


/*
```

```
  Результат иногда хочется посмотреть. Например,
так:
*/
template <class YourOwnFloatType>
ostream &operator<<(ostream &os,
matrix<YourOwnFloatType> &x)
{
  for(long i=0;i<x.m;i++)/*построчный вывод векто-
ров матрицы */
    os<<x[i]<<"\n";
  return os;
}

/*Или так... (Запись матрицы в файл в градациях се-
рого) */
template <class YourOwnFloatType>
void writebmp(matrix<YourOwnFloatType> &x, char
*name)
{
  ofstream f(name,ios::out|ios::binary);/*Битовое
изображение */
  BITMAPFILEHEADER fh;//файловый заголовок
  BITMAPINFOHEADER ih;//информационный заголовок
  RGBQUAD bmiColors[256];//256 цветов
  fh.bfType=0x4d42;//буквы "BM"

fh.bfSize=sizeof(fh)+sizeof(ih)+sizeof(bmiColors)+
  x.getm()*((x.getn()%4==0)?x.getn():(x.getn()+4-
x.getn()%4));//размер файла
  fh.bfReserved1=fh.bfReserved2=0;

fh.bfOffBits=sizeof(fh)+sizeof(ih)+sizeof(bmiColors
);//смещение до данных
  f.write((unsigned char*)&fh,sizeof(fh));/*пишем
файловый заголовок */
  ih.biSize=40;//размер информационного заголовка
  ih.biWidth=x.getn();//ширина изображения
  ih.biHeight=x.getm();//высота
  ih.biPlanes=1;//количество планов изображения
  ih.biBitCount=8;//бит на цвет
  ih.biCompression=BI_RGB;//без сжатия
  ih.biSizeImage=0;//не актуален
```

```
  ih.biXPelsPerMeter=2963;/*рекомендуемое разреше-
ние по X */
  ih.biYPelsPerMeter=3158;//                по У
  ih.biClrUsed=256;//количество используемых цветов
  ih.biClrImportant=0;//не актуально
  f.write((unsigned char*)&ih,sizeof(ih));/*пишем
информационный заголовок */
  for(int i=0;i< (1 << ih.biBitCount);i++)/*цикл по
количеству цветов */
  {
    /*приравнивая красную, зелёную и голубую со-
ставляющие цвета, получаем оттенок серого */
    bmiColors[i].rgbBlue=bmiColors[i].rgbGreen=
bmiColors[i].rgbRed=i;
    bmiColors[i].rgbReserved=0;
    f.write((unsigned char*)(bmiColors+i),
sizeof(RGBQUAD)); //пишем элемент палитры
  }
  YourOwnFloatType __max=x[0][0],
__min=x[0][0];//поиск диапазона значений матрицы
  for(i=0;i<x.getm();i++)
    for(long j=0;j<x.getn();j++)
    {
      if(x[i][j]>__max)
        __max=x[i][j];
      if(x[i][j]<__min)
        __min=x[i][j];
    }
  for(i=x.getm()-1;i>=0;i--)/*строки в BMP-файле
лежат снизу вверх */
  {
    for(long j=0;j<x.getn();j++)//записываем строку
    { //формируем номер в палитре оттенков серого
      BYTE b=0xff*(x[i][j]-__min)/(__max-__min);
      f.write(&b,1);
    }
    if(x.getn()%4)/*дополняем строку до границы
двойного слова */
      for(j=0;j<4-x.getn()%4;j++)
        f.write(&j,1);
  }
}
```

```
/*
  Ввод матрицы из потока целиком перекладываем на
векторный класс, используя его метод для ввода
*/
template <class YourOwnFloatType>
istream &operator>>(istream &is,
matrix<YourOwnFloatType> &x)
{
  for(long i=0;i<x.m;i++)
    is>>x[i];
  return is;
}


/*
  В матричной форме систему линейных алгебраических
уравнений (СЛАУ) можно представить в виде
                          A*X=B,
  где A - матрица коэффициентов при неизвестных,
      X - вектор-столбец этих самих неизвестных, а
      B - вектор-столбец свободных членов, взятых с
обратными знаками.
  Если система имеет решение, то оно будет таким:
           A^(-1)*A*X=A^(-1)*B
           (A^(-1)*A)*X=A^(-1)*B
           E*X=A^(-1)*B
           X=A^(-1)*B,
  Существенным моментом является выбор метода реше-
ния. Для небольших систем (<200) можно использовать
прямые методы (например, метод Гаусса); для реаль-
ных расчётов ми рекомендуем метод ортогонализации,
в основе которого лежат ортогональные преобразова-
ния матриц, которые вы осуществляли ранее
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType> SLAE_Orto(matrix
<YourOwnFloatType> &f, matrix<YourOwnFloatType> &s)
{
  matrix<YourOwnFloatType>
mtr2(f.m+1,f.m+1),res(f.m,1);
  //формируем матрицу из двух для решения СЛАУ
  for(long i=0;i<f.m;i++)
```

```
    for(long j=0;j<f.n;j++)
      mtr2[i][j]=f[i][j];
  for(long i=0;i<f.m;i++)/*заносим в последнюю
строку 0 0 ... 0 1 */
    mtr2[i][f.m]=-s[i][0];
  mtr2[f.m][f.m]=1;

  mtr2[0]=~mtr2[0];//нормируем нулевую строку
  /*для улучшения сходимости повторяем этот процесс
3 раза */
  for(long k=0;k<3;k++)
  {
    for(long l=1;l<f.m+1;l++)
    {
      for(long i=0;i<l;i++)
      {
        YourOwnFloatType p=mtr2[l]*mtr2[i];
//скалярное произведение
          for(long j=0;j<(f.m+1);j++)
            mtr2[l][j]-=p*mtr2[i][j];
      }
      mtr2[l]=~mtr2[l];
    }
  }
  for(long i=0;i<f.m;i++)//переписываем результат
    res[i][0]=mtr2[f.m][i]/mtr2[f.m][f.m];
  return res;
}


/*
  Метод Гаусса с выбором главного элемента
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType> SLAE_Gauss(matrix
<YourOwnFloatType> &f, matrix<YourOwnFloatType> &s)
{
  long i,j,k,num;

  matrix<YourOwnFloatType>
mtr2(f.m,f.m+1),res(f.m,1);
  //формируем матрицу из двух для решения СЛАУ
  for(i=0;i<f.m;i++)
```

```
  for(j=0;j<f.n;j++)
    mtr2[i][j]=f[i][j];
for(i=0;i<f.m;i++)
  mtr2[i][f.m]=s[i][0];
//выбор главного элемента
for(i=0;i<f.m;i++)
{
  YourOwnFloatType max=mtr2[0][i];
  for(j=i,num=i;j<f.m;j++)
    if(fabs(mtr2[j][i])>fabs(max))
      max=fabs(mtr2[j][i]),num=j;
  if(num!=i)
    for(k=0;k<f.m+1;k++)
    {
      YourOwnFloatType temp=mtr2[num][k];
      mtr2[num][k]=mtr2[i][k];
      mtr2[i][k]=temp;
    }
}
for(i=0;i<f.m;i++)
  if(mtr2[i][i]==(YourOwnFloatType)0)
    throw xmsg("Возможно, матрица вырождена\n");
//Прямой ход Гаусса
for(i=0;i<f.m;i++)
{
  YourOwnFloatType sw=mtr2[i][i];
  for(j=0;j<f.m+1;j++)
    mtr2[i][j]/=sw;
  for(k=i+1;k<f.m;k++)
  {
    YourOwnFloatType c=mtr2[k][i];
    for(j=0;j<f.m+1;j++)
      mtr2[k][j]-=mtr2[i][j]*c;
  }
}
//Обратный ход Гаусса
for(i=f.m-2;i>=0;i--)
{
  YourOwnFloatType s=0;
  for(j=i+1;j<f.m;j++)
    s+=mtr2[i][j]*mtr2[j][f.m];
  mtr2[i][f.m]-=s;
}
```

```
  //переписываем результат
  for(i=0;i<f.m;i++)
    res[i][0]=mtr2[i][f.m];
  return res;
}


/*
  Для обращения матрицы и нахождения детерминанта
классическими методами нужна функция получения ми-
нора матрицы для данного её элемента
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::minor(long a, long b)
{
  matrix<YourOwnFloatType> temp(getm()-1,getn()-1);
  for(long i1=0,i2=0;i1<getm();i1++)
    if(i1!=a)
    {
      for(long j1=0,j2=0;j1<getn();j1++)
        if(j1!=b)
          temp[i2][j2]=(*this)[i1][j1],
          j2++;
      i2++;
    }
  return temp;
}


/*
  Классический детерминант - вычисление разложением
по одной из строк
*/
template <class YourOwnFloatType>
YourOwnFloatType det2(matrix<YourOwnFloatType> x)
{
  if(x.getm()!=x.getn())
    throw xmsg("Детерминант определён только для
квадратных матриц\n");
  if(x.getm()==1)/*особые случаи - детерминанты 1-
го и 2-го порядка */
    return x[0][0];
```

```
  if(x.getm()==2)
    return x[0][0]*x[1][1]-x[0][1]*x[1][0];
  YourOwnFloatType result=0;
  matrix<YourOwnFloatType> tminor,tmult;
  YourOwnFloatType tdet;
  for(long i=0;i<x.getm();i++)
   // {tminor=x.minor(1,i);
    //tmult=tminor*x[1][i];
    //tdet=det(tmult);
    //result+=tdet*pow(-1,i+1);
    result+=pow(-1,1+i)*det(x.minor(1,i))*x[1][i];
    //}
  return result;
}


/*
  Медленная (аналитическая) операция обращения
квадратной матрицы
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::operator!()
{
  if(getm()!=getn())
    throw xmsg("Для неквадратных матриц обращение
не определено\n");
  matrix<YourOwnFloatType> temp=*this;
  YourOwnFloatType _det=det(*this);
  if(_det==(YourOwnFloatType)0)
    throw xmsg("Обращение особенной матрицы невоз-
можно\n");
  for(long i=0;i<getm();i++)
    for(long j=0;j<getn();j++)
      temp[i][j]=pow(-1,2+i+j)*det(minor(j,i))
/_det;
  return temp;
}


/*
  Численное нахождение детерминанта методом Гаусса
*/
```

```
template <class YourOwnFloatType>
YourOwnFloatType det(matrix<YourOwnFloatType> x)
//быстрый детерминант
{
  YourOwnFloatType sw,c,det,max;
  long i,j,k,how,num;

  if(x.getm()!=x.getn())
    throw xmsg("Детерминант определён только для
квадратных матриц\n");
  if(x.getm()==1)
    return x[0][0];
  if(x.getm()==2)
    return x[0][0]*x[1][1]-x[0][1]*x[1][0];
  for(how=i=0;i<x.getm();i++)
  {
    max=x[0][i];
    for(j=i,num=i;j<x.getm();j++)
      if(fabs(x[j][i])>fabs(max))
      {
        max=fabs(x[j][i]);
        num=j;
      }
    if(num!=i)
    {
      for(k=0;k<x.getm();k++)
      {
        YourOwnFloatType temp=x[num][k];
        x[num][k]=x[i][k];
        x[i][k]=temp;
      }
      how++;
    }
  }
  for(i=0;i<x.getm();i++)
  {
    for(sw=x[i][i],j=i+1;j<x.getm();j++)
      x[i][j]/=sw;
    for(k=i+1;k<x.getm();k++)
      for(c=x[k][i],j=0;j<x.getm();j++)
        x[k][j]-=x[i][j]*c;
  }
  for(det=1,i=0;i<x.getm();i++)
```

```
    det*=x[i][i];
  det*=pow(-1,how);
  return det;
}


/*
  Быстрое (численное) обращение матрицы
*/
template <class YourOwnFloatType>
matrix<YourOwnFloatType>
matrix<YourOwnFloatType>::operator*()
{
  long i,j,k,l;
  YourOwnFloatType v,maxabs,s,tsr;
  matrix<YourOwnFloatType> temp(getm(),2*getn());


  if(getm()!=getn())
    throw xmsg("Для неквадратных матриц обращение
не определено\n");
  if(det(*this)==(YourOwnFloatType)0)
    throw xmsg("Обращение особенной матрицы невоз-
можно\n");
  for(i=0;i<getm();i++)
  {
    for(j=0;j<getn();j++)
      temp[i][j]=(*this)[i][j];
    for(j=getm();j<2*getm();j++)
      temp[i][j]=(j==i+getm()) ? 1 : 0;
  }
  for(i=0;i<getm();i++)
  {

for(maxabs=fabs(temp[i][i]),k=i,l=i+1;l<getm();l++)
      if(fabs(temp[l][i])>fabs(maxabs))
        maxabs=fabs(temp[l][i]), k=l;
    if(k!=i)
      for(j=i;j<2*getm();j++)
        v=temp[i][j], temp[i][j]=temp[k][j],
temp[k][j]=v;
  }
  for(i=0;i<getm();i++)
```

```
  {
    for(s=temp[i][i],j=i+1;j<2*getm();j++)
      temp[i][j]/=s;
    for(j=i+1;j<getm();j++)
    {
      for(tsr=temp[j][i],k=i+1;k<2*getm();k++)
        temp[j][k]-=temp[i][k]*tsr;
    }
  }
  for(k=getm();k<2*getm();k++)
    for(i=getm()-1;i>=0;i--)
    {
      for(tsr=temp[i][k],j=i+1;j<getm();j++)
        tsr-=temp[j][k]*temp[i][j];
      temp[i][k]=tsr;
    }
  matrix<YourOwnFloatType> result=(*this);
  for(i=0;i<getm();i++)
    for(j=getm();j<2*getm();j++)
      result[i][j-getm()]=temp[i][j];
  return result;
}


/*
  Быстрый детерминант как операция
*/
template <class YourOwnFloatType>
inline YourOwnFloatType
matrix<YourOwnFloatType>::operator&()
{
  return det(*this);
}


/*
  Интересный случай - детерминант рассматривается
как оператор преобразования матрицы в число
*/
template <class YourOwnFloatType>
inline matrix<YourOwnFloatType>::operator
YourOwnFloatType()
{
```

```
  return det(*this);/*используем быстрый детерми-
нант */
}


/*
  функция, определяющая знак числа
*/
inline long sign(long double x)
{
  return (x<0) ? -1 : 1;
}

typedef matrix<double> dmatrix;
typedef matrix<dcomplex> cmatrix;

#endif
```

14.1.3. Параметризованный класс полиномов

```
//POLYNOM.H


#ifndef __POLYNOM_H
#define __POLYNOM_H
#ifndef __VECTOR_H
#include "vector.h"
#endif
#ifndef __IOSTREAM_H
#include <iostream.h>
#endif

#define max(a,b)    (((a) > (b)) ? (a) : (b))
#define min(a,b)    (((a) < (b)) ? (a) : (b))

/*
 Параметризованный класс для полиномов
 Внутренний формат:
 a0+a1*x+a2*x^2+a3*x^3+a4*x^4+a5*x^5+...+a(n-
1)*x^(n-1)
 Ввод и вывод производится, начиная с коэффициента
при наивысшей степени
```

```
*/
//Наш полином будет базироваться на векторе
template <class YourOwnFloatType>
class polynom:public vector<YourOwnFloatType>
{
  //эти функции являются внутренними


  void In(istream &);/*по аналогии с векторами -
функции ввода */
  void Out(ostream &);//                и вывода
 public:
  void optimize();/*преобразование полинома в кано-
ническую форму */
  polynom<YourOwnFloatType> reverse();/*запись по-
линома в обратном порядке */
  polynom(char *);//полином из файла
  polynom(long,YourOwnFloatType *);/*полином из
массива */
  polynom(polynom<YourOwnFloatType> &);
//конструктор копирования
  polynom();//конструктор по умолчанию
  polynom(long);//полином заданной степени
  polynom<YourOwnFloatType> operator-();/*унарный
минус */
  polynom<YourOwnFloatType> operator+();/*унарный
плюс */
  polynom<YourOwnFloatType>
operator+(polynom<YourOwnFloatType>);//сложение
  polynom<YourOwnFloatType> operator+=
(polynom<YourOwnFloatType>);//сокращённое сложение
  polynom<YourOwnFloatType> operator-
(polynom<YourOwnFloatType>);//вычитание
  polynom<YourOwnFloatType> operator-=(polynom
<YourOwnFloatType>);//сокращённое вычитание
  polynom<YourOwnFloatType> operator*(polynom
<YourOwnFloatType>);//умножение полинома на полином
  void operator*=(polynom<YourOwnFloatType>);
//сокращённое умножение на полином
  friend polynom<YourOwnFloatType> operator*
(YourOwnFloatType, polynom<YourOwnFloatType>);
//умножение числа на полином
```

```
  polynom<YourOwnFloatType> operator*
(YourOwnFloatType); //умножение полинома на число
  void operator*=(YourOwnFloatType);/*сокращённое
умножение на число */
  //набор операций для сравнения полиномов
  friend int operator==(polynom<YourOwnFloatType> ,
polynom<YourOwnFloatType> );
  friend long operator!=(polynom<YourOwnFloatType>
f, polynom<YourOwnFloatType> t);
  YourOwnFloatType &operator[](long);/*индексация
полинома */
  YourOwnFloatType operator()(YourOwnFloatType);
//значение полинома в заданной точке
  friend
  long  div(polynom<YourOwnFloatType> , polynom
<YourOwnFloatType> , polynom<YourOwnFloatType> &,
polynom<YourOwnFloatType> &);//деление
  friend polynom<YourOwnFloatType> operator/
(polynom<YourOwnFloatType> , polynom
<YourOwnFloatType> );//целая часть от деления
  friend polynom<YourOwnFloatType> operator%
(polynom<YourOwnFloatType> ,
polynom<YourOwnFloatType> );//остаток от деления
  long IsEqual(void *);//проверка на равенство
  polynom<YourOwnFloatType> operator=
(polynom<YourOwnFloatType> );//присвоение
  friend polynom<YourOwnFloatType> derive
(polynom<YourOwnFloatType>,long);//производная
  friend polynom<YourOwnFloatType> integral
(polynom<YourOwnFloatType>,long);//интеграл
  friend polynom<YourOwnFloatType> pow
(polynom<YourOwnFloatType>,unsigned int);//степень
  polynom<YourOwnFloatType> operator^(unsigned
int);//степень как операция
  polynom<YourOwnFloatType> operator^=(unsigned
int);//сокращённая степень
};


/*конструкторы полинома будут аналогичны конструк-
торам вектора полином из файла */
template <class YourOwnFloatType>
```

```
polynom<YourOwnFloatType>::polynom(char *f):
vector<YourOwnFloatType>(f)
{
}


//полином степени a-1
template <class YourOwnFloatType>
polynom<YourOwnFloatType>::polynom(long a):
vector<YourOwnFloatType>(a)
{
}


//нуль-полином
template <class YourOwnFloatType>
polynom<YourOwnFloatType>::polynom():
vector<YourOwnFloatType>()
{
}


/*полином (a-1)-ой степени с коэффициентами из мас-
сива v */
template <class YourOwnFloatType>
polynom<YourOwnFloatType>::polynom(long a,
YourOwnFloatType *v):vector<YourOwnFloatType>(a,v)
{
}


//конструктор копирования
template <class YourOwnFloatType>
polynom<YourOwnFloatType>::
polynom(polynom<YourOwnFloatType>
&ex):vector<YourOwnFloatType>(ex)
{
}


/*для индексации полинома вызываем соответствующий
метод векторного класса */
template <class YourOwnFloatType>
```

```
YourOwnFloatType &polynom<YourOwnFloatType>::
operator[](long a)
{
  return (*(vector<YourOwnFloatType>*)this)[a];
}


//вычисление значения полинома в точке x
template <class YourOwnFloatType>
YourOwnFloatType polynom<YourOwnFloatType>::
operator()(YourOwnFloatType x)
{
/*
  YourOwnFloatType temp=0,px=1;
  for(long i=0;i<getm();i++,px*=x)
    temp+=(*this)[i]*px;
  return temp;
*/
  polynom<YourOwnFloatType> temp=2;
  temp[0]=-x,temp[1]=1;
  return ((*this)%temp)[0];
}


//унарный минус
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::operator-()
{
  return *(polynom*)&(-(*(vector
<YourOwnFloatType>*)this));
}


//унарный плюс - просто возвращаем себя
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::operator+()
{
  return *this;
}
```

```
//сложение двух полиномов
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::
operator+(polynom<YourOwnFloatType> f)
{
  long mx=max(f.getm(),m),mn=min(f.getm(),m);
  polynom<YourOwnFloatType> temp(mx);/*создаём вре-
менный полином */
  //здесь работают операции индексирования
  for(long i=mx-1;i>=mn;i--)
    temp[i]=(f.getm()>m)?f[i]:this->vec[i];
  for(long i=mn-1;i>=0;i--)
    temp[i]=f[i]+this->vec[i];
  temp.optimize();/*пока есть, удаляем 0-
коэффициент при старшей степени */
  return temp;
}


//сокращённое сложение, определяемое через обычное
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::
operator+=(polynom<YourOwnFloatType> f)
{
  return *this=*this+f;
}


/*вычитание полиномов, выраженное через сложение и
отрицание */
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::operator-
(polynom<YourOwnFloatType> f)
{
  polynom<YourOwnFloatType> t=-f;
  return (*this+t);
}


//сокращённое вычитание
```

```
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::operator-=
(polynom<YourOwnFloatType> f)
{
  *this=*this-f;
}


//умножение полиномов
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::operator*
(polynom<YourOwnFloatType> f)
{
  polynom<YourOwnFloatType> temp(f.getm()+m);
  for(long i=0;i<f.getm();i++)
    for(long j=0;j<m;j++)
      temp[i+j]+=f[i]*this->vec[j];
  temp.optimize();
  return temp;
}


//сокращённое умножение
template <class YourOwnFloatType>
//polynom<YourOwnFloatType>
void polynom<YourOwnFloatType>::operator*=
(polynom<YourOwnFloatType> f)
{
  *this=(*this)*f;
}


//умножение числа на полином
template <class YourOwnFloatType>
polynom<YourOwnFloatType> operator*
(YourOwnFloatType ld, polynom<YourOwnFloatType> v)
{
  v=*(polynom<YourOwnFloatType>*)
 &((*(vector<YourOwnFloatType>*)(&v))*ld);
  v.optimize();
  return v;
```

```
}


//умножение полинома на число
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::
operator*(YourOwnFloatType ld)
{
  return ld*(*this);
}


//сокращённое умножение на число
template <class YourOwnFloatType>
void polynom<YourOwnFloatType>::operator*=
(YourOwnFloatType ld)
{
  *this=ld*(*this);
}


//присвоение
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::
operator=(polynom<YourOwnFloatType> x)
{
  //присваиваем как векторы
  (*(vector<YourOwnFloatType>*)this)=
(*(vector<YourOwnFloatType>*)&x);
  optimize();//приводим к нормальному виду
  return *this;//и возвращаем результат
}


//установление актуальной степени многочлена
template <class YourOwnFloatType>
void polynom<YourOwnFloatType>::optimize()
{
  if(getm()!=1)//если полином не нулевой степени
  {
```

```
    if((*this)[getm()-1]==(YourOwnFloatType)0)
//если коэффициент при
    //наивысшей степени нулевой
    {
      polynom<YourOwnFloatType> temp(getm()-1);
//создаём временный полином
     //степени на 1 меньше
      for(long i=0;i<getm()-1;i++)
        temp[i]=(*this)[i];
      *this=temp;//переписываем его в текущий
      optimize();//снова проверяем полином
    }
  }
}


/*в очень редких случаях может быть необходимым ре-
версировать полином */
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::reverse()
{
  long i,it;
  it=getm(); polynom temp=*this;
  for(i=0;i<it;i++)
    temp[i]=(*this)[getm()-i-1];
  return temp;
}


//неотрицательная степень полинома как функция
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
pow(polynom<YourOwnFloatType> x,unsigned int p)
{
  polynom<YourOwnFloatType> temp;
  if(p==0)
    temp[0]=1;/* полином в нулевой степени - это
просто число 1 */
  else
  {
    temp=x;
    for(unsigned i=0;i<p-1;i++)
```

```
      temp*=x;
  }
  return temp;//возвращаем результирующий полином
}


//производная j-го порядка
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
derive(polynom<YourOwnFloatType> p, long j)
{
  if(j==0)/*нулевая производная полинома есть сам
полином */
    return p;
  if(j<0)/*отрицательная производная интерпретиру-
ется как интеграл */
    return integral(p,-j);
  if(p.getm()==1)
    return polynom<YourOwnFloatType>();/*если более
понижать некуда */
  polynom<YourOwnFloatType> result=p.getm()-1;
  for(long i=0;i<result.getm();i++)/*вычисляем
первую производную */
    result[i]=(i+1)*p[i+1];
  if(j==1)//если её и надо было найти -
    return result;//возвращаем результат
  else
    return derive(result,j-1);/*вычисляем производ-
ную от данной */
}


//интеграл от полинома
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
integral(polynom<YourOwnFloatType> p, long j)
{
  if(j<=0)/*отрицательная кратность интегрирования
трактуется как производная */
    return derive(p,-j);
  polynom<YourOwnFloatType> result=p.getm()+1;
  for(long i=0;i<p.getm();i++)
    result[i+1]=p[i]/(i+1);
```

```
  if(j==1)
    return result;
  else
    return integral(result,j-1);
}


//степень как операция
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::operator^(unsigned int
p)
{
  return pow(*this,p);
}


//сокращённая степень
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
polynom<YourOwnFloatType>::operator^=(unsigned int
p)
{
  return (*this)=pow(*this,p);
}


/*Перегружать операторы ввода из потока и вывода в
поток необходимости нет - достаточно перегрузить
две виртуальные функции /ввод полинома из потока,
начиная с коэффициентов при старших степенях */
template <class YourOwnFloatType>
void polynom<YourOwnFloatType>::In(istream &is)
{
  for(long i=getm()-1;i>=0;i--)
    is>>(*this)[i];
  optimize();
}


/*вывод полинома в поток, начиная с коэффициентов
при старших степенях */
template <class YourOwnFloatType>
```

```
void polynom<YourOwnFloatType>::Out(ostream &os)
{
  for(long i=getm()-1;i>=0;i--)
  {
    os.precision(5);//устанавливаем точность вывода
    os<<(*this)[i]<<" ";/*компоненты разделяем про-
белами */
  }
}


//сравнение текущего полином с лежащим по адресу x
template <class YourOwnFloatType>
long polynom<YourOwnFloatType>::IsEqual(void *x)
{
  polynom<YourOwnFloatType>
test1=*(polynom<YourOwnFloatType>*)x,
    test2=(*this);
  test1.optimize();
  test2.optimize();
  return ((vector<YourOwnFloatType>*)&test1)->
vector<YourOwnFloatType>::IsEqual(&test2);
}


/*
  Из двух многочленов одинаковой длины меньше тот,
у которого коэффициент при старшей степени меньше.
При равенстве рассматриваем более низкую степень и
т.д.
*/
/*
template <class YourOwnFloatType>
long operator<(polynom<YourOwnFloatType> &f,
polynom<YourOwnFloatType> &s)
{
  polynom<YourOwnFloatType> test1=f,test2=s;
  test1.optimize();
  test2.optimize();
  if(test1.getm()<test2.getm())
    return 1;
  if(test1.getm()>test2.getm()||test1==test2)
    return 0;
```

```
  //точно не равны - выясняем, кто же из них меньше
  for(long i=test1.getm()-1;i>=0;i--)
    if(test1[i]<test2[i])
      return 1;
    else
      if(test1[i]>test2[i])
        return 0;
  return 0;
}
*/

/*все остальные операции определяем через уже из-
вестные */
/*
template <class YourOwnFloatType>
long operator>(polynom<YourOwnFloatType> &f,
polynom<YourOwnFloatType> &s)
{
  return (!(f<s))&&(f!=s);
}

*/
template <class YourOwnFloatType>
int operator==(polynom<YourOwnFloatType> f,
polynom<YourOwnFloatType> s)
{ if(f.getm()!=s.getm()) return 0;
 for(int i=0;i<f.getm();i++)
 if(f[i]!=s[i]) return 0;
 return 1;
}


template <class YourOwnFloatType>
long operator!=(polynom<YourOwnFloatType> f,
polynom<YourOwnFloatType> s)
{
  return !(f==s);
}


/*
  деление полинома на полином по алгоритму Евклида
*/
```

```
template <class YourOwnFloatType>
long div(polynom<YourOwnFloatType> f,
polynom<YourOwnFloatType> g,
polynom<YourOwnFloatType> &l,
polynom<YourOwnFloatType> &r)
{
  f.optimize();
  g.optimize();
  polynom<YourOwnFloatType> tp;
  if(g.getm()==1 && g[0]==tp[0])/*попытка деления
на ноль */
    throw xmsg("Попытка деления на 0\n");

  if(f.getm()<g.getm())
  {
    l=polynom<YourOwnFloatType>();
    r=f;
    return 1;
  }
  /*

for(l=polynom<YourOwnFloatType>();f.getm()>=g.getm(
);)
  {
    polynom<YourOwnFloatType> x(2);
    x[1]=1;
    x^=f.getm()-g.getm();
    x*=f[f.getm()-1]/g[g.getm()-1];
    f=f-g*x;
    r=f;
    l+=x;
  }
  */
  int rg=g.getm();
  //Полином-частное разностного порядка
  polynom<YourOwnFloatType> pch(f.getm()-
g.getm()+1);
  for(;f.getm()>(rg-1);)
  {//Текущий коэф частного
   YourOwnFloatType cch=f[f.getm()-1]/g[g.getm()-
1];
   // Полином текущего порядка
```

```
   polynom<YourOwnFloatType> p(f.getm()-
g.getm()+1);
   //получает старший коэффициент-остальные нулевые
   pch[f.getm()-g.getm()]=p[f.getm()-g.getm()]=cch;
   // Вычитаем из делимого
   f=f-p*g;
  }
  // остаток
   r=f;
  l=pch;//частное-результат
  return 1;
}


//целая часть от деления
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
operator/(polynom<YourOwnFloatType> f,
polynom<YourOwnFloatType> s)
{
  polynom<YourOwnFloatType> l,r;
  div(f,s,l,r);
  return l;
}


//остаток от деления
template <class YourOwnFloatType>
polynom<YourOwnFloatType>
operator%(polynom<YourOwnFloatType> f,
polynom<YourOwnFloatType> s)
{
  polynom<YourOwnFloatType> l,r;
  div(f,s,l,r);
  return r;
}

typedef polynom<dcomplex> cpolynom;

#endif
```

14.1.4. Класс полиномиальных уравнений

```
//EQUATION.H


/*Этот файл включения содержит объявление двух ти-
пов - комплексного полинома и комплексного вектора,
а также заголовки четырёх функций для решения урав-
нений 2-ой, 3-ей, 4-ой и высших степеней + функции
нахождения собственных значений и собственных век-
торов */

#include "matrix.h"
#include "polynom.h"
cvector square(dcomplex,dcomplex,dcomplex);
cvector kardano(double,double,double,double);
cvector
ferrary(double,double,double,double,double);
cvector newton(cpolynom);
cvector eigenval(cmatrix&);
cmatrix eigenvec(cmatrix&,cvector&);
/*Решение квадратного уравнения с комплексными ко-
эффициентами*/
cvector square(dcomplex a2,dcomplex a1,dcomplex a0)
{
  dcomplex a=a2,b=a1,c=a0;
  /*Найденные комплексные корни должны быть записа-
ны в двухкомпонентный комплексный вектор-результат
*/
  cvector res(2);
  //проверим, не нулевой ли коэффициент при x^2
  if(a!=dcomplex(0,0))
  {
    /*если да, ищем корни через квадратичный дис-
криминант */
    dcomplex D=b*b-4*a*c;
    res[0]=(-b+sqrt(D))/(2*a);/*и заносим их в со-
ответствующие */
    res[1]=(-b-sqrt(D))/(2*a);/*компоненты вектора-
результата */
  }
  else
```

```
    res[0]=res[1]=-c/b;/*иначе решаем уравнение
первой степени */
  return res;
}


/*Решение кубического уравнения с действительными
коэффициентами методом Кардано-Тартальи */
cvector kardano(double a3,double a2,double a1,
double a0)
{
  //Общий вид уравнения, корни которого мы ищем -
  //a3*x^3+a2*x^2+a1*x+a0=0
  /*так как уравнение имеет третью степень, то и
корней у него три, */
  cvector res=3;/*что и определяет размерность век-
тора-результата */
  if(a3==0)//если коэффициент при x^3 нулевой,
  {
    //решаем соответствующее квадратное уравнение
    cvector res2=square(a2,a1,a0);
    /*в этом случае принимаем, что два корня явля-
ются совпадающими */
    res[0]=res[1]=res2[0];
    res[2]=res2[1];
  }
  else
  {
    /*иначе - приводим кубическое уравнение к кано-
ническому виду, когда коэффициент при третьей сте-
пени равен 1 */
    double a=a2/a3,b=a1/a3,c=a0/a3;
    //находим слагаемые кубического дискриминанта
    double p=b-a*a/3,q=2*a*a*a/27-a*b/3+c;
    /*проведя переобозначение x=y-a/3, решаем в
дальнейшем уравнение y^3+p*y+q=0 */
    //находим кубический дискриминант
    double diskr=pow(q/2,2)+pow(p/3,3);
    /* если дискриминант равен 0, то имеем 3 дей-
ствительных корня, из них два совпадающих */
    if(diskr==0)
    {
      res[0]=3*q/p;
```

```
      res[1]=res[2]=-res[0]/2;
    }
    /*один действительный корень и два комплексно
сопряжённых */
    if(diskr>0)
    {
      double what=-q/2+sqrt(diskr);
      double u0=(what>0)?pow(what,1.0/3.0):-pow(-
what,1.0/3.0);
      double v0=-p/(3*u0);
      res[0]=u0+v0;
      res[1]=res[2]=-res[0]/2.0;
      dcomplex k3(0,sqrt(3)*(u0-v0)/2);
      res[1]+=k3;
      res[2]-=k3;
    }
    //три различных действительных корня
    if(diskr<0)
    {
      cvector zkub12=square(1,q,-p*p*p/27);
      dcomplex u0=pow(zkub12[0],1/3.);
      res[0]=2*real(u0);
      res[1]=-real(u0)-imag(u0)*sqrt(3);
      res[2]=-real(u0)+imag(u0)*sqrt(3);
    }
    //переходим обратно от y к x
    for(int i=0;i<3;i++)
      res[i]-=a/3;
  }
  return res;//возвращаем вектор-результат
}


/*Метод Феррари для решения уравнений четвёртой
степени с действительными коэффициентами */
cvector ferrary(double a4,double a3,double a2,
double a1,double a0)
{
  cvector res=4;//всего корней будет 4
  if(!a4)/*если коэффициент при четвёртой степени x
нулевой, пытаемся решить уравнение третьей степени
*/
  {
```

```
    cvector res3=kardano(a3,a2,a1,a0);
    res[0]=res[1]=res3[0];
    for(int i=1;i<3;i++)
      res[i+1]=res3[i];
  }
  else
  {
    double a=a3/a4,b=a2/a4,c=a1/a4,d=a0/a4;
    /*составляем и находим корни кубической резоль-
венты */
    cvector cy0=kardano(1,-b,-4*d+a*c,-d*a*a+4*b*d-
c*c);
    double y0;
    /*ищем хотя бы один действительный корень, с
помощью которого сводим уравнение четвёртой степени
к совокупности квадратных */
    for(int i=0;i<3;i++)
      if(!imag(cy0[i]))
      {
        y0=real(cy0[i]);
        break;
      }
    double A=a*a/4-b+y0,B=a*y0/2-c;
    double x12=-B/(2*A);
    //решаем квадратные уравнения
    cvector sq1=square(1,a/2-sqrt(A),
y0/2+sqrt(A)*x12);
    cvector sq2=square(1,a/2+sqrt(A),y0/2-
sqrt(A)*x12);
    for(long i=0;i<2;i++)
      res[i]=sq1[i],res[i+2]=sq2[i];
  }
  return res;//возвращаем результат
}


/*модифицированный метод Ньютона поиска комплексных
корней полинома */
cvector newton(cpolynom p)
{
  double t=1,eps=1e-8,j;

  p.optimize();
```

```
  cvector result=p.getm()-1,tv;
  /*если вектор-результат одномерный, то имеем дело
с полиномом первой степени */
  if(result==tv)
  {
    result[0]=-p[0]/p[1];
    return result;
  }
  //если двумерный - с квадратным уравнением и т.д.
  if(result.getm()==2)
    return square(p[2],p[1],p[0]);
  if(result.getm()==3&&!imag(p[3])&&!imag(p[2])
&&!imag(p[1])&&!imag(p[0]))
    return kardano(real(p[3]),real(p[2]),
real(p[1]),real(p[0]));
  if(result.getm()==4&&!imag(p[4])&&!imag(p[3])
&&!imag(p[2])&&!imag(p[1])&&
     !imag(p[0]))
    return ferrary(real(p[4]),real(p[3]),
real(p[2]),real(p[1]),real(p[0]));
  dcomplex z=0;/* начальное приближение к корню по-
ложим равным (0,0) */
  do
  {
    dcomplex dz=0;
    /*находим порядок и значение ненулевой произ-
водной в точке z */
    for(j=1;dz==dcomplex(0);dz=derive(p,j++)(z));
    z+=t*pow(-p(z)/dz,1/(--j));/*модифицируем при-
ближение */
    t*=(1-eps);
  }while(abs(p(z))>=eps);/*повторяем до достижения
заданной точности */
  /*В этом цикле находится только один корень z.
Для нахождения остальных делим исходный полином на
x-z, понижая тем самым степень на единицу, и нахо-
дим ещё один корень, и т.д. до первой степени */
  result[0]=z;
  cpolynom z1=2;
  z1[0]=-z,z1[1]=1;
  cvector more=newton(p/z1);
  for(long i=1;i<result.getm();i++)
    result[i]=more[i-1];
```

```
  return result;//возвращаем вектор результата
}


/*нахождение собственных значений произвольной мат-
рицы */
cvector eigenval(cmatrix &x)
{
  if(x.getn()!=x.getm())
    throw xmsg("Для неквадратных матриц собственные
значения не определены\n");
  /*Вид характеристического полинома хорошо изве-
стен – это обычный полином n степени с (n+1)-им ко-
эффициентом. Для того, чтобы найти эти коэффициен-
ты, используем следующий способ: характеристический
полином прямыми методами получается развёрткой де-
терминанта, и значение полинома в некоторой точке x
соответствует значению детерминанта матрицы, из
диагональных элементов которой вычтен x. Тогда,
взяв набор из (n+1) несовпадающих точек, мы можем
составить систему из n уравнений, линейных относи-
тельно коэффициентов характеристического полинома.
Результатом решения этой системы и будут искомые
коэффициенты, зная которые, мы можем найти корни
полинома, и являющиеся искомыми собственными значе-
ниями */
  cmatrix a(x.getm()+1,x.getm()+1),b(x.getm()+1,1);
  for(long i=0;i<a.getm();i++)//набор точек
  {
    for(long j=0;j<a.getm();j++)/*вычисляем степени
при */
      a[i][j]=pow(i,a.getm()-j-1);//коэффициентах
    cmatrix temp=x;
    for(long j=0;j<temp.getm();j++)
      temp[j][j]-=i;/*вычитаем из элементов главной
диагонали */
    b[i][0]=det(temp);/*текущую точку и находим де-
терминант */
  }
  cmatrix result=SLAE_Orto(a,b);//решаем СЛАУ
  cpolynom lambdaeq=result.getm();
  for(long i=0;i<result.getm();i++)/*формируем ха-
рактеристический */
```

```
    lambdaeq[i]=result[result.getm()-i-1][0];
//полином и находим
  return newton(lambdaeq);/*его корни модифициро-
ванным методом Ньютона */
}


/*нахождение собственных векторов при известных
матрице и её собственных значениях */
cmatrix eigenvec(cmatrix &x,cvector &e)
{
  if(x.getn()!=x.getm())
    throw xmsg("Для неквадратных матриц собственные
вектора не определены\n");
  if(x.getm()!=e.getm())/*при несовпадении размер-
ностей */
    throw xmsg("Собственные значения не соответ-
ствуют матрице\n");
  cmatrix ev(x.getm(),x.getm());/*в строках этой
матрицы будут находиться собственные вектора */
  for(long i=0;i<e.getm();i++)/*цикл по собственным
значениям */
  {
    cmatrix temp=x;
    for(long j=0;j<temp.getm();j++)/*вычитаем еди-
ничную матрицу, умноженную */
      temp[j][j]-=e[i]; //на собственное значение
    /*Для нахождения ненулевого решения однородной
системы уравнений проделаем следующее: примем одну
из компонент вектора равной 1. Затем перенесём в
системе уравнений в правую часть с противоположным
знаком столбец, соответствующий этой компоненте.
Решив такую СЛАУ, мы найдём все остальные компонен-
ты вектора. Для того, чтобы вектора не совпадали,
для каждого из них "объединичим" свою компоненту -
сначала первую, затем - вторую и т.д.*/
    cmatrix a(temp.getm()-1,temp.getm()-1),
b(temp.getm()-1,1);//матрицы для СЛАУ
    for(long k=0;k<a.getm();k++)
      for(long j=0,l=0;j<temp.getm();j++)
        if(j==i)/*компоненту вектора, номер которой
соответствует номеру собственного */
```

```
          b[k][0]=-temp[k][j];/*значения, сделаем
равным 1, а соответствующий столбец */
        else /*переносим в правую часть; все
остальное */
          a[k][l++]=temp[k][j];/*остаётся в левой
части */
    cmatrix res=SLAE_Orto(a,b);/*решаем СЛАУ отно-
сительно неизвестных составляющих */
    for(long j=0,l=0;j<ev.getm();j++)/*компонуем
собственный вектор */
      ev[i][j]=((i==j)?dcomplex(1,0):res[l++][0]);
      cvector ev1=~ev[i];
      ev[i]=ev1;
      /*ev[i]=(~ev[i]);для удобства нормируем его
по модулю */
  }
  return ev;//возвращаем массив векторов
}
```

14.2. Математические классы на объектном Паскале

14.2.1. Класс комплексных чисел

```
//T_COMPLEX.PAS


unit T_Complex;

interface
uses math,dialogs;
type

    TComplex = record
     re,im:extended;
    end;
Function CExp(c:TComplex):TComplex ;far;    {Число
е комплексной степени.}
Function CCPow(a1,a2:TComplex):TComplex;far;{a1^a2}
function Complex(r,i:real):TComplex;far; {Создание
компл. Числа}
function Arg(c:TComplex):real;far; {Аргумент числа}
```

```
function CNorm(c:TComplex):real;far; {Модуль числа}
function CSumma(c1,c2:TComplex):
TComplex;far;//Сумма
function CRazn(c1,c2:TComplex):
TComplex;far;//Разность
function CMult(c1,c2:TComplex):
TComplex;far;//Произведение
function CPow(c1:TComplex;s:real):
TComplex;far;//Возведение в степень
function CEquiv(c1,c2:TComplex):
boolean;far;//Сравнение
function CDiv(c1,c2:TComplex):TComplex;far;
//Деление чисел
function CMultOnDig(c:TComplex;d:real):
TComplex;far;//Умножение на действит. число
procedure MessageBox(s:string);far; //Сообщение

implementation

function CMultOnDig(c:TComplex;d:real):TComplex;
begin
result.re:=c.re*d;
result.im:=c.im*d;
end;

 {Экспонента комплексного аргумента}
function CExp(c:TComplex):Tcomplex;
var res:TComplex;
begin
res.re:=Exp(c.re)*cos(c.im);
res.im:=Exp(c.re)*sin(c.im);
result:=res;
end;

{a1 возводится в степень a2}
function CCpow(a1,a2:TComplex):TComplex;
var f,t1,r1,u2,v2,r:real;
    cx:TComplex;
begin
r1:=cnorm(a1);
u2:=a2.re;
v2:=a2.im;
t1:=arg(a1);
```

```
r:=power(r1,u2)*exp(-v2*t1);
f:=v2+ln(r1)+u2*t1;
cx.re:=r*cos(f);
cx.im:=r*sin(f);
result:=cx;
end;

{Ф-ция имитирующая действие конструктора - создаёт
комплексное число с реальной частью r и веществен-
ной i}
function Complex(r,i:real):TComplex;
var f:TComplex;
begin
f.re:=r;
f.im:=i;
result:=f;
end;

procedure MessageBox(s:string);
begin
MessageDlg(s,mtInformation,[mbOk],1);{Метод из мо-
дуля Dialogs}
end;

function Arg(c:TComplex):real; { Аргумент комплекс-
ного числа}
begin
result:=ArcTan2(c.im,c.re);{Лучше чем просто ArcTan
из модуля System }
end;

function CNorm(c:TComplex):real; {Модуль комплекс-
ного числа}
begin
result:=sqrt(c.re*c.re+c.im*c.im);
end;

function CSumma(c1,c2:TComplex):TComplex;
// Находим сумму комплексных чисел
begin
result.re:=c1.re+c2.re;
result.im:=c1.im+c2.im;
end;
```

```
function CRazn(c1,c2:TComplex):TComplex;
// Находим разность комплексных чисел
begin
result.re:=c1.re-c2.re;
result.im:=c1.im-c2.im;
end;

function CMult(c1,c2:TComplex):TComplex;
// Находим произведение комплексных чисел
begin
result.re:=c1.re*c2.re-c1.im*c2.im;
result.im:=c1.re*c2.im+c2.re*c1.im;
end;

{Формула Муавра}
function CPow(c1:TComplex;s:real):TComplex;
// Возведение комплексного числа c1 в степень s
begin
result.re:=power(Cnorm(c1),s)*cos(arg(c1)*s);
result.im:=power(Cnorm(c1),s)*sin(arg(c1)*s);
end;

function CEquiv(c1,c2:TComplex):boolean;
// Сравнение комплексных чисел
begin
if (round(c1.re*1e+6)=round(c2.re*1e+6)) and
(round(c1.im*1e+6)=round(c2.im*1e+6))
then result:=true else result:=false;
end;

function CDiv(c1,c2:TComplex):TComplex;
// Деление комплексных чисел
begin
if (c2.re=0) and (c2.im=0) then
begin MessageBox('Деление на ноль;(Модуль
T_Complex)');
result:=complex(0,0);exit;
end;
result.re:=(c1.re*c2.re+c1.im*c2.im)/(c2.re*c2.re+
c2.im*c2.im);
result.im:=(c1.im*c2.re-c1.re*c2.im)/
(c2.re*c2.re+c2.im*c2.im);
```

```
exit;
end;

// End file !!!!!
end.
```

14.2.2. Класс действительных векторов

```
//T_VECTOR.PAS


unit T_Vector;

interface
uses math,dialogs;

type
    DVector=array of extended; { Тип : динамический
вектор. Заполнение динамического вектора начинается
с 0 и заканчивается len-1}

    TVector = class { Определяем класс векторов с
действительными числами }
    len:integer;    // Длина вектора
    Vector:DVector; // Сам вектор
    procedure SetLen(l:integer);virtual;{Установка
длины вектора}
    constructor VCreate(l:integer); { Конструктор
класса : создаёт вектор длины l }

    private

    protected

    public
    // Внутренние ф-ции
    function VMultOnDig(d:Real):
TVector;virtual;//Умножение вектора на число
    function VNormir:TVector;virtual; //Нормировка
    function VModul:Real;virtual;
//Возвращает модуль вектора

    published
```

```
    end;
    // Внешние ф-ции
    function VSumma(v1,v2:DVector):
DVector;far;//Сумма векторов
    function VRazn(v1,v2:DVector):DVector;far;
//Разность векторов
    function VMult(v1,v2:DVector):DVector;far;
//Произведение векторов
    function VScalarMult(v1,v2:DVector):
real;far;//Скалярное произведение векторов
    procedure MessageBox(s:string);far;
        // Просто сообщение


implementation
procedure TVector.SetLen(l:integer);
begin
setlength(vector,l); {Установка длины динамического
массива vector равной l}
len:=l;    // Заносим в поле len длину вектора
end;

function TVector.VMultOnDig(d:real):TVector;
//Умножение вектора на число
var t:TVector;i:integer;
begin
t:=TVector.VCreate(len); { Создаём экземпляр объек-
та Tvector}
for i:=0 to len-1 do t.Vector[i]:=Vector[i]*d;
result:=t;
end;

procedure MessageBox(s:string);
begin
MessageDlg(s,mtInformation,[mbOk],1);// Без слов
end;


constructor TVector.VCreate(l:integer);{Конструктор
для объекта}
 TVector
var i:integer;
```

```
begin
inherited Create; {Вызываем родительский конструк-
тор ( TObject.Create )}
// Все объекты являются наследниками TObject
setlen(l);        // Вызываем нашу ф-цию установки
длинны вектора
for i:=0 to len-1 do vector[i]:=0;{Сразу заносим
нули}
end;

function TVector.VModul:real; // Модуль вектора
var tt:real;i:integer;
begin
tt:=0;
for i:=0 to len-1 do
  tt:=tt+Power(vector[i],2); {Сумма квадратов всех
элементов массива}
result:=Sqrt(tt);//Корень квадратный из этой суммы
end;

function VSumma(v1,v2:DVector):DVector; {Сумма век-
торов}
var i:integer;d:DVector;
begin
if high(v1)>high(v2) then begin
setlength(d,high(v1)+1);
for i:=0 to high(v1)
    do d[i]:=v1[i]; { Заносим в вектор-результат
пока значения большего вектора}
for i:=0 to high(v2) do d[i]:=v1[i]+v2[i]; {Находим
сумму}
end else begin
setlength(d,high(v2)+1);
for i:=0 to high(v2) do d[i]:=v2[i]; {здесь анало-
гично}
for i:=0 to high(v1) do d[i]:=v1[i]+v2[i];
end;
result:=d;
end;

function VRazn(v1,v2:DVector):DVector; {Разность
векторов}
var i:integer;d:DVector;
```

```
begin
if high(v1)>high(v2) then begin
setlength(d,high(v1)+1);
for i:=0 to high(v1) do d[i]:=v1[i];
for i:=0 to high(v2) do d[i]:=v1[i]-v2[i];
end else begin
setlength(d,high(v2)+1);
for i:=0 to high(v2) do d[i]:=v2[i];
for i:=0 to high(v1) do d[i]:=v1[i]-v2[i];
end;
result:=d;
end;

function VMult(v1,v2:DVector):DVector; {Произведе-
ние векторов}
var i:integer;d:DVector;
begin
if high(v1)>high(v2) then begin
setlength(d,high(v1)+1);
for i:=0 to high(v1) do d[i]:=v1[i];
for i:=0 to high(v2) do d[i]:=v1[i]*v2[i];
end else begin
setlength(d,high(v2)+1);
for i:=0 to high(v2) do d[i]:=v2[i];
for i:=0 to high(v1) do d[i]:=v1[i]*v2[i];
end;
result:=d;
end;

function VScalarMult(v1,v2:DVector):real; {Скаляр-
ное произведение векторов}
var d:DVector;tt,n1,n2:real;i:integer;
    tc:TVector;
begin
if high(v1)=high(v2) then begin {Нужно чтобы векто-
ра были одинаковой длины}
tc:=TVector.VCreate(high(v1)+1);
tt:=0;
d:=VMult(v1,v2); // Произведение векторов
for i:=0 to high(d) do tt:=tt+d[i]; {Сумма элемен-
тов вектора d}
tc.vector:=v1;n1:=tc.VModul;//n1- модуль вектора v1
tc.vector:=v2;n2:=tc.VModul;//n2- модуль вектора v2
```

```
if (n1=0) or (n2=0) then begin
MessageBox('Один из векторов нулевой;(Модуль
T_Vector)');
result:=0;
exit;
end;
result:=tt/(n1*n2); // Результат
tc.Free; // Очистка памяти от мусора
exit;
end else result:=0;
end;

function TVector.VNormir:TVector;
var t:TVector;tt:real;i:integer;
begin
t:=TVector.VCreate(len);
tt:=VModul;
// Проверка  деления на ноль
if tt=0 then begin
MessageBox('Деление на ноль;(Модуль T_Vector)');
result:=self;
exit;
end;
for i:=0 to len-1 do t.Vector[i]:=Vector[i]/tt;
result:=t;
exit;
end;
//End file
end.
```

14.2.3. Класс комплексных векторов

```
//T_CVECTOR.PAS


unit T_CVector;

interface

uses T_Complex,// Модуль комплексных чисел
     dialogs; // Стандартный модуль диалоговых окон

type
```

```
    CVector=array of TComplex; /* Тип : динамиче-
ский вектор комплексных чисел */

    TCVector = class // Класс комплексного вектора
    len:integer;     // Длина вектора
    Vector:CVector;  // Сам вектор
    procedure SetLen(l:integer);virtual;
//Установка длины вектора
    constructor CVCreate(l:integer);  /* Создание
вектора длины l*/

    private

    protected

    public
    // Внутренние ф-ции
    function CVMultOnDig(d:TComplex):
TCVector;virtual; { <-- ф-ция - Умножение вектора
на комплексное число (действительное число - это
комлексное число с нулевой мнимой частью :
complex(dig,0) ) }

    function CVNormir:TCVector;virtual; { Нормиро-
вание вектора}
    function CVModul:TComplex;virtual; {Модуль век-
тора}
    Procedure Revers;virtual;   // Реверсирование
    published

    end;
    {Внешние ф-ции (директивой far избавимся от
написания методов в указаном порядке) }
    function CVOptimize(c:CVector):CVector;far; {
Отбрасывание ведущих нулей и приведение к 0 с точ-
ностью 1e-8}
    function CVSumma(v1,v2:CVector):CVector;far;
// Сумма векторов
    function CVRazn(v1,v2:CVector):CVector;far;
// Разность векторов
    function CVMult(v1,v2:CVector):CVector;far;
// Умножение векторов
```

```
    function CVScalarMult(v1,v2:CVector):
TComplex;far; // Скалярное произведение
    function CVEquiv(v1,v2:cvector):boolean;far;
// Сравнение векторов
    procedure MessageBox(s:string);far; {Просто со-
общение}
implementation


procedure TCVector.Revers;//Реверсирование вектора
var res:Cvector;
    i,j:integer;
begin
setlength(res,len);
for j:=0 to len-1 do res[j]:=vector[j];
for i:=0 to len-1 do vector[i]:=res[abs(len-i-1)];
end;

function CVOptimize(c:CVector):CVector; {Отбрасыва-
ние ведущих нулей и приведение к 0 с точностью 1e-
8}
var i:integer;res:CVector;
begin
 res:=c;
 for i:=0 to high(c) do begin
  if abs(res[i].re)<=1e-8 then res[i].re:=0;
  if abs(res[i].im)<=1e-8 then res[i].im:=0;
 end;
 result:=res;
end;

procedure MessageBox(s:string);
begin
MessageDlg(s,mtInformation,[mbOk],1); // Без слов
end;

function CVEquiv(v1,v2:cvector):boolean; {Сравнение
векторов}
var i:integer;
begin
result:=true;
 if high(v1)<>high(v2) then begin
result:=false;exit end;
```

```
 for i:=0 to high(v1) do
  if not CEquiv(v1[i],v2[i]) then result:=false;
end;

procedure TCVector.SetLen(l:integer);
begin
setlength(vector,l); {Устанавливаем вектору vector
длину l}
len:=l;
end;

function TCVector.CVMultOnDig(d:TComplex):TCVector;
var t:TCVector;i:integer;
begin
t:=TCVector.CVCreate(len);
for i:=0 to len-1 do
t.Vector[i]:=CMult(Vector[i],d);{ Вызов процедуры
произведения комплексных чисел (из модуля
T_Copmlex) }
result:=t;
end;

constructor TCVector.CVCreate(l:integer);
// Создаём вектор длины l
var i:integer;
begin
inherited Create; // Конструктор предка
setlen(l);        // Наша ф-ция установки длинны
for i:=0 to len-1 do vector[i]:=Complex(0,0);
// Сразу заполняем нулями
end;

function TCVector.CVModul:TComplex;//Модуль вектора
var tt:TComplex;i:integer;
begin
tt:=Complex(0,0);
{Возведение каждого элемента вектора в квадрат}
for i:=0 to len-1 do
tt:=CSumma(tt,CPow(vector[i],2));
                          {Сумма элементов вектора}
result:=CPow(tt,0.5); // SQRT(tt)
end;
```

```
function CVSumma(v1,v2:CVector):CVector;
//Сумма векторов
var i:integer;d:CVector;
begin
if high(v1)>high(v2) then begin
 setlength(d,high(v1)+1); { Устанавливаем размер по
наибольшей длине }
 for i:=0 to high(v2) do
 d[i]:=CSumma(v1[i],v2[i]); { Заносим в вектор-
результат значения суммы }
 for i:=high(v2)+1 to high(v1) do d[i]:=v1[i];
// Переписываем что осталось
end else
 begin
 setlength(d,high(v2)+1); { Устанавливаем размер по
наибольшей длине}
 for i:=0 to high(v1) do d[i]:=CSumma(v1[i],v2[i]);
// Заносим в вектор-результат значения суммы
 for i:=high(v1)+1 to high(v2) do d[i]:=v2[i];
// Переписываем что осталось
 end;
result:=d;
end;

function CVRazn(v1,v2:CVector):CVector;
// Разность векторов
var i:integer;d:CVector;
begin
// Аналогичные действия

if high(v1)>high(v2) then begin
 setlength(d,high(v1)+1);
 for i:=0 to high(v2) do d[i]:=CRazn(v1[i],v2[i]);
 for i:=high(v2)+1 to high(v1) do d[i]:=v1[i];
end else
 begin
 setlength(d,high(v2)+1);
 for i:=0 to high(v1) do d[i]:=CRazn(v1[i],v2[i]);
 for i:=high(v1)+1 to high(v2) do
d[i]:=CMultOnDig(v2[i],-1);
 end;
result:=d;
end;
```

```
function CVMult(v1,v2:CVector):CVector;
// Произведение векторов
var i:integer;d:CVector;
begin
setlength(d,high(v1)+1);
for i:=0 to high(v1) do d[i]:=CMult(v1[i],v2[i]);
result:=d;
end;

function CVScalarMult(v1,v2:CVector):TComplex;
var d:cvector;tt,n1,n2:tcomplex;i:integer;
    tc:TCVector;
begin
if high(v1)=high(v2) then begin
tc:=TCVector.CVCreate(high(v1)+1);
tt:=Complex(0,0);
d:=CVMult(v1,v2);
for i:=0 to high(d) do tt:=CSumma(tt,d[i]);
tc.vector:=v1;n1:=tc.CVModul;
tc.vector:=v2;n2:=tc.CVModul;
if (((n1.re=0) and (n1.im=0)) or((n2.re=0) and
(n2.im=0))) then begin
MessageBox('Один из векторов нулевой;(Модуль
T_CVector)');
result:=complex(0,0);
exit;
end;
result:=CDiv(tt,CMult(n1,n2));
tc.Free;
end else result:=complex(0,0);
exit;
end;

function TCVector.CVNormir:TCvector;
// Нормирование вектора
var t:TCVector;tt:TComplex;i:integer;
begin
t:=TCVector.CVCreate(len);
t.vector:=vector;
tt:=t.CVModul;
for i:=0 to len-1 do
```

```
 t.Vector[i]:=CDiv(t.Vector[i],tt);{ Деление каждо-
го элемента вектора на модуль вектора}
result:=t;
end;

//End file.
end.
```

14.2.4. Класс действительных матриц

```
//T_MATRIX.PAS


unit T_Matrix;

interface
uses math,Dialogs,T_Vector;
type

    TMatrix = class  // <-- Класс матриц
    Row,Col:integer; // <-- Кол-во срок и столбцов
    Matrix:array of DVector; {<-- Динамический мас-
сив векторов (сама матрица)}

    constructor MCreate(R,C:integer); {<-- Матрица
размерности R x C}
    constructor MCopy(m:TMatrix;var d:Tmatrix);
//<-- Конструктор копирования (иногда полезен)

    private

    protected

    public
    procedure SetDim(R,C:integer);virtual;{<--
Установка размерности R x C}
    function MMultOnDig(Dig:real):
TMatrix;virtual;//<-- Умножение матрицы на число
    function Minor(x,y:integer):
TMatrix;virtual;//<-- Минор
    function TRansp:TMatrix;virtual;
//<-- Транспонирование
    function Orto:TMatrix;virtual;
```

```
 //<-- Ортогонализация
    function Gauss:DVector;virtual;
 //<-- Нахождение корней методом Гаусса
    function MPow(s:integer):TMatrix;virtual;
 //<-- Возведение в целую степень
    function Det:real;virtual;
 //<-- Определитель матрицы
    function ObrMatrix:TMatrix;virtual;
 //<-- Обратная матрица
    published

    end;

function MSumma(m1,m2:TMatrix):TMatrix;far;
 //<-- Сумма матриц
function MMult(m1,m2:TMatrix):TMatrix;far;
 //<-- Произведение матриц
function MRazn(m1,m2:TMatrix):TMatrix;far;
 //<-- Разность матриц
function MEquiv(m1,m2:TMatrix):boolean;far;
 //<-- Сравнение матриц
procedure MessageBox(s:string);far;
 //<-- Вывод сообщения

implementation

constructor TMatrix.MCreate(R,C:integer);
 //<-- Создаем матрицу размерности R x C
begin
inherited Create;//<-- Конструктор родителя
SetDim(R,C);     {<-- Вызываем наш метод установки
размерности матриц}
end;

constructor TMatrix.MCopy(m:TMatrix;var d:TMatrix);
// Без слов
var i,j:integer;
begin
for i:=0 to row-1 do
for j:=0 to col-1 do
d.Matrix[i,j]:=m.matrix[i,j];
end;
```

```
procedure MessageBox(s:string); // Без слов
begin
MessageDlg(s,mtInformation,[mbOk],1);{<-- Стандарт-
ный метод вызова сообщений}
end;

procedure TMatrix.SetDim(R,C:integer); {Установка
размерности матрицы}
var i:integer;
begin
setlength(matrix,r); // Устанавливаем r столбцов
for i:=0 to r-1 do  SetLength(matrix[i],c); {Уста-
навливаем длину строк - C}
Row:=r;
Col:=c;
end;

function TMatrix.MMultOnDig(dig:real):TMatrix;
// Умножение матрицы на число
var i,j:integer;
    tp:TMatrix;
begin
tp:=TMatrix.MCreate(row,col);{<-- Создаём копию
объекта матрицы : row x col}
for i:=0 to row-1 do
for j:=0 to col-1 do
tp.Matrix[i,j]:=matrix[i,j]*dig;
result:=tp;
end;

function TMatrix.Minor(x,y:integer):TMatrix;
// Минор
var i,j,i1,j1:integer;
    tp:TMatrix;

begin
 tp:=TMatrix.MCreate(row,col);
 i1:=0;i:=0;
 while  i1<=row-1 do begin {<-- Перебираем все
строки}
  if i=x then i1:=i1+1; {<-- если нашли ту которую
надо вычеркнуть - переходим на следующую}
```

```
  if i1>row-1 then break; {<-- ограничения на раз-
мерность}
  for j:=0 to col-1 do
tp.Matrix[i,j]:=matrix[i1,j]; {<-- Переписываем
строки}
  i:=i+1;
  i1:=i1+1;
 end;
j1:=0;j:=0;
           //<-- Аналогично для столбцов
 while  j1<=col-1 do begin
  if j=y then j1:=j1+1;
    if j1>col-1 then break;
  for i:=0 to row-1 do
tp.Matrix[i,j]:=tp.matrix[i,j1];
  j:=j+1;
  j1:=j1+1;
 end;
 tp.SetDim(row-1,col-1);{Понижаем размерность мат-
рицы на 1}
 result:=tp;
end;

function TMatrix.TRansp:TMatrix;
// Транспонирование матрицы
var i,j:integer;
    res:TMatrix;
begin
res:=TMatrix.MCreate(row,col);
for i:=0 to row-1 do
for j:=0 to col-1 do
res.Matrix[i,j]:=Matrix[j,i]; // Aij меняем на Aji
result:=res;
end;

function TMatrix.Orto:TMatrix; {Ортогонализация
матрицы}
var
  s,l,dd,i,m,n:longint;
  ort:Tmatrix;
  md:real;
  d1:DVector;
```

```
begin
// Сумма квадратов 1-ой строки
setlength(d1,row*row);
 ort:=TMatrix.MCreate(row,col);
 Mcopy(self,ort);
  md:=0;
  m:=row;
  n:=col;
  for i:=0 to n-1 do
    md:=md+ort.matrix[0,i]*ort.matrix[0,i];
  md:=sqrt(md);
// Нормируем 1 строку
  for i:=0 to n-1 do
    ort.matrix[0,i]:=ort.matrix[0,i]/md;
 // Процесс повторяем 3 раза
  for dd:=0 to 2 do
  begin
    for l:=1 to m-1 do
    begin
      for i:=0 to l-1 do
      begin
      // Скалярное произведение l и i -ой строк
        md:=0;
        for s:=0 to n-1 do
          md:=md+ort.matrix[l,s]*ort.matrix[i,s];
     // Ортогонализируем l k i
        for s:=0 to n-1 do
          ort.matrix[l,s]:=ort.matrix[l,s]-
md*ort.matrix[i,s];
      end;
      //Нормируем l строку
      md:=0;
      for s:=0 to n-1 do
        md:=md+ort.matrix[l,s]*ort.matrix[l,s];
      md:=sqrt(md);
      for s:=0 to n-1 do
        ort.matrix[l,s]:=ort.matrix[l,s]/md;
    end;
  end;
 result:=ort;
end;

function TMatrix.Gauss:DVector; // Метод Гаусса
```

```
var
  i,j,k,num,m,n:longint;
mtr:TMatrix; res:dvector; temp:real;
begin
mtr:=TMatrix.MCreate(row,col);
MCopy(self,mtr);
m:=row;
n:=col;
setlength(res,m);
  for i:=0 to m-1 do
  begin
    temp:=mtr.matrix[0,i];  num:=i;
    for j:=i to m-1 do
      if(abs(mtr.matrix[j,i])>temp)then
      begin
        temp:=abs(mtr.matrix[j,i]); num:=j;
      end;
    if(num<>i)then
      for k:=0 to n-1 do
      begin
temp:=mtr.matrix[num,k];mtr.matrix[num,k]:=mtr.matr
ix[i,k];mtr.matrix[i,k]:=temp;
      end;
  end;
  for i:=0 to m-1 do
    if(mtr.matrix[i,i]=0) then
    begin
    MessageBox('матрица вырожденна');
    end;
  (*Прямой ход Гаусса*)
  for i:=0 to m-1 do
  begin
    temp:=mtr.matrix[i,i];
    for j:=0 to n-1 do
      mtr.matrix[i,j]:=mtr.matrix[i,j]/temp;
    for k:=i+1 to m-1 do
    begin
      temp:=mtr.matrix[k,i];
      for j:=0 to n-1 do
        mtr.matrix[k,j]:=mtr.matrix[k,j]-
mtr.matrix[i,j]*temp;
    end;
  end;
```

```
  (* Обратный ход Гаусса *)
  for i:=m-2 downto 0 do
  begin
    temp:=0;
    for j:=i+1 to m-1 do
      temp:=temp+mtr.matrix[i,j]*mtr.matrix[j,n-1];
    mtr.matrix[i,n-1]:=mtr.matrix[i,n-1]-temp;
  end;
  // Заносим результат в последнюю колонку
  for i:=0 to m-1 do
    res[i]:=mtr.matrix[i,n-1];
result:=res;
end;

Function TMatrix.MPow(s:integer):TMatrix; { Возве-
дение матрицы в целую неотрицательную степень }
var tp:TMatrix;
    i:integer;
begin
if row<>col then begin
MessageBox('Неквадратную матрицу возводить в сте-
пень нельзя');
result:=self;
exit;
end;
tp:=TMatrix.MCreate(row,col);
Mcopy(self,tp);
for i:=1 to s-1 do tp:=MMult(tp,self);
result:=tp;
exit;
end;

function MSumma(m1,m2:TMatrix):TMatrix; {Сумма мат-
риц}
var i,j:integer;
    res:TMatrix;
begin
if (m1.Row<>m2.Row) or (m1.Col<>m2.Col) then
begin
 MessageBox('Размерность матриц не совпадает');
 result:=TMatrix.MCreate(0,0);exit;
end;
res:=TMatrix.MCreate(m1.row,m1.col);
```

```
for i:=0 to m1.row-1 do
for j:=0 to m2.col-1 do
 res.Matrix[i,j]:=m1.Matrix[i,j]+m2.Matrix[i,j];
result:=res;
exit;
end;


function MRazn(m1,m2:TMatrix):TMatrix; {Разность
матриц}
var i,j:integer;
    res:TMatrix;
begin
if (m1.Row<>m2.Row) or (m1.Col<>m2.Col) then
begin MessageBox('Размерность матриц не совпада-
ет');result:=TMatrix.MCreate(0,0);exit;end;
res:=TMatrix.MCreate(m1.row,m1.col);
for i:=0 to m1.row-1 do
for j:=0 to m1.Col-1 do
 res.Matrix[i,j]:=m1.Matrix[i,j]-m2.Matrix[i,j];
result:=res;
exit;
end;

function MMult(m1,m2:TMatrix):TMatrix; {Произведе-
ние матриц}
var i,j,k:integer;s:real;
    res:TMatrix;
begin
if (m1.col<>m2.row) then
begin MessageBox('Размерность матриц не совпада-
ет');result:=TMatrix.MCreate(0,0);exit;end;
res:=TMatrix.MCreate(m1.row,m2.col);
for i:=0 to m1.row-1 do
 for j:=0 to m2.col-1 do begin
  s:=0;
  for k:=0 to m1.col-1 do
s:=s+m1.matrix[i,k]*m2.matrix[k,j];
  res.matrix[i,j]:=s;
 end;
result:=res;
exit;
end;
```

```
function TMatrix.Det:real; // Детерминант матрицы
var
  temp,sw,c,det,max:real;
  i,j,k,how,num,m:longint;
  mtr:Tmatrix;
begin

if row<>col then begin
 MessageBox('Для неквадратных матриц детерминант не
определён');
 result:=0;exit;
end;

mtr:=TMatrix.MCreate(row,col);
m:=row;
MCopy(self,mtr);
  case m of
    1:det:=mtr.matrix[0,0];
    2:det:=mtr.matrix[0,0]*mtr.matrix[1,1]-
    mtr.matrix[0,1]*mtr.matrix[1,0];
    else
{ Количество перестановок определяет знак детерми-
нанта }
      how:=0;
      for i:=0 to m-1 do
      begin
        max:=mtr.matrix[0,i];  num:=i;
        for j:=i to m-1 do
          if(abs(mtr.matrix[j,i])>max) then
          begin
            max:=abs(mtr.matrix[j,i]); num:=j;
          end;
        if(num<>i)then
        begin
          for k:=0 to m-1 do
          begin
          temp:=mtr.matrix[num,k];
          mtr.matrix[num,k]:=mtr.matrix[i,k];
          mtr.matrix[i,k]:=temp;
          end;
          inc(how);
        end;
```

```
      end;
      // Прямой ход Гаусса
      for i:=0 to m-1 do
      begin
        sw:=mtr.matrix[i,i];
        for j:=i+1 to m-1 do
mtr.matrix[i,j]:=mtr.matrix[i,j]/sw;
        for k:=i+1 to m-1 do
        begin
          c:=mtr.matrix[k,i];
for j:=0 to m-1 do
mtr.matrix[k,j]:=mtr.matrix[k,j]-mtr.matrix[i,j]*c;
        end;
      end;
      det:=1;
      for i:=0 to m-1 do det:=det*mtr.matrix[i,i];
      if(wordbool(how and 1)) then det:=-det;
  end;
result:=det;
end;

function MEquiv(m1,m2:TMatrix):boolean;
// Сравнение матриц
var i,j:integer;
begin
result:=true;
if (m1.Row<>m2.Row) or (m1.Col<>m2.Col) then begin
result:=false;exit;end;
for i:=0 to m1.row-1 do
for j:=0 to m2.col-1 do
if (m1.Matrix[i,j]<>m2.Matrix[i,j]) then begin
result:=false;exit;end;
end;

function TMatrix.ObrMatrix:TMatrix;
// Обратная матрица
var i,j:integer;
    tp,tp2:TMatrix;
    dd,d:real;
begin
if Det=0 then begin result:=self;
 messageBox('Матрица не является не особенной');
 exit;
```

```
end;

tp:=TMatrix.MCreate(row,Col);
tp2:=TMatrix.MCreate(row,Col);
MCopy(self,tp);
d:=det;
for i:=0 to col-1 do
for j:=0 to row-1 do begin
tp:=Minor(j,i);
dd:=tp.Det;
tp2.Matrix[j,i]:=Power(-1,i+j)*dd/d;
end;
result:=tp2;
tp.Free; // Убираем мусор из памяти
end;

//End file !!!!!
end.
```

14.2.5. Класс комплексных матриц

```
//T_CMATRIX.PAS


unit T_CMatrix;

interface
uses T_CVector,T_Complex;

type

    TCMatrix=class(TCVector) {<-- Класс комплексных
матриц (наследник от комплексных векторов)}
    Row,Col:integer; //<-- Кол-во строк и столбцов
    Matrix:array of CVector;   //<-- Сама матрица
    procedure SetDim(m,n:integer);
//<-- Установка размерности матрицы
    constructor CMCreate(m,n:integer);
//<-- Создание матрицы m x n

    private

    protected
```

```
    public
    {Внутренние ф-ции}
    function
CMultMatOnDig(c:TComplex):TCMatrix;virtual;
//<-- Умножение матрицы на комплексное число
    function CMTransp:TCMatrix;virtual;
//<-- Транспонирование матрицы

    published

    end;
    {Внешние ф-ции}
    function CMSumma(m1,m2:TCMatrix):TCMatrix;far;
//<-- Сумма матриц
    function CMRazn(m1,m2:TCMatrix):TCMatrix;far;
//<-- Разность матриц
    function CMMult(m1,m2:TCMatrix):TCMatrix;far;
//<-- Произведение матриц
    function CMEquiv(m1,m2:TCMatrix):Boolean;far;
//<-- Сравнение матриц

implementation

constructor TCMatrix.CMCreate(m,n:integer);
// Создание матрицы
var i,j:integer;
begin
 inherited Create; // Конструктор предка
 SetDim(m,n);
// Вызываем наш метод установки размерности матрицы
 for i:=0 to row-1 do
  for j:=0 to col-1 do matrix[i,j]:=complex(0,0);
// Сразу заносим нули
end;

procedure TCMatrix.SetDim(m,n:integer);
// Метод установки размерности матрицы
var i:integer;
begin
 setlength(matrix,m);
 for i:=0 to m-1 do setlength(matrix[i],n);
 row:=m;
```

```
 col:=n;
end;

function TCMatrix.CMultMatOnDig(c:TComplex):
TCMatrix; // Умножение матрицы на комплексное число
var i,j:integer;temp:TCMatrix;
begin
temp:=TCMatrix.CMCreate(row,col);
// Создаём экземпляр объекта TCMatrix
for i:=0 to row-1 do
for j:=0 to col-1 do
begin
 temp.matrix[i,j]:=CMult(matrix[i,j],c);
end;
result:=temp;
end;

function TCMatrix.CMTransp:TCMatrix;
// Транспонирование
var i,j:integer;temp:TCMatrix;
begin
temp:=TCMatrix.CMCreate(row,col);
for i:=0 to row-1 do
 for j:=0 to col-1 do
  temp.matrix[i,j]:=matrix[j,i];
result:=temp;
end;

function CMSumma(m1,m2:TCMatrix):TCMatrix;
var i:integer;temp:TCMatrix;
begin
if (m1.row<>m2.row) or (m1.col<>m2.col) then
// Если матрицы разных размеров
begin
 result:=TCMatrix.CMCreate(0,0);
// результат - матрица нулевых размеров.
 exit;
end;
// иначе
temp:=TCMatrix.CMCreate(m1.row,m1.col);
for i:=0 to m1.row-1 do

temp.matrix[i]:=CVSumma(m1.matrix[i],m2.matrix[i]);
```

```
result:=temp;
end;

function CMRazn(m1,m2:TCMatrix):TCMatrix;
var i:integer;temp:TCMatrix;
begin
if (m1.row<>m2.row) or (m1.col<>m2.col) then
// Если матрицы разных размеров
begin
 result:=TCMatrix.CMCreate(0,0);
// результат - матрица нулевых размеров
 exit;
end;
// иначе
temp:=TCMatrix.CMCreate(m1.row,m1.col);
for i:=0 to m1.row-1 do

temp.matrix[i]:=CVRazn(m1.matrix[i],m2.matrix[i]);
result:=temp;
end;

function CMMult(m1,m2:TCMatrix):TCMatrix;
var i,j,k:integer;temp:TCMatrix;
    s:TComplex;
begin
if (m1.col<>m2.row) then
// Если матрицы разных размеров
begin
 result:=TCMatrix.CMCreate(0,0);
// результат - матрица нулевых размеров
 exit;
end;
// иначе
temp:=TCMatrix.CMCreate(m1.row,m2.col);
for i:=0 to m1.row-1 do
 for j:=0 to m2.col-1 do begin
  s:=Complex(0,0);
  for k:=0 to m1.col-1 do
s:=CSumma(s,CMult(m1.matrix[i,k],m2.matrix[k,j]));
  temp.matrix[i,j]:=s;
 end;
result:=temp;
end;
```

```
function CMEquiv(m1,m2:TCMatrix):Boolean;
// Сравнение матриц
var i:integer;
begin
 result:=true;
 if (m1.row<>m2.row) or (m1.col<>m2.col) then begin
result:=false;exit end;
 for i:=0 to m1.row do
  if not CVEquiv(m1.matrix[i],m2.matrix[i]) then
result:=false; // Используем сравнение векторов
end;

// End File !!!!!
end.
```

14.2.6. Класс полиномов

```
//T_POLYNOM.PAS


unit T_Polynom;

interface
uses T_CVector,T_Complex,Math;

type
    TPolynom = class(TCVector) {<-- Класс полиномов
с операциями над ними <-- наследник от комплексных
векторов.}
    constructor PCreate(l:integer);
//<-- Конструктор.

    private

    protected

    public
    {Внутренние ф-ции}
    function GetFun(x:TComplex):TComplex;virtual;
//<-- Значение ф-ции в точке x
    function Derive(x:integer):TPolynom;virtual;
//<-- X-ая производная
```

```
    function Integral(x:integer):TPolynom;virtual;
//<-- Интеграл X-го порядка
    function PPow(x:integer):TPolynom;virtual;
//<-- Возведение в целую степень
    function PMultOnDig(x:TComplex):
TPolynom;virtual;//<--Умножение на число
    procedure Optimize;virtual;
//<-- Оптимизация (удаление нулевых старших членов)

    published

    end;
     // Внешние функции для работы с классом
    Function PMult(p1,p2:TPolynom):TPolynom;far;
//<-- Умножение полиномов
    Function PSumma(p1,p2:TPolynom):TPolynom;far;
//<-- сложение
    Function PRazn(p1,p2:TPolynom):TPolynom;far;
//<-- Разность
    procedure Evklid(p1,p2:TPolynom;var l,r:
TPolynom);far;
//<-- Метод Евклида для деления полиномов
    Function PIntOfDiv(p1,p2:TPolynom):
TPolynom;far;//<-- Целая часть от деления
    Function PMidOfDiv(p1,p2:TPolynom):
TPolynom;far; //<-- Остаток от деления
    Function PEquiv(p1,p2:TPolynom):Boolean;far;
//<-- Сравнение
    function Square(const d:array of TComplex):
CVector;far; //<-- Нахождение корней кв. ур-я
    function Kardano(const d:array of real):
CVector;far; //<-- Нахождение корней кубич. ур-я
    function Ferrary(const p:array of real):
CVector;far; //<-- Нахождение корней ур-я 4 степени
    function Newton(p:TPolynom):CVector;far;
//<-- Нахождение корней ур-я любой степени
    Function Root(dig:extended; st:integer) :
extended ; Far;
implementation

constructor TPolynom.PCreate(l:integer);
// Конструктор
begin
```

```
{По сути дела полином (в нашем случае)- это вектор
комплексных чисел , поэтому мы можем смело вызывать
конструктор для комплексных векторов.}
inherited CVCreate(l);
end;

Function Root(dig:extended; st:integer) : extended
; Far;
begin
if (dig<0) then
 if odd(st) then result:=-Power(abs(dig),1/st) else
exit;
if  dig>=0 then result:=power(dig,1/st);
end;

function TPolynom.GetFun(x:TComplex):TComplex;
//Значение ф-ции в точке x
var i:integer;t:TComplex;
begin
t:=Complex(0,0);
for i:=0 to len-1 do
t:=CSumma(t,CMult(vector[i],CPow(x,i)));
result:=t;
end;


function TPolynom.Derive(x:integer):TPolynom;
//X-ая производная
var i:integer;res:TPolynom;
begin
 res:=TPolynom.Pcreate(1);
 if x=0 then begin result:=self;exit end;
 if x<0 then begin result:=Integral(-x);exit end;
 if len<=1 then begin
res.setlen(0);result:=res;exit;end;
 res.setlen(len-1);
 for i:=0 to res.len-1 do
res.vector[i]:=CMultOnDig(vector[i+1],i+1);
 res.Optimize;
 if x=1 then result:=res else result:=res.Derive(x-
1);
end;
```

```
function TPolynom.Integral(x:integer):TPolynom;
//Интеграл X-го порядка
var res:TPolynom;i:integer;
begin
//Отрицательная степень трактуется как производная
 if x<=0 then begin result:=Derive(-x);exit end;
 res:=TPolynom.PCreate(len+1);
 for i:=0 to len-1 do
res.Vector[i+1]:=CMultOnDig(vector[i],1/(i+1));
 res.Optimize;
 if x=1 then result:=res else
result:=res.Integral(x-1);
end;

function TPolynom.PPow(x:integer):TPolynom;
//Возведение в целую степень
var temp:TPolynom;i:integer;
begin
 temp:=TPolynom.Pcreate(1);
 if x=0 then
    temp.Vector[0]:=complex(1,0) else begin
  temp:=self;
  for i:=0 to x-2 do temp:=PMult(temp,self);
 end;
// temp.Optimize;
 result:=temp;
end;

function TPolynom.PMultOnDig(x:TComplex):TPolynom;
//Умножение на комплексное число
var i:integer;
    t:TPolynom;
begin
 t:=TPolynom.PCreate(len);
 for i:=0 to len-1 do
t.vector[i]:=CMult(vector[i],x);
 t.Optimize;
 result:=t;
end;

procedure TPolynom.Optimize;
//Оптимизация (удаление нулевых старших членов)
var i:integer;
```

```
begin
 if len<2 then exit;
  // Ограничим точность до 1e-8
 for i:=0 to high(vector) do begin
 if abs(vector[i].re)<=1e-8 then vector[i].re:=0;
 if abs(vector[i].im)<=1e-8 then vector[i].im:=0;
 end;
 if (vector[len-1].re=0)and(vector[len-1].im=0)
then
     begin setlen(len-1);Optimize; end;
end;


Function PMult(p1,p2:TPolynom):TPolynom;
//Умножение полиномов
var i,j:integer;t:TPolynom;
begin
 t:=TPolynom.PCreate(p1.len+p2.len);
 for i:=0 to p1.len-1 do
  for j:=0 to p2.len-1 do

t.vector[i+j]:=CSumma(t.vector[i+j],CMult(p1.vector
[i],p2.vector[j]));
// t.Optimize;
 result:=t;
end;

Function PSumma(p1,p2:TPolynom):TPolynom;
// Сложение
var tt:CVector;pp:TPolynom;
begin
pp:=TPolynom.PCreate(1);
 tt:=CVSumma(p1.vector,p2.vector);
 pp.SetLen(high(tt)+1);
 pp.Vector:=tt;
 pp.Optimize;
 result:=pp;
end;

Function PRazn(p1,p2:TPolynom):TPolynom; //Разность
var tt:CVector;pp:TPolynom;
begin
pp:=TPolynom.PCreate(1);
```

```
 tt:=CVRazn(p1.vector,p2.vector);
 pp.SetLen(high(tt)+1);
 pp.Vector:=tt;
 pp.Optimize;
 result:=pp;
end;

//Алгоритм Евклида для деления полиномов
procedure Evklid(p1,p2:TPolynom;var l,r:TPolynom);
var tf,x,tg,pm:TPolynom;
begin
 tf:=TPolynom.PCreate(1);
 tg:=TPolynom.PCreate(1);
 x:=TPolynom.PCreate(2);
 pm:=TPolynom.PCreate(2);
 tf:=p1;tg:=p2;
 tf.Optimize;tg.Optimize;
 if tg.len=0 then exit;
{ если делитель больше делимого, то остаток- дели-
мое, результат - нуль-полином}
 if tf.len<tg.len then begin
  l.SetLen(0);
  r:=tf;exit;
 end;

 l.setlen(0);
 while tf.len>=tg.len do begin
  x.SetLen(2);
  x.vector[1]:=Complex(1,0);
  x:=x.PPow(tf.len-tg.len);
  x:=x.PMultOnDig(CDiv(tf.vector[tf.len-1],
tg.vector[tg.len-1]));
  pm:=PMult(tg,x);
  tf:=PRazn(tf,pm);
  r:=tf;
  l:=PSumma(l,x);
 end;
end;

//Целая часть от деления
Function PIntOfDiv(p1,p2:TPolynom):TPolynom;
var l,r:TPolynom;
begin
```

```
 l:=TPolynom.PCreate(1);
 r:=TPolynom.PCreate(1);
 Evklid(p1,p2,l,r);
 l.Optimize;
 result:=l;
end;

//Остаток от деления
Function PMidOfDiv(p1,p2:TPolynom):TPolynom;
var l,r:TPolynom;
begin
 l:=TPolynom.PCreate(1);
 r:=TPolynom.PCreate(1);
 Evklid(p1,p2,l,r);
 r.Optimize;
 result:=r;
end;

//Сравнение
Function PEquiv(p1,p2:TPolynom):Boolean;
begin
 result:=CVEquiv(p1.vector,p2.vector);
end;

//Нахождение корней кв. ур-я
function Square(const d:array of TComplex):CVector;
var res:CVector;a,b,c,dd:TComplex;
begin
  c:=d[0];b:=d[1];a:=d[2];
  setlength(res,2);
  if  (a.re=0) and (a.im=0) then begin
    res[0]:=CDiv(CMultOnDig(c,-1),b);
    res[1]:=res[0];
   end else begin
    dd:=CRazn(Cpow(b,2),CMultOnDig(CMult(a,c),4));
    res[0]:=CDiv(CRazn(CPow(dd,0.5),b),
CMultOnDig(a,2));
    res[1]:=CDiv(Crazn(CMultOnDig(CPow(dd,0.5),-
1),b),CMultOnDig(a,2))
   end;
   res:=CVoptimize(res);
   result:=res;
end;
```

```
//Нахождение корней кубич. ур-я по методу Кардано
function Kardano(const d:array of real):CVector;
var res,res2,zkubl2:CVector;
    a,b,c,p,q,diskr,what,u0,v0:real;
    cu0,cv0:TComplex;
    i:integer;
begin
 setlength(res,3);
 if d[3]=0 then begin
  res2:=Square([Complex(d[0],0),Complex(d[1],0),
Complex(d[2],0)]);
  res[0]:=res2[0];res[1]:=res[0];
  res[2]:=res2[1];
 end else begin
  a:=d[2]/d[3];b:=d[1]/d[3];c:=d[0]/d[3];
  p:=b-a*a/3;
  q:=2*a*a*a/27-a*b/3+c;
// Находим кубический дискриминант
  diskr:=power(q/2,2)+power(p/3,3);
  if diskr=0 then begin {Если дискриминант = 0 то
корни действительные, 2 из них одинаковые}
   res[0]:=Complex(3*q/p,0);
   res[1]:=CMultOnDig(res[0],-0.5);
   res[2]:=res[1];
  end;
  if diskr>0 then begin {1 корень действительный, 2
комплексных сопряженных}
   what:=-q/2+sqrt(diskr);
   u0:=root(what,3);
   v0:=-p/3/u0;
   res[0]:=Complex(u0+v0,0);
   res[1]:=CMultOnDig(res[0],-0.5);
   res[1].im:=(u0-v0)/2*sqrt(3);
   res[2]:=res[1];
   res[2].im:=-res[1].im;
  end;
  if diskr<0 then begin {3 действительных разных
корня}
   zkubl2:=square([complex(-p*p*p/27,0),
complex(q,0),complex(1,0)]);
   cu0:=CPow(zkubl2[0],1/3);
   res[0]:=Complex(2*cu0.re,0);
```

```
   res[1]:=Complex(-cu0.re-cu0.im*sqrt(3),0);
   res[2]:=Complex(-cu0.re+cu0.im*sqrt(3),0);
  end;
  for i:=0 to 2 do res[i].re:=res[i].re-a/3;
 end;
 res:=CVoptimize(res);
 result:=res;
end;

{Нахождение корней ур-я 4 степени по методу
Феррари}
function Ferrary(const p:array of real):CVector;
var res,res3,cy0,sq1,sq2:cvector;
    a,b,c,d,y0,aa,bb,xl2:real;
    i:integer;
begin
 setlength(res,4);
 if p[4]=0 then begin
  res3:=Kardano([p[0],p[1],p[2],p[3]]);
  res[0]:=res3[0];
  res[1]:=res[0];
  for i:=1 to 2 do res[i+1]:=res3[i];
 end else begin
  a:=p[3]/p[4];
  b:=p[2]/p[4];
  c:=p[1]/p[4];
  d:=p[0]/p[4];
  cy0:=Kardano([-d*a*a+4*b*d-c*c,-4*d+a*c,-b,1]);
  for i:=0 to 2 do
   if cy0[i].im=0 then begin
    y0:=cy0[i].re;
    break;
  end;
  aa:=a*a/4-b+y0;
  bb:=a*y0/2-c;
  xl2:=-bb/2/aa;

sq1:=Square([Complex(y0/2+sqrt(aa)*xl2,0),Complex(a
/2-sqrt(aa),0),Complex(1,0)]);
  sq2:=Square([Complex(y0/2-sqrt(aa)*xl2,0),
Complex(a/2+sqrt(aa),0),Complex(1,0)]);
  for i:=0 to 1 do begin
   res[i]:=sq1[i];
```

```
   res[i+2]:=sq2[i];
  end;
 end;
 res:=CVoptimize(res);
 result:=res;
end;

{Нахождение корней ур-я любой степени методом Нью-
тона}
function Newton(p:TPolynom):CVector;
const eps=1e-8; // Точность вычисления корней
var res,m:cvector;
    t:real;
    j,i:integer;
    z,dz,c1,c2:TComplex;
    z1:TPolynom;
begin
 t:=1;
 p.Optimize;
 setlength(res,p.len-1);
 if high(res)=0 then begin
  res[0]:=CDiv(CMultOnDig(p.vector[0],-1),
p.Vector[1]);
  result:=res;exit;
 end;
{Если полиномы 2,3,4 степени - корни находим преды-
дущими методами}
 if high(res)=1 then begin
  result:=square([p.vector[0],p.vector[1],
p.vector[2]]);
  exit;
 end;
 if (high(res)=2)and(p.Vector[0].im=0)and
(p.Vector[1].im=0)and(p.Vector[2].im=0)
  and(p.Vector[3].im=0) then begin
  result:=Kardano([p.Vector[0].re,p.Vector[1].re,
p.Vector[2].re,p.Vector[3].re]);
  exit;
 end;
 if (high(res)=3)and(p.Vector[0].im=0)and
(p.Vector[1].im=0)and(p.Vector[2].im=0)
  and(p.Vector[3].im=0)and(p.Vector[4].im=0) then
begin
```

```
  result:=Ferrary([p.Vector[0].re,p.Vector[1].re,
p.Vector[2].re, p.Vector[3].re,p.Vector[4].re]);
  exit;
 end;
 z:=Complex(0,0);
 repeat
  dz:=Complex(0,0);
  j:=1;
  while (dz.re=0) and (dz.im=0) do begin
   dz:=p.derive(j).Getfun(z);
   j:=j+1;
  end;
  j:=j-1;
  c1:=CMultOnDIg(CDiv(p.Getfun(z),dz),-1);
  c2:=CPow(c1,1/j);
  z:=CSumma(z,CMultOnDig(c2,t));
  t:=t*(1-eps);
//Цикл продолжается до достижения заданной точности
 until (abs(p.GetFun(z).re)<eps) and
(abs(p.GetFun(z).im)<eps);
 res[0]:=z;
 z1:=TPolynom.PCreate(2);
 z1.Vector[0]:=CMultOnDig(z,-1);
 z1.Vector[1]:=Complex(1,0);
 // Понижаем степень и находим следующий корень
 m:=Newton(PIntOfDiv(p,z1));
 for i:=1 to high(res) do res[i]:=m[i-1];
 z1.vector:=res;
 z1.Optimize;
 result:=z1.Vector;
end;

//End file.
end.
```

Литература

1. Алгоритмы и программы восстановления зависимостей. – М.: Наука, 1984.
2. Анго А. Математика для электро- и радиоинженеров. – М.: Наука, 1964.
3. Бабак В.П., Хандецький В.С., Шрюфер Е. Обробка сигналів. – К.: Либідь, 1996.
4. Боглаев Ю.П. Вычислительная математика и программирование: Учебное пособие для студентов втузов. – М.: Высшая школа, 1990.
5. Вентцель Д.А., Вентцель Е.С. Элементы теории приближенных вычислений. – М.: Издательство ВВИА им. Н.Е. Жуковского, 1949.
6. Винер Н. Кибернетика. – М.: Советское радио, 1968.
7. Гавурин М.К. Лекции по методам вычислений. – М.: Наука, 1971.
8. Гантмахер Ф.Р. Теория матриц. – М.: Государственное издательство технико-теоретической литературы, 1953.
9. Гринчишин Я.Т. TURBO PASCAL: Чисельні методи в фізиці та математиці: Навч. посібник. – Тернопіль, 1994.
10. Гроп Д. Методы идентификации систем. – М.: Мир, 1979.
11. Гутер Р.С., Овчинский Б.В. Элементы численного анализа и математической обработки результатов опыта. – М.: Наука, 1970.
12. Дейч А.М. Методы идентификации динамических объектов. – М.: Энергия, 1979.
13. Демидович Б.П., Марон И.А. Основы вычислительной математики. – М.: Физматгиз, 1963.
14. Демидович Б.П., Марон И.А., Шувалова Э.З. Численные методы анализа. – М.: Наука, 1967.
15. Джордж А., Лю Дж. Численное решение больших разряжённых систем уравнений. – М.: Мир, 1984.

16. Иванов В. В. Методы вычислений на ЭВМ: Справочное пособие. – К.: Наукова думка, 1986.
17. Иващенко Н.Н. Автоматическое регулирование. – М.: Машгиз, 1962.
18. Красовский А.А., Поспелов Г.С. Основы автоматики и технической кибернетики. – М.: Госэнергоиздат, 1962.
19. Красовский Н.Н. Теория управления движением. – М.: Наука, 1968.
20. Левинштейн М.Л. Операционное исчисление и его приложения к задачам электротехники. – М.: Энергия, 1964.
21. Мак-Кракен Д., Дорн У. Численные методы и программирование на Фортране. – М.: Мир, 1977.
22. Маликов В.Т., Кветный Р.Н. Вычислительные методы и применение ЭВМ: Учебное пособие. – К.: Вища школа, 1989.
23. Ортега Дж., Пул У. Введение в численные методы решения дифференциальных уравнений. – М.: Наука, 1986.
24. Перельмутер В.М., Соловьев А.К. Цифровые системы управления тиристорным электроприводом. – К.: Техніка, 1983.
25. Полищук А.П. Персональный компьютер и его программирование (С, С++, Паскаль). Учебно-справочное пособие. – Кривой Рог: КГПИ, 1997.
26. Полищук А.П., Семериков С.А. Методы вычислений в классах языка С++. Учебное пособие. – Кривой Рог: Издательский отдел КГПИ, 1999.
27. Попов В.С., Мансуров Н.Н., Николаев С.А. Электротехника. – М.: Издательство Министерства коммунального хозяйства РСФСР, 1958.
28. Уткіна С.В., Наришкіна Л.С. Алгебра і числові системи: Навч. посібник. – К.: Вища школа, 1995.
29. Фаддеева В.Н. Вычислительные методы линейной алгебры. – М.: Гостехиздат, 1950.
30. Файнштейн В.Г., Файнштейн Э.Г. Микропроцессорные

системы управления тиристорными электроприводами. – М.: Энергоатомиздат, 1986.
31. Фельдбаум А.А. Основы теории оптимальных систем. – М.: Наука, 1965.
32. Форсайт Д., Малькольм М., Моллер К. Машинные методы математических вычислений. – М.: Мир, 1980.
33. Форсайт Д., Моллер К. Численное решение систем линейных алгебраических уравнений. – М.: Мир, 1968.
34. Хемминг Р.В. Численные методы. – М.: Наука, 1968.
35. Шуп Т.Е. Прикладные численные методы в физике и технике. – М.: Высшая школа, 1990.
36. Шуп Т.Е. Решение инженерных задач на ЭВМ. Практическое руководство. – М.: Мир, 1982.
37. Электрические измерения. Средства и методы измерений (общий курс) под редакцией Е.Г. Шрамкова. – М.: Высшая школа, 1972.

Учебное пособие

Александр Павлович Полищук

Сергей Алексеевич Семериков

Автоматика

Подп. к печати 19.03.99 Формат 80х84 1/16.
Бумага офсетная №1 Усл. печ. л. 14,60
Усл. кр.-от. 14,59 Уч.-изд. л. 17,70
Тираж 500 Зак. №03-1914

КГПИ, 324086, Кривой Рог-86, пр. Гагарина, 54

Криворожская городская типография
324050, Кривой Рог-50, пр. Металлургов, 28.