Криворожский государственный педагогический институт
Кафедра информатики и прикладной математики

А.П. Полищук
С.А. Семериков

ЧИСЛЕННЫЕ МЕТОДЫ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ МЕТОДОЛОГИИ

Учебное пособие

Раздел 5: Дифференциальные уравнения

Символический метод для линейных уравнений

Одношаговые методы численного решения

Многошаговые методы

Кривой Рог

1998

Полищук А.П., Семериков С.А. Численные методы в объектно-ориентированной методологии. Раздел «Дифференциальные уравнения». Учебное пособие. – Кривой Рог: КГПИ, 1998. – 72 с.

Авторы:

Полищук А.П.	к. т. н., с. н. с., доцент кафедры информатики и прикладной математики.
Семериков С.А.	магистр математики.

Рецензенты:

Рашевский Н.А.	к. ф.-м. н., доцент кафедры математики (КГПИ)
Теплицкий И.А.	учитель-методист, зам. директора по научной работе (Центрально-Городская гимназия)

Под общей редакцией доктора физико-математических наук, профессора В.Н. Соловьёва.

Рекомендовано к печати на заседании кафедры информатики КГПИ, протокол №1 от 31.08.98 г.

Підп. до друку 14.09.98
Друк №3. Друк офсетний
Умовн. фарбо-відб. 3,45
Тираж 300

Формат 80x84 1/16.
Умовн. друк. арк. 3,6
Зам. №9-1470

КДПІ, 324086, Кривий Ріг-86, пр. Гагаріна, 54

Криворізька міська друкарня
324050, Кривий Ріг-50, пр. Металургів, 28.

Оглавление

5. Введение в численные методы решения дифференциальных уравнений

5.1. Обыкновенные дифференциальные уравнения (общие сведения)

Обыкновенным дифференциальным уравнением для функции $f(t)$ называется уравнение вида

$$F(t, f(t), f^{(1)}(t), \ldots, f^{(n)}(t))=0,$$

где F – заданная функция для бесконечного или конечного интервала t. ***Порядок уравнения*** определяется порядком n старшей производной $f^{(n)}(t)$ в этом уравнении. Уравнение называют ***линейным***, если функция F линейно зависит от всех своих аргументов:

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

где a_0, a_1, …, a_{n-1} – либо заданные постоянные коэффициенты, либо заданные функции t. Это уравнение можно рассматривать и в векторном варианте, когда $\boldsymbol{f}$ и $\boldsymbol{F}$ являются вектор-функциями и мы имеем дело не с одним уравнением, а с системой уравнений.

Для рассмотрения методов решения дифференциальных уравнений в принципе достаточно иметь метод решения системы уравнений первого порядка

$$\boldsymbol{f}^{(1)}(t)=\boldsymbol{F}(t,\boldsymbol{f}(t)),$$

где $\boldsymbol{f}$ и $\boldsymbol{F}$ - векторы с n координатами $F_1, F_2, \ldots, F_n, f_1, f_2, \ldots f_n$, поскольку уравнение n-го порядка сводится к системе n уравнений первого порядка, а систему m уравнений n-го порядка можно свести к системе nm уравнений первого порядка ***методом замены переменных***:

$$f_i(t)=f^{(i-1)}(t),\ i=1, \ldots, n.$$

В случае линейности системы уравнений относительно $\boldsymbol{f}(t)$ она принимает вид

$$\boldsymbol{f}^{(1)}(t)+\mathbf{A}\boldsymbol{f}(t)=\boldsymbol{b}(t),$$

где $\mathbf{A}$ – заданная матрица размера nxn, $\boldsymbol{b}$ – заданная вектор-функция от t.

Если элементы матрицы $\mathbf{A}$ не зависят от t и $\boldsymbol{b}=0$, то мы приходим к ***линейной однородной системе с постоянными коэффициентами***:

$$\boldsymbol{f}^{(1)}(t)+\mathbf{A}\boldsymbol{f}(t)=0,$$

решение которой можно получить явно в виде разложения

$$\boldsymbol{f}(t)=\boldsymbol{c}(\mathbf{E}+\mathbf{A}t+\mathbf{A}^2t^2/2+\ldots),$$

где $\boldsymbol{c}$ – произвольный постоянный вектор с n координатами. Разложение в скобках есть экспонента с показателем $\mathbf{A}t$ и решение можно записать в компактной форме

$$\boldsymbol{f}(t)=e^{\mathbf{A}t}\boldsymbol{c}.$$

Так как общее решение системы зависит от n произвольных постоянных c_i, то для определения конкретного единственного решения необходимо задать n дополнительных условий, которые обычно представляют собой начальные условия вида $\boldsymbol{f}(0)=\boldsymbol{f}_0$ или граничные условия вида $\boldsymbol{f}(a)=\boldsymbol{f}_a$ и, например, решение для системы однородных уравнений первого порядка с постоянными коэффициентами для заданных начальных условий (***задача Коши***) будет

$$\boldsymbol{f}(t)=\mathbf{e}^{\mathbf{A}t}\boldsymbol{f}_0.$$

Если дополнительные условия для функции или ее производной заданы не в одной точке диапазона, а в нескольких, то мы имеем дело с так называемой ***краевой задачей***, для которой ключевым является вопрос наличия и единственности решения.

В общем случае уравнения n-го порядка надо задать n условий для функции f и/или ее производных до n-1-го порядка, а для системы уравнений первого порядка надо задать p условий для функции в точке a и n-p условий в точке b.

5.2. Процессы как объект исследования и управления

В этом небольшом вступительном разделе мы кратко обсудим, откуда возникают дифференциальные уравнения и кому необходимо их решение. Этот вопрос не возникает при подготовке специалистов-прикладников, например инженерного профиля, – изучение прикладной области неизбежно сопровождается математическим описанием динамики изучаемых в ней процессов и дифференциальные уравнения воспринимаются как естественный инструмент исследования. Иначе обстоит дело при подготовке специалистов по общей и прикладной математике, так сказать «чистых математиков» - изучение аналитических и численных методов решения систем алгебраических, трансцендентных и дифференциальных уравнений часто воспринимается просто как «упражнения для ума» - преподавание ведется «чи-

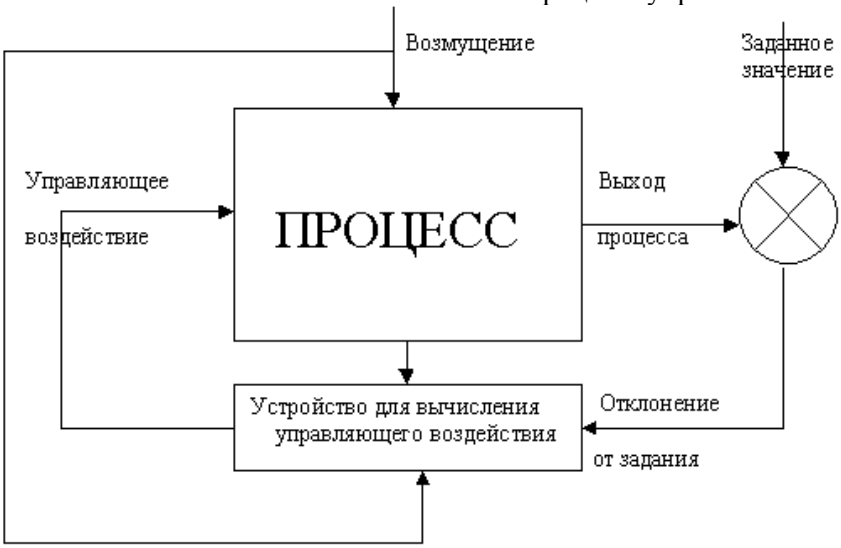
стыми» математиками и у студента зачастую остаются весьма смутные представления о прикладном значении изучаемых математических курсов (это особенно заметно в подготовке математиков в педагогических институтах). Если школьные задачи все же включают содержательную часть типа наполняемых жидкостями емкостей или движущихся в разных направлениях автомобилей, то студент-математик чаще всего просто получает задание решить конкретную систему уравнений, не представляя зачем и кому это может понадобиться (кроме необходимости получить оценку). Поэтому мы сочли необходимым хотя бы кратко остановиться на одном из прикладных аспектов изучаемого курса.

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.

По-видимому, одной из важнейших и труднейших областей интеллектуальной деятельности является управление процессами, которое 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.



На рисунке мы представили примерную общую схему системы управления процессом:

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.

С общей задачей управления связаны три основные ее составляющие:

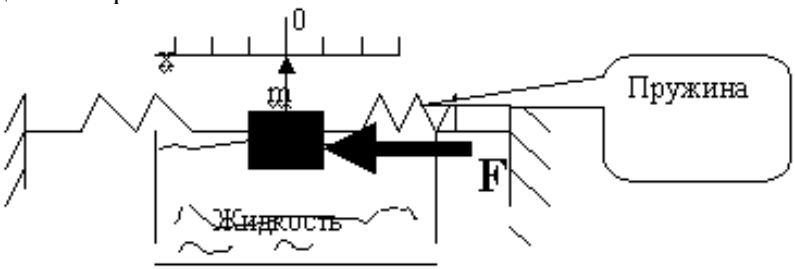
1) ***Задача идентификации*** математической модели управляемого процесса возникает, если математические соотношения, связывающие входы и выходы процесса неизвестны. Эти соотношения могут быть определены теоретически с использованием основных законов соответствующей прикладной области или экспериментально (что чаще всего и случается, по крайней мере, в технических системах). 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 - перемещение груза x. В соответствии с законами механики входное воздействие уравновешивается сопротивлением деформируемых пружин $k_n x$, возникающей во время движения силой вязкого трения $r\frac{dx}{dt}$ (r - коэффициент вязкого трения) и силой инерции $m\frac{d^2x}{dt^2}$ (m - масса тела).

Уравнение процесса движения тела имеет вид:

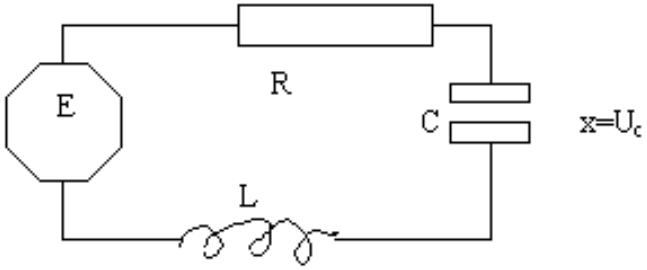
$$m\frac{d^2x}{dt^2}+r\frac{dx}{dt}+k_{\text{п}}x=F(t).$$

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:



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

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 $\mathrm{d}(m\boldsymbol{\upsilon})/\mathrm{d}t=\boldsymbol{F}$, где $m(t)$ - масса ракеты, изменяющаяся во времени из-за расхода топлива, $\boldsymbol{\upsilon}$ - 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:

$$x^{(2)}=(1/m)(T-0.5c\rho s\upsilon^2)\cos\theta-(m^{(1)}/m)x^{(1)},$$
$$y^{(2)}=(1/m)(T-0.5c\rho s\upsilon^2)\sin\theta-(m^{(1)}/m)y^{(1)}-g.$$

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:

$$x^{(2)}=(1/2m)(-0.5c\rho s\upsilon^2)\cos\theta,$$
$$y^{(2)}=(1/2m)(-0.5c\rho s\upsilon^2)\sin\theta-g \text{ при } \upsilon(0)=\upsilon_0,\ \theta(0)=\theta_0.$$

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 и какие-либо аналитические методы их решения неизвестны - приходится обращаться к численным методам.

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 полу-

бесконечном 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 произвольного (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 управле-

ния - 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 времени t=0 и отсутствует в предшествующие моменты, а для t=0 заданы значения выходной координаты и ее производных.

5.3. Операционное исчисление и его применение к исследованию динамики линейных систем

5.3.1. 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 $|f(t)|<Me^{\alpha t}$.

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

$$pF(p)\text{-}f(0)\subset f^{(1)}(t).$$

Если $f(0)=0$, то

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

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

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

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

Интегрирование f(t).

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

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

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

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

(суммирование по n от 1 до m).

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

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

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:

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

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

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

5.3.2. 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

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

Обозначим слагаемое, обусловленное ненулевыми начальными условиями

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

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 решение 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 $A_i=u(i\Delta\tau)\Delta\tau$. Если реакция системы на $Y^{(1)}$ есть $h^{(1)}(t)$, то реакция на i-й прямоугольный импульс будет приближенно равна $h^{(1)}(t-i\Delta\tau)u(i\Delta\tau)\Delta\tau$, 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 $f(t)=\int_0^{\tau} h^{(1)}(\tau)u(t-\tau)d\tau.$

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

$$a_2 \frac{d^2x}{dt^2} + a_1 \frac{dx}{dt} + a_0 x = ku,$$

приводят к передаточной функции вида

$$W(p)=\frac{k}{a_2 p^2 + a_1 p + a_0}.$$

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

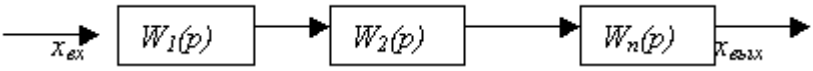
Уравнение интегрирующего звена $Tdx/dt=u$, его решение - $x=\int_0^t udt$, а передаточная функция $W(p)=1/Tp$.

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 $W(p)=a_2 p^2+a_1 p+1$, или дифференцирующее звено $W(p)=ap$.

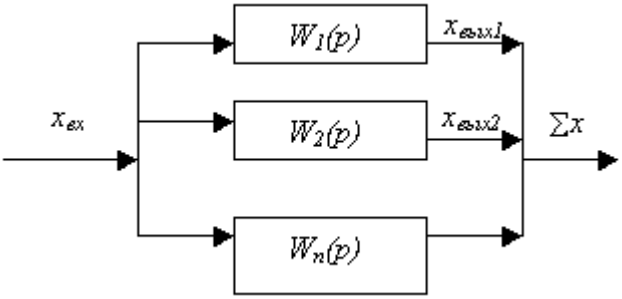
Звенья в системе могут соединяться ***последовательно***, ***параллельно*** и ***встречно-параллельно (обратной связью)***.

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



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

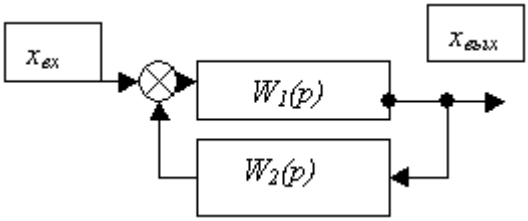
При параллельном соединении звеньев результирующая передаточная функция равна сумме передаточных функций входящих в соединение звеньев:



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

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

$$x_{вх1}=x_{вх}\pm x_{вых2}$$



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.

Частотные характеристики 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 си-

стему. 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 характеристика как разность фазовых углов: $\varphi(\omega)=\operatorname{arctg}\dfrac{I_Q(\omega)}{R_Q(\omega)}-\operatorname{arctg}\dfrac{I_p(\omega)}{R_p(\omega)}$.

Наличие базовых математических классов с набором основных операций (в частности класса полиномов и класса матриц) позволяет легко составить компактные программы для реализации приведенных вычислений, основные примеры которых приведены в программной реализации класса дифференциальных уравнений.

Фазовые портреты динамических систем

Еще один способ отображения динамических свойств си-

стем состоит в построении годографов в фазовом пространстве. Под ***фазовым пространством*** понимают пространство, образованное совокупностью вектор-функций решения уравнения системы и его производных. Так как графическое отображение возможно только на плоскости (максимум - псевдотрехмерное графическое построение), то обычно ограничиваются парой фазовых координат, например, «функция - ее первая производная». Годографы в такой координатной системе позволяют наглядно видеть взаимосвязь между фазовыми координатами в исследуемом временном интервале - при программной реализации мы построим фазовый портрет системы и рассмотрим его зависимость от различных параметров ее математической модели.

Ограничения области применения символического метода

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. к обыкновенным дифференциальным уравнениям или уравнениям 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 привести к получению решения, но не для первоначальной задачи. Поэтому разработка эффективных методов для решения нелинейных дифференциальных уравнений будет всегда актуальной вычислительной задачей.

5.3.3. Программная реализация класса символического метода

Состав программы

При составлении алгоритмов символического исчисления мы будем широко использовать объекты таких ранее созданных классов, как полиномы и векторы. Все данные и обслуживающие методы сосредоточим либо в заголовочном файле laplas.h, либо дадим в нем только интерфейсную часть, а реализационную перенесем в файл laplas.cpp и включим его в состав программного проекта - последний вариант даст возможность использовать однократную предварительную компиляцию заголовочного файла при отладке программы (если не использовать в этом файле процедуры инициализации данных). Кроме того, если в дальнейшем создавать из файла реализации библиотечный файл, то такое разделение на интерфейсную и реализационную части обязательно.

Результаты решения дифференциальных уравнений или расчета частотных характеристик, или портретов на фазовой плоскости - это функции, представляющие собой при компьютерном решении числовые массивы - одномерные или многомерные. Анализировать эти результаты непосредственным просмотром можно, но неудобно - гораздо удобнее анализировать их в графической форме в виде графиков функций. Кроме того, в учебном процессе желательно иметь возможность оперативно менять структуру уравнения и его коэффициенты и просматривать их влияние на характер переходных процессов - графические иллюстрации при этом значительно улучшают восприятие.

Поэтому в состав программы включен простейший вариант графического интерфейса пользователя для Windows, содержащий пользовательское окно с меню и простейшей панелью диалога для ввода исходных данных о дифференциальном уравнении и параметрах его решения - порядка уравнения и коэффициентов, начальных условий, свободных параметров стандартных возмущений (например, частоты гармонических воздействий), количества точек в дискретном представлении результатов решения, имени файла с дискретным вариантом произвольной возмущающей функции.

Наличие меню и диалоговой панели требует включения в программный проект, кроме файла основной программы laplas.cpp, файла ресурсов в скомпилированном или текстовом формате - у нас это будет текстовый файл laplas.rc.

О качестве программы. Это не более чем учебная программа - эскиз, ее не надо рассматривать как профессиональный образец; в частности, она не содержит достаточной защиты от неправильных действий пользователя - такая защита в несколько раз превысила бы размер содержательной части подпрограмм и за ее текстом алгоритмы решения было бы трудно рассмотреть. Она недостаточно структурирована, почти все используемые данные определены в глобальной области, чтобы не затруднять студента передачей аргументов, содержит другие недостатки, оставленные вполне осознанно с целью улучшить, сделать более прозрачной алгоритмическую часть вычислений.

Главная функция программы для Windows

/*Программа для решения ОЛДУ с постоянными коэффициентами символическим методом */

/*Включим в программу пока не рассмотренный нами файл с классом линейных дифференциальных уравнений, который будет содержать исходные данные и методы решения*/

```
#include "difequat.h"
```

/*Для графического отображения функций решения определим массив структур типа POINT, в который потом перенесем пересчитанные в

координаты значения отображаемой функции и значения ее аргумента */
```
POINT *pt;
```

/*Переменные для хранения размеров клиентской области окна вывода*/
```
int cxClient,cyClient;
```

//**Прототипы функций** окна, диалога и решения задач анализа
//Функция программного окна
```
LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
WPARAM wParam, LPARAM lParam);
```
//Функция обслуживания диалога с пользователем
```
LRESULT CALLBACK DlgProc(HWND hwnd, UINT uMsg,
WPARAM wParam, LPARAM lParam);
```

/*Функция для пересчета натуральных единиц в координаты конкретного окна - очевидно, что само рисование будет выполняться в оконной процедуре по запросам пользователя, который укажет, какой именно график он хочет вывести */
```
void getCoord(dvector y,dvector x)
```

/*Функция решения уравнений для свободного движения, различных стандртных возмущений и произвольной возмущающей функции*/
```
void solution(void);
```

```
void AmpFChar(); //Для расчета амплитудно-частотной характеристики
void PhazeFreqChar(); //Для расчета фазо-частотной характеристики
```

```
HINSTANCE hInst; //Для сохранения идентификатора приложения
```

```
#pragma argsused /*Чтобы не беспокоили сообщения о неиспользуемых аргументах */
```

//А теперь сама **главная Windows-функция**
```
int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
  hInst=hInstance;
  WNDCLASSEX wc; //Конструируем объект оконного класса
```

```
  //Определяем его параметры
  wc.cbSize=sizeof(wc);
  wc.style=CS_HREDRAW|CS_VREDRAW;
  wc.lpfnWndProc=(WNDPROC)WndProc;
  wc.cbClsExtra=0;
  wc.cbWndExtra=0;
  wc.hInstance=hInstance;
  wc.hIcon=(HICON)LoadIcon(0,IDI_WINLOGO);
  wc.hCursor=(HCURSOR)LoadCursor(0,IDC_ARROW);
wc.hbrBackground=(HBRUSH)GetStockObject(COLOR_BACKG
ROUND);
  wc.lpszMenuName="Laplas";
  wc.lpszClassName="Numerical Methods Class";
  wc.hIconSm=0;

  //Регистрируем в операционной системе
  if(!RegisterClassEx(&wc))
  {
    MessageBox(0,"Не удалось зарегистрировать класс
окна",0, MB_OK|MB_ICONEXCLAMATION);
    return 0;
  }

  //Создаем описание окна в памяти
  HWND hwnd=CreateWindowEx(
    WS_EX_CONTEXTHELP,
    "Numerical Methods Class",
    "Численные методы. Символический метод решения ОЛДУ",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    0,
    0,
    hInstance,
    0);
  //Вызываем стандартную пару функций для вывода окна
  ShowWindow(hwnd,nCmdShow);
  UpdateWindow(hwnd);
```

```
/*Здесь мы разместили оператор формирования в куче строки для имени файла с произвольной возмущающей функцией - ему надо бы найти лучшее место, но так уж получилось */
userfile=new char[250];

  //Запускаем цикл обработки сообщений
  MSG msg;
  while(GetMessage(&msg,0,0,0))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
/*Перед выходом из программы освобождаем память, занятую массивом структур для построения графиков (выделение этой памяти в подпрограмме рисования) и строкой для имени файла*/
  delete[] pt;
  delete[] userfile;
  return 0;
}
```

Файл описания ресурсов

Прежде чем перейти к оконной процедуре, приведем файл описания используемых нами ресурсов - меню и панели диалога. Это текстовый файл и он может быть прописан вручную или создан автоматически с помощью визуального редактора ресурсов. Для вызова редактора ресурсов достаточно щелкнуть мышью по имени файла описания ресурсов в окне проекта программы - он автоматически включается во вновь создаваемый проект. Можно комбинировать ручной и автоматизированный способы - последний особенно удобен при создании панелей диалога, чтобы не просчитывать координаты размещения органов управления.

Итак, файл laplas.rc.

Включаемый в него файл laplas.rh содержит числовую кодировку мнемоники команд с помощью директив #define - этот файл мы приведем ниже. Инструкции MENUITEM содержат надпись в пунктах меню и имя команды. Инструкции POPUP формируют в фигурных скобках выпадающие меню и команд не содержат.

Раздел описания ресурса типа "диалоговая панель" начинается тоже с ее имени DIALOG_1, описание стиля панели в целом, а затем описание размещенных на панели органов управления - у нас помимо неизбежной кнопки "ОК" для подачи команды завершения диалога присутствуют только статические надписи и однострочные редакторы текста для ввода исходных данных.

```
#include "laplas.rh"

Laplas MENU
{
  MENUITEM "Ввод исходных данных", CM_DATE
  POPUP "Решение уравнения"
  {
    MENUITEM "Свободное движение", CM_F
    MENUITEM "Реакци\377 на импульс", CM_IMP
    MENUITEM  "Реакци\377  на  ступенчатое  воздей-
ствие", CM_ST
    MENUITEM "Реакци\377 на синусоиду", CM_SIN
    MENUITEM "Реакци\377 на косинусоиду", CM_COS
    MENUITEM "Реакци\377 на экспоненту", CM_EXP
    MENUITEM  "Реакци\377  на  произвольное  воздей-
ствие", CM_SOL
  }

  POPUP "Частотные характеристики"
  {
    MENUITEM "Амплитудно-частотна\377", CM_AMP
    MENUITEM "Фазо-частотна\377", CM_PHAZE
    MENUITEM "Амплитудно-фазова\377", CM_AP
  }
  MENUITEM "Фазовый портрет", CM_PORTRET

  MENUITEM "Очистка области вывода", CM_CLEAR
}



DIALOG_1 DIALOG 0, 0, 240, 185
EXSTYLE WS_EX_DLGMODALFRAME | WS_EX_CONTEXTHELP
```

```
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP |
WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU |
WS_MINIMIZEBOX | WS_MAXIMIZEBOX

CAPTION "ВВОД ИСХОДНЫХ ДАННЫХ"
FONT 10, "MS Sans Serif"
{
  CONTROL "OK", IDOK, "BUTTON", BS_PUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE, 56, 148, 50, 14

  CONTROL "Порядок левой части уравнения", -1,
"static", SS_LEFT|WS_CHILD|WS_VISIBLE, 7, 5, 113, 8

  CONTROL "Коэффициенты левой части уравнения", -1,
"static", SS_LEFT|WS_CHILD|WS_VISIBLE, 6, 15, 138,
8

  CONTROL "Начальные условия", -1, "static",
SS_LEFT | WS_CHILD | WS_VISIBLE, 5, 25, 70, 8

  CONTROL "Порядок правой части уравнения", -1,
"static", SS_LEFT|WS_CHILD|WS_VISIBLE, 4, 36, 120,
8

  CONTROL "Коэффициенты правой части уравнения", -
1, "static", SS_LEFT|WS_CHILD|WS_VISIBLE, 3, 48,
141, 8

  CONTROL "ЧИСЛА РАЗДЕЛЯЙТЕ ТОЛЬКО ПРОБЕЛАМИ!", -1,
"static", SS_LEFT|WS_CHILD|WS_VISIBLE, 16, 132,
164, 8

  CONTROL "2", IDC_EDIT1, "edit", ES_LEFT |
WS_CHILD                                    |
ES_AUTOHSCROLL|WS_VISIBLE|WS_BORDER|WS_TABSTOP,
152, 4, 20, 9

  CONTROL "40 4 1", IDC_EDIT2, "edit", ES_LEFT |
ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_TABSTOP, 151, 15, 89, 9
```

```
  CONTROL "0  0", IDC_EDIT3, "edit", ES_LEFT |
ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_TABSTOP, 151, 26, 87, 9

  CONTROL "1", IDC_EDIT4, "edit", ES_LEFT |
ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_TABSTOP, 151, 36, 23, 9

  CONTROL "0  1", IDC_EDIT5, "edit", ES_LEFT |
ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_TABSTOP, 151, 47, 89, 9

  CONTROL "Частота гармонических возмущений", -1,
"static", SS_LEFT|WS_CHILD|WS_VISIBLE, 5, 61, 135,
8

  CONTROL "Показатель экспоненциального
воздействия", -1, "static",
SS_LEFT|WS_CHILD|WS_VISIBLE, 3,71,142,8

  CONTROL "Имя файла с произвольным возмущением", -
1, "static", SS_LEFT|WS_CHILD|WS_VISIBLE, 5, 82,
140, 8

  CONTROL "0.05", IDC_EDIT6, "edit", ES_LEFT |
ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_TABSTOP, 151, 60, 88, 9

  CONTROL "0.05", IDC_EDIT7, "edit", ES_LEFT |
ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_TABSTOP, 151, 71, 88, 8

  CONTROL "func.txt", IDC_EDIT8, "edit", ES_LEFT |
ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_TABSTOP, 151, 83, 89, 9

  CONTROL "Усиление на частоте среза", -1, "stat-
ic", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 97, 100, 7

  CONTROL "0.01", IDC_EDIT9, "edit", ES_LEFT |
ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_TABSTOP, 151, 95, 20, 9
```

```
  CONTROL "Количество точек решения", -1, "static",
SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 108, 115, 8, 0

  CONTROL  "200",  IDC_EDIT10,  "edit",  ES_LEFT  |
ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_TABSTOP, 151, 108, 20, 8, 0
}
```

Заголовочный файл для описания ресурсов

```
#define IDC_EDIT9       123
#define IDC_EDIT10      124
#define IDC_EDIT1       113
#define IDC_EDIT2       114
#define IDC_EDIT3       115
#define IDC_EDIT4       121
#define IDC_EDIT5       122
#define IDC_EDIT6       118
#define IDC_EDIT7       119
#define IDC_EDIT8       120
#define DIALOG_1        116
#define CM_F            101
#define CM_IMP          102
#define CM_ST           103
#define CM_SIN          104
#define CM_COS          105
#define CM_EXP          106
#define CM_SOL          107
#define CM_AMP          108
#define CM_PHAZE        109
#define CM_AP           110
#define CM_DATE         111
#define CM_CLEAR        112
#define CM_PORTRET      117
```

Оконная функция

Предназначена для обработки сообщений и выполняет у нас работу по отрисовке графиков функций.

```
HPEN hpen[12]; //Массив идентификаторов рисующих перьев

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
WPARAM wParam, LPARAM lParam)
```

```
{
  HDC hdc; //Для контекста отображения
  randomize();
  switch(uMsg)
  {
    case WM_SIZE: /*При получении этого сообщения определяем
размеры клиентской области окна */
    {
       cxClient=LOWORD(lParam);
       cyClient=HIWORD(lParam);
       return 0;
    }

/*В момент создания диалога создаем 12 перьев для рисования различ-
ными цветами - больше развлечение, чем необходимость */

    case WM_CREATE:
    {
      for(int i=0;i<12;i++)
      hpen[i]=CreatePen(PS_SOLID,5,rand());
      return 0;
    }
  }

  //Если поступила команда
  if(uMsg==WM_COMMAND)
  //Выделяем ее по старшему слову wParam
  switch(LOWORD(wParam))
  {
    //По этой команде выбора из меню выводится панель диалога
    case CM_DATE:
    {
      Dialog-
Box(hInst,MAKEINTRESOURCE(DIALOG_1),hwnd,
(DLGPROC)DlgProc);
      return 0;
    }
/*По этой команде очищается область рисования для подготовки к дру-
гому рисунку*/
    case CM_CLEAR:
    {
      InvalidateRect(hwnd,NULL,TRUE);
      return 0;
```

```
    }
```

/***Рисуем свободное движение.**
Все блоки рисования однотипны, они получают контекст через GetDC, вызывают функцию используют функцию getCoord для заполнения структуры типа POINT с координатами точек и для вычисления смещения координатных осей (мы приведем ее ниже), выбирают перо для рисования кривой, рисуют с помощью Безье сплайн-функции PolyBezier, рисуют оси, освобождают контекст. Здесь напрашивается подпрограмма с двумя аргументами - оформите ее сами, а мы не стали этого делать в стремлении к большей наглядности */

```
    case CM_F:
    {
      hdc=GetDC(hwnd);//Получаем контекст
      getCoord(liberty,tc);//Вычисляем массив координат
точек
      SelectObject(hdc,hpen[random(10)]);
      PolyBezier(hdc,pt,3*(((sc-4)/3)+1)+1);
      SelectObject(hdc,hpen[8]);
      //Вертикальная ось
      MoveToEx(hdc,fabs(off.x),0,NULL);
      LineTo(hdc,fabs(off.x),cyClient);
      //Горизонтальная ось
      MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
      LineTo(hdc,cxClient,cyClient-fabs(off.y));
      ReleaseDC(hwnd,hdc);
      return 0;
    }
```

//**Реакция на импульс**

```
    case CM_IMP:
    {
      getCoord(impuls,tc);  //Вычисляем массив координат
точек
      hdc=GetDC(hwnd);//Получаем контекст
      SelectObject(hdc,hpen[rand()]);
      PolyBezier(hdc,pt,3*(((sc-4)/3)+1)+1);
      SelectObject(hdc,hpen[8]);
      MoveToEx(hdc,fabs(off.x),0,NULL);
      LineTo(hdc,fabs(off.x),cyClient);
      MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
      LineTo(hdc,cxClient,cyClient-fabs(off.y));
```

```
        ReleaseDC(hwnd,hdc);
        return 0;
      }

      //Реакция на ступенчатое воздействие
      case CM_ST:
      {
        hdc=GetDC(hwnd);//Получаем контекст
        SelectObject(hdc,hpen[2]);
        getCoord(stupen,tc); //Вычисляем массив координат
точек
        PolyBezier(hdc,pt,3*(((sc-4)/3)+1)+1);
        SelectObject(hdc,hpen[8]);
        MoveToEx(hdc,fabs(off.x),0,NULL);
        LineTo(hdc,fabs(off.x),cyClient);
        MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
        LineTo(hdc,cxClient,cyClient-fabs(off.y));
        ReleaseDC(hwnd,hdc);
        return 0;
      }

      //Реакция на синусоидальное возмущение
      case CM_SIN:
      {
        getCoord(sinus,tc);
        hdc=GetDC(hwnd);//Получаем контекст
        SelectObject(hdc,hpen[3]);
        PolyBezier(hdc,pt,3*(((sc-4)/3)+1)+1);
        SelectObject(hdc,hpen[8]);
        MoveToEx(hdc,fabs(off.x),0,NULL);
        LineTo(hdc,fabs(off.x),cyClient);
        MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
        LineTo(hdc,cxClient,cyClient-fabs(off.y));
        ReleaseDC(hwnd,hdc);
        return 0;
      }

      //Реакция на вход вида косинусоиды
      case CM_COS:
      {
        hdc=GetDC(hwnd);//Получаем контекст
        SelectObject(hdc,hpen[4]);
```

```
      getCoord(cosinus,tc); //Вычисляем массив координат
точек
      PolyBezier(hdc,pt,3*(((sc-4)/3)+1)+1);
      SelectObject(hdc,hpen[8]);
      MoveToEx(hdc,fabs(off.x),0,NULL);
      LineTo(hdc,fabs(off.x),cyClient);
      MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
      LineTo(hdc,cxClient,cyClient-fabs(off.y));
      ReleaseDC(hwnd,hdc);
      return 0;
    }
```

//**Реакция на экспоненту на входе**

```
    case CM_EXP:
    {
      hdc=GetDC(hwnd);//Получаем контекст
      SelectObject(hdc,hpen[5]);
     getCoord(exponent,tc);//Вычисляем массив координат
точек
      PolyBezier(hdc,pt,3*(((sc-4)/3)+1)+1);
      SelectObject(hdc,hpen[8]);
      MoveToEx(hdc,fabs(off.x),0,NULL);
      LineTo(hdc,fabs(off.x),cyClient);
      MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
      LineTo(hdc,cxClient,cyClient-fabs(off.y));
      ReleaseDC(hwnd,hdc);
      return 0;
    }
```

//**Рисуется амплитудно-частотная характеристика**

```
    case CM_AMP:
    {
      hdc=GetDC(hwnd);//Получаем контекст
      getCoord(afch,tc);  //Вычисляем массив координат то-
чек
      SelectObject(hdc,hpen[6]);
      PolyBezier(hdc,pt,3*(((sc-4)/3)+1)+1);
      SelectObject(hdc,hpen[8]);
      MoveToEx(hdc,fabs(off.x),0,NULL);
      LineTo(hdc,fabs(off.x),cyClient);
      MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
      LineTo(hdc,cxClient,cyClient-fabs(off.y));
      ReleaseDC(hwnd,hdc);
```

```
      return 0;
    }
```

//**Фазо-частотная характеристика**

```
    case CM_PHAZE:
    {
      hdc=GetDC(hwnd);//Получаем контекст
      SelectObject(hdc,hpen[7]);
      //Вычисляем массив координат точек
      getCoord(ffch,3*(((sc-4)/3)+1)+1);
      PolyBezier(hdc,pt,sc-1);
      SelectObject(hdc,hpen[8]);
      MoveToEx(hdc,fabs(off.x),0,NULL);
      LineTo(hdc,fabs(off.x),cyClient);
      MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
      LineTo(hdc,cxClient,cyClient-fabs(off.y));
      ReleaseDC(hwnd,hdc);
      return 0;
    }
```

//**Аплитудно-фазовая характеристика**

```
    case CM_AP:
    {
      hdc=GetDC(hwnd);//Получаем контекст
      SelectObject(hdc,hpen[9]);
      //Вычисляем массив координат точек
      getCoord(afcm[1],afcm[0]);
      PolyBezier(hdc,pt,3*(((sc-4)/3)+1)+1);
      SelectObject(hdc,hpen[8]);
      MoveToEx(hdc,fabs(off.x),0,NULL);
      LineTo(hdc,fabs(off.x),cyClient);
      MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
      LineTo(hdc,cxClient,cyClient-fabs(off.y));
      ReleaseDC(hwnd,hdc);
      return 0;
    }
```

//**Рисуется реакция на произвольное возмущение на входе**

```
    case CM_SOL:
    {
      hdc=GetDC(hwnd);//Получаем контекст
      SelectObject(hdc,hpen[10]);
      //Вычисляем массив координат точек
```

```
      getCoord(fnc,tc);
      PolyBezier(hdc,pt,3*(((sc-4)/3)+1)+1);
      SelectObject(hdc,hpen[8]);
      MoveToEx(hdc,fabs(off.x),0,NULL);
      LineTo(hdc,fabs(off.x),cyClient);
      MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
      LineTo(hdc,cxClient,cyClient-fabs(off.y));
      ReleaseDC(hwnd,hdc);
      return 0;
    }
```

//**Фазовый портрет в координатах «функция-производная»**

```
    case CM_PORTRET:
    {
      hdc=GetDC(hwnd);//Получаем контекст
      SelectObject(hdc,hpen[9]);
      //Вычисляем массив координат точек
      getCoord(portret[0],portret[1]);
      PolyBezier(hdc,pt,3*(((sc-4)/3)+1)+1);
      SelectObject(hdc,hpen[8]);
      MoveToEx(hdc,fabs(off.x),0,NULL);
      LineTo(hdc,fabs(off.x),cyClient);
      MoveToEx(hdc,0,cyClient-fabs(off.y),NULL);
      LineTo(hdc,cxClient,cyClient-fabs(off.y));
      ReleaseDC(hwnd,hdc);
      return 0;
    }
  }


  //Если окно надо закрыть
  if(uMsg==WM_DESTROY)
  {
    PostQuitMessage(0);
    return 0;
  }
  return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
```

/*Теперь обещанная функция для пересчета натуральных единиц подлежащих рисованию значений функций в значения координат в клиентской области окна getCoord - в качестве аргументов ей понадобятся

контекст устройства и ссылка на массив значений функции. Прежде всего подпрограмма должна вычислить диапазон изменения значений функции и количество пикселей на одну натуральную единицу по обеим осям. */

```
// Структура для смещений координатных осей
struct OFF{long x,y;}off;
void getCoord(dvector y,dvector x)
{
  long i;
```

/*Определим наибольшее, наименьшее значение функции, диапазон изменения - хотя было бы неплохо получать эти значения из класса vector - будет хорошо, если вы его дополните и используете здесь соответствующий метод */

```
  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 pt точками для рисования и на этом черновая работа завершена */
  if(fxmax!=fxmin && fymax!=fymin)
  {
    for(i=0;i<sc;i++)
      pt[i].x=x[i]*(double(cxClient)/(fxmax-
fxmin));
    for(i=0;i<sc;i++)
      pt[i].y=cyClient-long(y[i]*(double(cyClient)/
              (fymax-fymin)));
  }
  if(fxmax-fxmin)
    off.x=fxmin*((double)cxClient/(fxmax-fxmin));
```

```
  if(fymax-fymin)
    off.y=fymin*((double)cyClient/(fymax-fymin));
}
```

Функция обслуживания диалога с пользователем

```
#pragma argsused
LRESULT CALLBACK DlgProc(HWND hdlg, UINT uMsg,
WPARAM wParam, LPARAM lParam)
{
  int i;
  //Выделим промежуточный буфер для приема ввода пользователя
  char* buf=new char[250];
  //И указатель для адреса слова, возвращаемого функцией strtok
  //при разборке строк
  char*tmp;

  switch (uMsg)
  {
    case WM_INITDIALOG:
      return TRUE; //При инициализации диалога

    case WM_COMMAND:
      switch(wParam)
      {
        case IDOK: //Если пользователь завершил ввод и нажал
                   //кнопку "ОК"
        //Получаем введенные строки
        BOOL err; //Для обнаружения ошибки в общении со строками
                  // ввода
//Получаем и сразу преобразуем в целое количество точек в решении

sc=GetDlgItemInt(hdlg,IDC_EDIT10,&err,FALSE);

/*После получения количества точек определяются размеры массивов для хранения информации, в том числе:*/

        pt=new POINT[sc]; //координат точек кривой
        {
          dvector vt(sc);
          /*реакций на все возмущения и всех характеристик*/
          tc=func=liberty=dliberty=dliberty=impuls=
```

```
          dim-
puls=stupen=dstupen=sinus=dsinus=cosinus=
          dcosi-
nus=exponent=dexponent=fnc=dfnc=afch=
          ffch=vt;
        }
        {
          dmatrix mt(2,sc);
          //то же для двумерных кривых - матрицы
          afcm=portret=mt;
        }
        //Получаем и преобразуем в целое порядок уравнения слева

rng=GetDlgItemInt(hdlg,IDC_EDIT1,&err,FALSE);

/*После определения порядка - корректируем размерность векторов */
        cfn=dvector(rng+1);
        nusl=dvector(rng);

/*Получаем символьное представление коэффициентов и начальных
условий */

SendDlgItemMessage(hdlg,IDC_EDIT2,EM_GETLINE,

(WPARAM)0,(LPARAM)(LPSTR)buf);

        //Выполням разборку строки и заполнение вектора cfn
        tmp=strtok(buf," ");
        for(i=0;tmp!=NULL;i++)
          cfn[i]=atof(tmp), tmp=strtok(NULL," ");
        SendDlgIteMesage(hdlg,IDC_EDIT3,EM_GETLINE,

(WPARAM)0,(LPARAM)(LPSTR)buf);
        tmp=strtok(buf," ");
        for(i=0;tmp!=NULL;i++)
          nusl[i]=atof(tmp), tmp=strtok(NULL," ");

        //Порядок справа

rngr=GetDlgItemInt(hdlg,IDC_EDIT4,&err,FALSE);

/*Корректируем вектор правой части уравнения по полученной его
размерности*/
```

```
        cfnr=dvector(rngr+1);
        //Символьный формат коэффициентов справа

SendDlgItemMessage(hdlg,IDC_EDIT5,EM_GETLINE,

(WPARAM)0,(LPARAM)(LPSTR)buf);

        tmp=strtok(buf," ");
        for(i=0;tmp!=NULL;i++)
          cfnr[i]=atof(tmp), tmp=strtok(NULL," ");

        //Частота гармонических возмущений

SendDlgItemMessage(hdlg,IDC_EDIT6,EM_GETLINE,

(WPARAM)0,(LPARAM)(LPSTR)buf);
        freq=atof(buf);
        //Показатель степени экспоненциального воздействия
        SendDlgItemMessage(hdlg,IDC_EDIT7,
EM_GETLINE,
                           (WPARAM)0,
(LPARAM)(LPSTR)buf);
        sexp=atof(buf);

        //Имя файла с произвольным возмущением
        SendDlgItemMessage(hdlg,IDC_EDIT8,
EM_GETLINE,
                     (WPARAM)0,
(LPARAM)(LPSTR)userfile);
        //Коэффициент усиления на частоте среза
        SendDlgItemMessage(hdlg,IDC_EDIT9,
EM_GETLINE,
                           (WPARAM)0,
(LPARAM)(LPSTR)buf);
        cufz=atof(buf);

        delete[] buf;//Освобождаем уже ненужный бу-
фер
```

/*После приема ввода пользователя и его преобразования вызываем подпрограмму решения стандартных задач - она приведена ниже*/

```
        solution();
```

```
        //И выходим
        EndDialog(hdlg,0);
        return TRUE;
      }
  }
  return FALSE;
}
```

Функция для решения связанных с дифуравнением задач

```
void solution(void)
{
  double i;
```
/*За неимением в демонстрационном варианте конкретных данных о значениях аргумента функций решения заполним его числами натурального ряда*/
```
  for(i=0;i<double(sc);i++) tc[i]=i;
```

/*Методы для выполнения необходимых расчетов мы определили в заголовочном файле laplas.h (приводится ниже). Всю диспетчерскую работу мы возложим на функцию с неудачным названием FreeSolution и теперь просто вызываем ее в нашей программе, указывая код стандартной входной функции и и порядок вычисляемой производной (0 - для функции, мы вычисляем здесь только первые производные, хотя запрограммировали возможность вычисления для любого осмысленного порядка)*/

/***Свободное движение при заданных (ненулевых) начальных условиях.***
Чтобы обеспечить возможность графической интерпретации результатов с помощью других инструментальных средств (например, в Excel) будем сохранять их в файлах с фиксированными именами */

```
  ofstream os0("lyberty.txt"),os0d("dliberty.txt");
```
/*Для свободного движения вычислим функцию и ее производную. Аргумент - порядок производной. */
```
  liberty=FreeSolution(0);
  dliberty=FreeSolution(1);
  os0<<liberty;os0.close();
  os0d<<dliberty;os0d.close();
```

/***Реакция на входной импульс.**
Второй аргумент FreeSolution- код импульсного воздействия*/

```
  ofstream os1("impuls.txt"),os1d("dimpuls.txt");
  impuls=FreeSolution(0,1);
  dimpuls=FreeSolution(1,1);
  os1<<impuls;os1.close();
  os1d<<dimpuls;os1d.close();
```

//Реакция на ступенчатый вход

```
  ofstream os2("stupen.txt"),os2d("dstupen.txt");
  stupen=FreeSolution(0,2);
  dstupen=FreeSolution(1,2);
  os2<<stupen; os2.close();
  os2d<<dstupen; os2d.close();
```

//Реакция на синусоидальный вход

```
  ofstream os3("sinus.txt"),os3d("dsinus.txt");
  sinus=FreeSolution(0,3);
  dsinus=FreeSolution(1,3);
  os3<<sinus; os3.close();
  os3d<<dsinus; os3d.close();
```

//Реакция на косинусоидальный вход

```
  ofstream os4("cosinus.txt"),os4d("dcosinus.txt");
  cosinus=FreeSolution(0,4);
  dcosinus=FreeSolution(1,4);
  os4<<cosinus; os4.close();
  os4d<<dcosinus; os4d.close();
```

//Реакция на экспоненциальный вход

```
  ofstream
os5("exponent.txt"),os5d("dexponent.txt");
  exponent=FreeSolution(0,5);
  dexponent=FreeSolution(1,5);
  os5<<exponent; os5.close();
  os5d<<dexponent; os5d.close();
```

//Реакция на произвольный вход

/*Чтобы иметь что-нибудь по умолчанию, сгенерируем этот вход для тестирования и запишем в файл с именем, заданным нами по умолчанию в панели диалога */

```
  for(int i=0;i<sc;i++)
    func[i]=exp(-0.02*i*(200/sc))*
```

```
            cos(i*(200/sc)*(2*M_PI/sc));
  ofstream ouf("func.txt");
  ouf<<func;ouf.close();
  //Прочтем функцию из файла пользователя
  if(strlen(userfile))
  {
    ifstream iuf(userfile);
    iuf>>func;iuf.close();
  }
  //Затем вычислим и запишем в файл реакцию на него
  ofstream os7("function.txt");
  fnc=FreeSolution(0,6);
  os7<<fnc;os7.close();


  //Амплитудно-частотная характеристика
  ofstream af("afchar.txt");
  AmpFChar();
  af<<afch; af.close();

  //Фазо-частотная характеристика
  ofstream fc("phazefchr.txt");
  PhazeFreqChar();
  fc<<ffch; fc.close();
```

/***Амплитудно-фазовая характеристика** состоит из двух координат, которые вычисляются по известным амплитудно-частотной и фазо-частотной характеристикам умножением значений амплитудно-частотной на косинус и синус фазо-частотной. Мы не составляли для ее вычисления отдельной подпрограммы из-за ее простоты. */

```
  ofstream ac("ampfchr.txt");
  for(i=0;i<sc;i++)
    afcm[0][i]=afch[i]*cos(ffch[i]),
    afcm[1][i]=afch[i]*sin(ffch[i]);
  ac<<afcm;ac.close();
```

/*Фазовые портреты объекта в координатах функция-первая производная при свободном движении и различных коэффициентах */

```
  portret[1]=FreeSolution(0);
  portret[0]=FreeSolution(1);
}
```

Файл включения difequat.h с объявлением класса линейных дифференциальных уравнений и базирующихся на преобразовании Лапласа подпрограмм для их решения.

В этом файле сосредоточены объявления большей части используемых в программе переменных, ядра программы - класса DifferentialEquation, и реализации всех методов решения. В профессиональной программе надо бы разделить интерфейсную и реализационную части, но мы знаем, как трудно студенту блуждать по разным файлам для сопоставления интерфейса и реализации и оставили все «под рукой».

Все включаемые файлы в кавычках с классами векторов, полиномов, матриц и файл с методом Ньютона для вычисления комплексных корней полиномов вам уже знакомы по предыдущему материалу пособия и здесь не обсуждаются.

```
#include <windows.h>
#include <alloc.h>
#include "vector.h"
#include "equation.h"
#include "wlaplas2.rh"
#include "matrix.h"


/*Для упрощения записи конкретизируем типы векторных и матричных элементов */
typedef vector<double> dvector;
typedef matrix<double> dmatrix;

long sc;//Для количества дискретных точек решения

//Для порядка уравнения в левой и правой части
int rng,rngr;

//Для частоты гармонических возмущений и степени экспоненциального
double freq, sexp;

//Для кратности снижения амплитуды на частоте среза
double cufz;

/*Массив для значений аргументов функций */
```

```
dvector tc;

char* userfile;//Имя введенного пользователем файла

/*Для коэффициентов уравнения в левой, правой части и начальных условий */
dvector cfn,cfnr,nusl;

/*Для задания произвольной входной функции тоже отведем вещественный массив значений - заполнить его можно, например, из файла. Результаты решения тоже удобнее всего для последующего анализа сохранять в дисковых файлах */

dvector func;

/*После ввода пользователя мы собираемся решить набор связанных с ОЛДУ типовых задач, в том числе - вычисление свободного движения при ненулевых начальных условиях и реакций на стандартные возмущения. Для хранения результатов решения нам понадобятся векторы; их размер - в заполняемой пользователем переменной sc и пока нам неизвестен, поэтому мы определяем их минимальной размерности, которую расширим в функции диалога после завершения ввода пользователем - см. выше */

dvector //это вектор с элементами типа double
liberty, //для свободного движения при отсутствии входа
dliberty, //для производной
impuls, //для реакции на входной импульс
dimpuls,
stupen, //для реакции на ступенчатый вход
dstupen,
sinus, //для реакции на входную синусоиду
dsinus,
cosinus, //для реакции на входную косинусоиду
dcosinus,
exponent, //для реакции на входную экспоненту
dexponent,
fnc, //для реакции на произвольную входную функцию
dfnc,
afch, //для амплитудно-частотной характеристики
ffch; //для фазо-частотной характеристики
```

```
dmatrix
portret, /*матрица значений функции и ее первой производной при свободном движении */
afcm; /*матрица из двух строк для амплитудно-фазовой частотной характеристики */
```

Класс обыкновенных линейных дифференциальных уравнений (ОЛДУ), содержащий методы, базирующиеся на операционном исчислении

```
class DifferentialEquation
{
  //приватные данные
  dvector  rfunc; //вектор значений функции в правой части ОЛДУ
  cpolynom P; //Характеристический полином
  cpolynom PR; //Полином правой части уравнения
  cvector polyroot; //корни характеристического полинома
  cpolynom Q; //полином начальных условий
  cpolynom chislitel, //Полином числителя
        znamenatel;  //и знаменателя в изображении решения ОЛДУ
  cpolynom CH,ZN;
/*Сведения о корнях характеристического полинома сгруппируем в структуре, а для всех корней придется разместить в памяти массив таких структур - быть может не лучшее решение, но оно позволяет лучше ориентироваться в обозначениях */

  struct CA
  {
    complex A; //значение корня
    complex AO; //коэффициент для корня в оригинале
    long J; //кратность
    long O; //кому кратен
  };
  CA * R; //указатель на структуру

  long coeff_count, //Количество коэффициентов в уравнении
  root_count, //Количество корней всего
  rcount; //Количество различных корней
  double maxW, //Частота среза
```

```
  stepW; //Количество шагов по частоте для частотных характеристик

 public: //общедоступные данные и функции

/*Конструирование по данным пользователя */
  DifferentialEquation(cpolynom& co, cpolynom& cr,
                       cpolynom& st);

//Конструктор копирования
  DifferentialEquation(DifferentialEquation &ex);

  //Деструктор
  ~DifferentialEquation() {farfree(R);}

  //Набор функций, возвращающих элементы исходных данных
  complex &GetCoeff(long index) { return P[index];
}
  complex    &GetStartcond(long    index)    {return
Q[index];}
  double &GetFunction(long idx){return rfunc[idx];}

/*Функция, вычисляющая коэффициенты оригинала для всех корней результат помещается в массив структур R*/
  void GetCoeffOrigin();
  //Функция для вычисления корней характеристического полинома
  void GetRoot();
/*Функция, вычисляющая решение неоднородного ОЛДУ при стандартных функциях времени в правой части - свободное движение без правой части (variant=0), импульсная функция (variant=1),ступенька (variant=2), синусоида (variant=3), косинусоида (variant=4), экспонента (variant=5). У синусоиды param это частота и т.д. */

  void StandardSolution(long variant);
/*Функция - друг класса, вычисляющая реакции на произвольные воздействия, заданные либо числовым массивом вещественных значений, либо является стандартной. */
  friend dvector FreeSolution (long dnumber,
  //порядок вычисляемой производной решения ОЛДУ
  long variant=0); //код стандартного возмущения

//Функция, возвращающая количество коэффициентов в уравнении
  long getn() { return P.getm(); }
```

```
/*Функция, возвращающая значение выхода при заданном значении аргумента */
  double GetValue(double t);
};


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

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

Определения объявленных функций

```
//Конструктор по данным пользователя
DifferentialEquation::DifferentialEquation
(cpolynom& co,cpolynom& cr, cpolynom& st)
{
  P=co; PR=cr; coeff_count=P.getm();
  int i;

  //Определяем количество корней
  root_count=coeff_count-1;

//Выделяем память для массива структур со сведениями о корнях
  R=(CA*)farcalloc(root_count,sizeof(CA));
  memset(R,0,root_count*sizeof(CA));
```

/***Формирование полинома начальных условий.***
Здесь мы просто программируем вычисление по приведенной в теоретическом разделе формуле $\sum_{i=0}^{n-1}\left[f^{(i)}(0)\sum_{j=i+1}^{n} a_j p^{j-1} \right]$. */

```
  complex tt[]={complex(0,0),complex(1,0)};
```

```
  cpolynom d(2,tt), D=P;
  //d[0]=0,d[1]=1;d-полином первой степени p+0
  for(i=0;i<(coeff_count-1);i++)
  {
    D=D/d;//делим полином на d, понижая степень на
1
    /*прибавляем произведение производной i-го порядка в начальной точке на полиномиальный коэффициент */
    Q+=st[i]*D;
  }
}

//Конструктор копирования
DifferentialEquation::DifferentialEquation
(DifferentialEquation &ex)
{P=ex.P,PR=ex.PR,Q=ex.Q;coeff_count=P.getm();}
```

/***Вычисление корней характеристического полинома.**
Функция для вычисления корней характеристического полинома - она просто вызывает функцию newton из файла equation.h, заносит вычисленные корни в отведенный нами массив структур, а затем перебирает все корни с целью определения кратности каждого из них и результат тоже заносит в тот же массив структур */

```
void DifferentialEquation::GetRoot()
{
  int i,j;

  //ищем корни характеристического полинома
  polyroot=cvector(root_count);
  polyroot=(znamenatel);

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

  //Заносим в структуру значения коней
  for(i=0;i<root_count;i++) R[i].A=polyroot[i];
```

```
  //определяем кратность каждого корня и заносим в структуру
  int repeat;
  rcount=root_count;/*Вначале предполагаем, что все корни различны */
  for(i=0;i<root_count;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<root_count;j++)
        if((R[j].J!=-1)&&(R[i].A==R[j].A))
        {
          repeat++;
          rcount--;
          R[j].J=-1;R[j].O=i;
          //устанавливаем признак того, что этот корень уже учтён
        }
      R[i].J=repeat;//кратность корня
    }
  }
}

/*
        Вычисление коэффициентов в формуле для оригинала.
        Функция, вычисляющая коэффициенты оригинала для всех корней; результат помещается в массив структур R. Для вычисления коэффициентов используется приведенная в теоретической части формула
```

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

```
*/

void DifferentialEquation::GetCoeffOrigin()
{
  long i,j,k,l,rd;
  complex zz;
  for(k=0;k<rcount;k++)
  {
    if(R[k].J==1)
    {
```

```
      cpolynom pw=2;pw[0]=-R[k].A;pw[1]=1;
      CH=chislitel; ZN=znamenatel;
      ZN=ZN/pw;
      zz=ZN(R[k].A);
      R[k].AO=CH(R[k].A)/zz;
    }
    if(R[k].J>1)
    {
      for(j=1;j<=R[k].J;j++)
      {
        cpolynom pw=2;pw[0]=-R[k].A;pw[1]=1;
        CH=chislitel; ZN=znamenatel;
        CH=CH/(pw^R[k].J);ZN=ZN/(pw^R[k].J);
        zz=ZN(R[k].A);
        if(j==1)
          R[k].AO=CH(R[k].A)/zz;
        else
        {
          long rd=1;
          for(i=k+1;i<root_count;i++)
          {
            if((R[i].O==k)&&(R[i].AO==complex(0,0))
                          &&(R[i].J==-1))
            {
              for(l=1;l<=rd;l++)
              {
                CH=derive(CH,1)*ZN-derive(ZN,1)*CH;
                ZN^=2;
              }

R[i].AO=CH(R[k].A)/(ZN(R[k].A)*fact(rd));
              rd++;
            }
          }
        }
      }
    }
  }
}


/*
```

Решение ОЛДУ для заданного значения аргумента. Функ-

ция, возвращающая значение решения ОЛДУ при заданном t. Она использует формулу

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

```
*/

double DifferentialEquation::GetValue(double t)
{
  long k,i,index;
  double d;
  complex result(0,0);
  for(k=0;k<rcount;k++)
  {
    index=0;
    if(R[k].J==1)//Если корень простой
      result+=R[k].AO*exp(R[k].A*t);
    if(R[k].J>1)//Если корень кратный
    {
      result+=R[k].AO*exp(R[k].A*t);index++;
      for(i=k+1;i<rcount;i++)
      {
        if((R[i].O==k)&&(R[i].J==-1))
        {
          result+=R[k].AO*pow(t,R[k].J-index)*
                  exp(R[k].A*t)/fact(R[k].J-index);
          index++;
        }
      }
    }
  }
  d=real(result);
  return d;
}
```

/*Определение изображений решения при стандартных возмущениях. Реакция динамического объекта на произвольные входные воздействия может быть получена как комбинация реакций на некоторые стандартные воздействия - другими словами, произвольное входное воздействие может быть разложено в ряд по некоторым стандартным функциям времени. В качестве таких стандартных функций обычно используют единичный импульс, ступенчатую функция, синусоиду или косинусоиду с единичной амплитудой, экспоненту. Поэтому мы выделяем от-

дельные методы решения дифуравнения при стандартизованной в перечисленных вариантах правой части. При этом аргумент variant будет кодировать вид возмущающей функции, а param - ее единственный параметр, если он существует – например, частота синусоиды. */

```
void DifferentialEquation :: StandardSolution (
long variant)
{
  complex tmp1[]={complex(0,0),complex(1,0)};
  complex tmp2[]={complex(1,0)};
  cpolynom p(2,tmp1),one(1,tmp2);
  //p[0]=0,p[1]=1;one[0]=1;
  switch(variant)
  {
    //Свободное движение
    case 0:
      chislitel=Q;
      znamenatel=P;
      break;
    //На входе - импульс
    case 1:
    {
      chislitel=Q+PR*one+one;
      znamenatel=P;
      break;
    }
    //Вход - ступенька
    case 2:
    {
      chislitel=(Q*p+PR*p+one);
      znamenatel=P*p;
      break;
    }
    //Вход - синусоида w/(p^2+w^2)
    case 3:
    {
      chislitel=Q*((p^2)+((freq*one)^2))+
      PR*(p^2)+((freq*one)^2)+one*freq/*PR*freq*/;
      znamenatel=P*(((p^2)+((freq*one)^2)));
      break;
    }
    //Вход косинусоида p/(p^2+w^2)
    case 4:
    {
```

```
      chislitel=Q*((p^2)+((freq*one)^2))+
           PR*((p^2)+((freq*one)^2))+one*p/*PR*p*/;
      znamenatel=P*((p^2)+((freq*one)^2));
      break;
    }
    //Вход - экспонента exp(wt)-> 1/(p+w)
    case 5:
    {
      chislitel=Q*(p+sexp*one)+PR*(p+sexp*one)+one;
      znamenatel=P*(p+sexp*one);
      break;
    }
  }
}
```

/*Решение уравнения при произвольной функции в правой части.
Произвольная функция аппроксимируется последовательностью коротких импульсов, следующих друг за другом, решение ищется в виде суммы реакций на сдвинутые импульсы с использованием дискретного варианта свертки функций:

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

при $\Delta\tau$=1.
Приведенная функция вычисляет реакции на стандартные возмущения или произвольную возмущающую функцию в зависимости от кода в первом аргументе. */

```
dvector FreeSolution(long dnumber,long variant)
{
  long i;
  double t,j;
  //Массивы для решения
  dvector mas(sc),mas1(sc);
  //Сконструируем полиномы заданных размерностей
  cpolynom p(rng+1), pr(rngr+1), s(rng);
  //Инициализипуем полиномы
  for(i=0;i<(rng+1);i++) p[i]=complex(cfn[i],0.0);
  for(i=0;i<(rngr+1);i++)
pr[i]=complex(cfnr[i],0.0);
  for(i=0;i<rng;i++) s[i]=complex(nusl[i],0.0);
```

```
/*Для удобства индексирования переставим коэффициенты полинома в обратном порядке*/
  p=p.reverse();pr=pr.reverse(); s=s.reverse();

  //Конструируем объект-уравнение
  DifferentialEquation de(p,pr,s);

  //Если надо вычислить реакцию на произвольное возмущение
  if(variant==6)
  {
  de.StandardSolution(1);//Общее решение для импульса на входе

    if(dnumber>0)//если задано вычисление производной
    {
      //то доформируем числитель изображения
      complex tmp[]={complex(0,0),complex(1,0)};
      cpolynom pp(2,tmp), sum;
      for(i=0;i<dnumber;i++)
        sum+=(nusl[rng-i-1]*(pp^(dnumber-i-1)));
      de.chislitel=(de.chislitel*(pp^dnumber)-
                    de.znamenatel*sum);
    }

    de.GetRoot();//Вычисляем корни характеристического полинома

    de.GetCoeffOrigin();/*Вычисляем для корней коэффициенты оригинала */

    mas[0]=(dnumber>0)?nusl[1]:nusl[0];

    //Вычисляем реакцию на импульс
    for(t=1;t<double(sc);t++)

mas1[t]=de.GetValue(t*(double(200)/double(sc)));

    //Вычисляем свертку импульсной реакции и функции
    for(t=1.0;t<double(sc);t+=1.0)
    {
      mas[long(t)]=0;
      for(j=1;j<t;j++)
        mas[long(t)]=mas[long(t)]+func[long(t-j)]*
```

```
                    mas1[long(t)];
    }
    return mas;
  }
  else
  //Иначе вызываем функцию для изображения стандартного возмущения
  {
    de.StandardSolution(variant);
    if(dnumber>0)
    {
      complex tmp[]={complex(0,0),complex(1,0)};
      cpolynom pp(2,tmp), sum;
      for(i=0;i<dnumber;i++)
        sum+=(nusl[rng-i-1]*(pp^(dnumber-i)));
      de.chislitel=(de.chislitel*(pp^dnumber)-
                    de.znamenatel*sum);
    }

    //Вычислим корни
    de.GetRoot();

    //Вызываем функцию, вычисляющую коэффициенты оригинала
    de.GetCoeffOrigin();
    mas[0]=(dnumber>0)?nusl[1]:nusl[0];
    for(t=1;t<double(sc);t++)
      mas1[long(t)]=de.GetValue(t*(double(200)/
                    double(sc)));
  }
  return mas1;
}
```

/***Вычисление шага по частоте при расчете частотных характеристик.***
Функция, вычисляющая частоту среза - такую, при которой амплитуда на выходе составит заданную пользователем часть от входной - через нее определяется такая величина шага по частоте, чтобы частота среза была достигнута за заданное пользователем число sc шагов. */

```
double GetMaxWAndStep(cpolynom& cp)
{
  int i;
  complex p(0,0), //аргумент характеристического полинома
  value; //значение характеристического полинома
```

```
  double w, //текущая частота
  stepw, //текущий шаг по частоте
  ccurr; //текущий коэффициент усиления

  w=1,stepw=w/sc, ccurr=0; //просто чтобы войти в цикл
  for(;(ccurr>1.1*cufz) || (ccurr<0.9*cufz);)
  {
    ccurr=10*cufz; //просто чтобы войти в цикл
    for(i=0;i<sc && ccurr>cufz;w+=stepw)
    {
      p=complex(0,w);
      ccurr=1/sqrt(norm(cp(p)));
    }
     if(ccurr<cufz)
       stepw=w/sc;//если вышли раньше по ccurr - уменьшим шаг
     else w*=ccurr/cufz; //иначе меняем начальную частоту
  }
  return(w/sc);
}
```

/***Вычисление амплитудно-частотной характеристики.**
Функция, вычисляющая по заданному уравнению амплитудно-частотную характеристику описываемого уравнением объекта - зависимость амплитуды выхода от частоты на входе. Использует формулу

$$W(\omega)=\frac{|Q(j\omega)|}{|P(j\omega)|}$$

при нулевых начальных условиях. Мы вычисляем ее только для одного варианта - когда правая часть уравнения не содержит производных или интегралов. Вы можете обобщить вычисление для более сложных вариантов. */

```
void AmpFChar()
{
  int i;
  //Вектор коэффициентов из массива
  dvector tp(cfn);

  //Сконструируем полином
  cpolynom p(rng+1);
```

```
  //Инициализируем
  for(i=0;i<(rng+1);i++)
    p[i]=complex(tp[i],0);
```

/*Для удобства индексирования переставим коэффициенты полинома в обратном порядке */

```
  p=p.reverse();

  double step=GetMaxWAndStep(p),w=0;
  complex ap(0,0); //аргумент характеристического полинома
  for(i=0;i<sc;i++)
  {
    ap=complex(0,w);
    afch[i]=1/sqrt(norm(p(ap)));
    w+=10*step;
  }
}
```

/***Вычисление фазо-частотной характеристики.**
Функция, вычисляющая фазо-частотную характеристику – зависимость сдвига фазы выхода по отношению ко входу от частоты на входе. Используется формула $\varphi(\omega)=\operatorname{arctg}\frac{I_Q(\omega)}{R_Q(\omega)}-\operatorname{arctg}\frac{I_p(\omega)}{R_p(\omega)}$ при нулевом первом слагаемом. */

```
void PhazeFreqChar()
{
  int i;
  complex value;
  //Вектор коэффициентов уравнения из массива
  dvector tp(cfn);

  //Сконструируем характеристический полином
  cpolynom p(rng+1);

  //Инициализируем
  for(i=0;i<(rng+1);i++)
    p[i]=complex(tp[i],0);
```

/*Для удобства индексирования переставим коэффициенты полинома в обратном порядке */

```
  p=p.reverse();
```

```
/*Вычислим шаг по частоте, приводящий к заданному усилению cz за
sc шагов */
  double step=GetMaxWAndStep(p),w=0;

  complex ap(0,0); //аргумент характеристического полинома

  double re_value, im_value;
  //Варьируем частоту от нуля с шагом step
  for(i=0;i<sc;i++)
  {
    ap=complex(0,w);
//Вычисляем значение характеристического полинома при заданной
частоте
    value=p(ap);
    //Вычисляем вещественную и мнимую части значения
    re_value=real(value); im_value=imag(value);
    if(re_value>0)
      ffch[i]=-atan(im_value/re_value);
    if(re_value<=0)
      ffch[i]=-M_PI-atan(im_value/re_value);
    w+=10*step;
  }
}
```

5.4. Общие численные методы решения задачи Коши для линейных и нелинейных ОДУ

Рассмотренный в предыдущем разделе символический метод решения линейных ОДУ является аналитическим; к численному решению мы прибегали только при вычислении комплексных корней характеристического полинома. В настоящем разделе мы рассмотрим класс численных методов, пригодных для решения как линейных, так и нелинейных дифференциальных уравнений.

Приступая к численному решению задачи Коши для конкретного дифференциального уравнения (в общем случае - нелинейного относительно искомой функции), мы имеем скудный исходный материал - само уравнение n-го порядка и начальные условия в виде значений функции и n-1 ее производных в момент t=0 (или в более общем случае при t=a, если интервал t

определен как [a,b]).

При этом мы не ставим перед собой задачу отыскания общего решения соответствующего класса уравнений, наша задача - получить частное конкретное решение: найти последовательность значений функции для конечной последовательности значений аргумента, удовлетворяющих заданному уравнению.

По-видимому, решение может быть найдено только в случае его единственности - для корректно поставленных задач. Кроме того, уравнение (или система уравнений) должно быть хорошо обусловленным - малые изменения его коэффициентов или начальных условий должны приводить к малым изменениям решения, в противном случае решение может оказаться неустойчивым. Соответствующее требование устойчивости предъявляется и к методу решения задачи - малым изменениям шага дискретизации по аргументу должны соответствовать малые изменения решения. Теория устойчивости алгоритмов - специальный раздел вычислительной математики, мы коснемся этих вопросов только «вскользь», чтобы предостеречь от возможных недоумений при получении неверных решений.

5.4.1. Одношаговые методы

Систему, заданную одним уравнением n-го порядка или системой таких уравнений, можно привести к системе уравнений первого порядка методом введения новых неизвестных функций - мы обсудили это во вводной части. Чтобы упростить изложение методов решения, ограничимся одним уравнением первого порядка

$$f^{1)}(t)=F(t,f(t)),\ a\le t\le b,$$

с одной неизвестной функцией и начальным условием $f(a)=f_a$, а затем распространим эти методы на систему таких уравнений.

Методы, которые мы будем рассматривать, относятся к классу конечно-разностных, поэтому на первом этапе решения разобьем участок [a, b] на конечное число N узловых точек с равным расстоянием $h=(b-a)/N$ между ними, при этом $t_k=a+kh$, $k=1,2,\ldots N$. Через f_k обозначим значение точного решения в точке t_k, а через $f(t_k)$ - полученное приближенное значение. Нам хотелось бы, чтобы при $h\to 0$ погрешность нашего решения достаточно быстро уменьшалась.

Так как в нашем распоряжении в момент t_k (в самом начале - в момент t_a) есть значение $f(t_k)$, естественно попытаться выразить значение функции на следующем шаге через значение функции и ее аргумента на текущем:

$$f(t_{k+1})=f(t_k)+h\varphi(t_k, f(t_k)).$$

с хорошей конструкцикй функции φ.

Метод Эйлера

Простейшим решением является взять в качестве φ саму функцию F:

$$f(t_0)=t_a, f(t_{k+1})=f(t_k)+hF(t_k, f(t_k)),\ k=0, 1, \ldots, N\text{-}1.$$

Вывести этот метод просто. Из разложения $f(t)$ в окрестности точки t_k имеем

$$f_{k+1}=f_k+h\, f_k^{(1)}+(h^2/2)\, f_{zk}^{(2)}=f_k+hF(t_k, f_k)+(h^2/2)\, f_{zk}^{(2)},$$

Отбрасывая последний член в предположении ограниченности второй производной и с учетом предполагаемой малости шага h, получаем приближение вида:

$$f_{k+1}\cong f_k+hF(t_k, f_k).$$

Геометрический смысл последнего выражения в аппроксимации решения на отрезке $[t_k,\ t_{k+1}]$ отрезком касательной к графику решения в точке t_k.

Метод Эйлера очень прост, но характеризуется первым порядком точности, то есть при стремлении шага дискретизации h к нулю приближенное решение будет сходиться к точному с линейной скоростью по h. Такая медленная сходимость к точному решению препятствует широкому применению метода Эйлера и мы рассмотрим методы, погрешность которых уменьшается быстрее с уменьшением h.

Методы Рунге-Кутта 2-го порядка

Для уменьшения погрешности методов, использующих тейлоровское разложение искомого решения, необходимо использовать большее количество членов ряда. Однако при этом возникает необходимость аппроксимации производных от правых частей ОДУ. Основная идея методов Рунге-Кутта заключается в том, что производные аппроксимируются через значения функции $F(t_k, f(t_k))$ в точках на интервале $[t_k,\ t_k+h]$, которые выби-

раются из условия наибольшей близости алгоритма к ряду Тейлора.

В зависимости от старшей степени h, с которой учитываются члены ряда, построены схемы Рунге-Кутта разных порядков точности. Так, например, для 2-го порядка получено однопараметрическое семейство схем вида

$$f(t_k+h)=f(t_k+h[(1-L)F_k+LF(t_k+\text{г}h,\ t_k+\text{г}F_kh)]+O(h^3),\ (*)$$

где $0<L\le 1$ - свободный параметр, $F_k=F(t_k,\ f(t_k)$, г$=1/2L$, $O(h^3)$ - остаточная погрешность.

Для параметра L наиболее часто используют значения L=0.5 и L=1. В первом случае формула (*) приобретает вид

$$f(t_k+h)=f(t_k)+h[F_k+F(t_k+h,\ f(t_k))+hF_k)]/2$$

Вначале вычисляется приближенное решение ОДУ в точке t_k+h по формуле Эйлера

$$f_э=f_k+hF_k.$$

Затем определяется наклон интегральной кривой в найденной точке

$F(t_k+h,\ f_э)$ и после нахождения среднего наклона на шаге h находится уточненное значение $f_{RK}=f(t_{k+}h)$.

Схемы подобного типа называют «прогноз – коррекция». С целью экономии памяти при программировании алгоритма, обобщенного на системы ОДУ, изменим его запись с учетом того, что $f_k=f_э-hF_k$:

$$f_i(t_k+h)=f_{iэ}+0.5h[F_i(t_k+h,\ f_{iэ})],$$

где i - номер решения для системы ОДУ.

Теперь не придется держать в памяти массив начальных значений f_{i0} - его можно забыть после вычисления значений эйлеровских приближений $f_{iэ}$, размещаемых на месте массива f_{i0}.

Во 2-м случае при L=1 от формулы (*) переходим к схеме

$$f(t_k+h)=f_k+hF(t_k+h/2,\ f_k+hF_k/2).$$

Здесь при прогнозе определяется методом Эйлера решение в точке $(t_k+h/2)$:

$$f_{1/2}=f_k+hF_k/2,$$

а после вычисления наклона касательной к интегральной кривой в средней точке решение корректируется по этому наклону.

Таким образом, для методов Рунге-Кутта 2-го порядка функции $\varphi(t, f(t))$ имеют вид:

$$\varphi(t, f(t))=0.5[F(t, f(t))+F(t+h), f(t)+hF(t, f(t))]$$

или

$$\varphi(t, f(t))=F(t_k+h/2, f_k+hF_k/2).$$

Метод Рунге-Кутта 4-го порядка

Это наиболее популярный из методов Рунге-Кутта классический одношаговый метод четвертого порядка точности.

Для построения вычислительных схем методов Рунге-Кутта 4-го порядка в тейлоровском разложении искомого решения $f(t)$ учитываются члены со степенью шага h до 4-й включительно. После аппроксимации производных правой части ОДУ $F(t, f(t))$ получено семейство схем Рунге-Кутта 4-го порядка, из которых наиболее используемой в вычислительной практике является следующая

$$f(t_k+h)=f(t_k)+\frac{1}{6}(q_1+2q_2+2q_3+q_4)+O(h^5),$$

где

$$q_1=hF(t_k, f(t_k)),$$
$$q_2=hF(t_k+0.5h, f(t_k)+0.5hq_1),$$
$$q_3=hF(t_k+0.5h, t_k+0.5q_2),$$
$$q_4=hf(t_k+h, t_k+q_3).$$

Эта схема на каждом шаге требует вычисления правой части ОДУ в 4-х точках. Схема обобщается для систем ОДУ, записанных в форме Коши. Для удобства программной реализации формулы рекомендуется преобразовать к виду:

$$f_i(t_k+h)=f_i(t_k)+\frac{1}{3}(q_{i1}+2q_{i2}+q_{i3}+q_{i4})+O(h^5), \text{ где}$$

$q_{i1}=h_2F_i(t_k, f_i(t_k))$, $h_2=0.5h$
$q_{i2}=h_2F_i(t_k+h_2, f_i(t_k)+q_{i1})$,
$q_{i3}=hF_i(t_k+h_2, f_i(t_k)+q_{i2})$,
$q_{i4}=h_2F_i(t_k+h, f_i(t_k)+q_{i3})$,

i=1, 2, ..., n - номер уравнения в системе ОДУ из n уравнений.

5.4.2. Многошаговые методы (методы Адамса)

Перед нами все та же задача Коши

$$f^{(1)}(t)=F(t, f(t)), a\le t\le b, f(a)=f_a.$$

В одношаговых методах значение $f(t_{k+1})$ определялось только информацией в предыдущей точке t_k. Представляется

возможным повысить точность решения, если использовать информацию в нескольких предыдущих точках при ее наличии. Так и поступают в методах, которые называются многошаговыми. С первого взгляда на постановку задачи становится очевидным, что в момент старта $t=t_a$ есть только одно начальное условие и, если мы собираемся работать с двумя, тремя или четырьмя предыдущими точками, то не видно, как получить вторую, кроме использования одношаговых методов. Так и поступают; «комплексный» алгоритм решения может выглядеть так:

на первом шаге одношаговым методом получают вторую точку, на втором получают третью с помощью двухшагового метода, на третьем - четвертую с помощью трехшагового метода и т.д., пока для основного метода, который предполагается использовать, не наберется достаточно предыдущих точек.

Другой вариант состоит в том, что весь стартовый набор точек получается с помощью одношагового метода, например, Рунге-Кутта четвертого порядка. Поскольку многошаговые методы предполагаются более точными, для стартового одношагового метода используют обычно большее число промежуточных точек, т.е. работают с более коротким шагом.

Многошаговые алгоритмы можно создать так. Учитывая, что

$$f(t_{k+1})=f(t_k)+\int_{t_k}^{t_k+1} F\left(t, f\left(t\right)\right)dt\,,$$

можно численно проинтегрировать стоящую под знаком интеграла правую часть ОДУ. Если использовать метод прямоугольников (интерполяционный полином для интегрируемой функции - константа), получим обычный метод Эйлера. Если использовать 2 точки и интерполяционный полином первого порядка

$$p(x)=\frac{t-t_{k-1}}{h}F_k-\frac{t-t_k}{h}F_{k-1}\,,$$

то интегрирование по методу трапеций от t_k до t_{k+1} даст следующий алгоритм:

$$f(t_{k+1})=f(t_k)+0.5h(3F_k-F_{k-1}).$$

Аналогично для трех точек будем иметь квадратичный интерполирующий полином по данным (t_{k-2}, F_{k-2}), (t_{k-1}, F_{k-1}), (t_k,

F_k) и интегрирование по методу Симпсона даст алгоритм:

$$f(t_{k+1})=f(t_k)+\frac{h}{12}(23F_k-16F_{k-1}+5F_{k-2}).$$

Для 4-х точек полином будет кубическим и его интегрирование даст:

$$f(t_{k+1})=f(t_k)+\frac{h}{24}(55F_k-59F_{k-1}+37F_{k-2}-9F_{k-3}).$$

В принципе мы могли бы продолжать так сколь угодно долго.

Приведенные алгоритмы носят название методов Адамса-Башфорта второго, третьего и четвертого порядков.

Формально мы можем при построении интерполяционного полинома помимо N уже просчитанных точек использовать и еще R будущих t_{k+1}, t_{k+2}; в простейшем случае набор

$$t_{k+1}, t_k, t_{k-1},\ldots, t_{k-N}.$$

При этом порождается класс так называемых методов Адамса-Моултона. В четырехшаговом варианте он оперирует с данными (t_{k+1}, F_{k+1}), (t_k, F_k), (t_{k-1}, F_{k-1}), (t_{k-2}, F_{k-2}) и его алгоритм:

$$f(t_{k+1})=f(t_k)+\frac{h}{24}(9F_{k+1}+19F_k-5F_{k-1}+F_{k-2}).$$

Нельзя, разумеется, вести расчет по отсутствующим данным, поэтому алгоритмы Адамса объединяют в последовательность алгоритмов Адамса-Башфорта и Адамса-Моултона, получая при этом так называемые методы прогноза и коррекции. Например, метод прогноза и коррекции четвертого порядка выглядит так: вначале прогнозируем по алгоритму Адамса-Башфорта с использованием "прошлых" точек

$$f(t_{k+1})=f(t_k)+\frac{h}{24}(55F_k-59F_{k-1}+37F_{k-2}-9F_{k-3}).$$

Затем по вычисляем приближенное значение правой части уравнения

$$F_{k+1}=F(t_{k+1}, f(t_{k+1}).$$

И, наконец, корректируем $f(t_{k+1})$ с использованием его же приближенного значения

$$f(t_{k+1})=f(t_k)+\frac{h}{24}(9F_{k+1}+19F_k-5F_{k-1}+F_{k-2}).$$

Наиболее эффективные из имеющихся компьютерных

программ, позволяющих пользователю менять величину шага и порядок метода, основаны на методах Адамса высокого порядка (свыше 10). Опыт эксплуатации этих программ показывает, что различия в их реализации могут оказывать более существенное влияние на точность, чем различия во внутренних свойствах самих методов.

5.4.3. Проблема устойчивости

Одношаговые и многошаговые методы можно объединить в одну обобщенную запись:

$$f(t_{k+1})=\sum_{i=1}^{m}\alpha_i f(t_{k+1-i})+h\varphi(t_{k+1},t_k,...,t_{k+1-m};f(t_{k+1}),f(t_k),...,f(t_{k+1-m})).$$

Для одношаговых методов m=1. Если при этом α_1=1, и φ не зависит от t_{k+1} и $f(t_{k+1})$, то обобщенный алгоритм превращается в одношаговый метод. Если функция φ имеет вид

$$\varphi=\sum_{i=1}^{m}\beta_i F(t_{k+1-i},f((t_{k+1-i}),$$

то обобщенный алгоритм превращается в линейный многошаговый метод. Обобщенный алгоритм включает все используемые в настоящее время методы.

Описываемый обобщенный метод является *устойчивым*, если все нули полинома

$$\rho(\lambda)\equiv\lambda^m-\alpha_1\lambda^{m-1}-\ldots-\alpha_m$$

удовлетворяют условию $|\lambda_i|\le 1$, а любой нуль, такой что $|\lambda_i|=1$, является простым. Если в дополнение к этому m-1 нулей полинома таковы, что $|\lambda_i|<1$, то метод является *строго устойчивым.*

Любой метод, имеющий по крайней мере первый порядок точности, должен удовлетворять условию $\sum_{i=1}^{m}\alpha_i=1$ и, следовательно, единица должна быть нулем соответствующего полинома. В этом случае строго устойчивого метода полином будет иметь один нуль в точке 1, а все остальные - строго меньше 1. Так как методы Рунге-Кутта одношаговые, то для них $\rho(\lambda)=\lambda-1$. Его единственный корень равен 1, поэтому методы Рунге-Кутта всегда строго устойчивы. Для m-шагового метода Адамса $\rho(\lambda)=\lambda^m-\lambda^{m-1}$, так что остальные m-1 корней равны нулю и такие

методы тоже строго устойчивы.

Рассмотренные результаты теории устойчивости относятся к устойчивости в пределе при $h \to 0$. Даже строго устойчивые методы могут вести себя неустойчиво, если h слишком велик. Попытка обойти эту трудность чрезмерным уменьшением h может привести к недопустимо большим затратам машинного времени, в частности для дифференциальных уравнений, которые называют *жесткими.*

Жесткость есть свойство задачи, а не численного метода решения. Проблемы обеспечения устойчивости для жестких систем выходят за рамки нашего вводного курса; скажем только, что общий подход к решению этой проблемы состоит в использовании неявных методов, то есть использующих, например, значения в точке (t_{k+1}, F_{k+1}). Например, использование формулы

$$f(t_{k+1})=f(t_k)+hF(t_{k+1}, f(t_{k+1}),$$

приводит к неявному методу, который называют *обратным методом Эйлера.*

Он напоминает по форме обычный метод Эйлера, но использует информацию в точке (t_{k+1}, F_{k+1}) и поэтому является неявным.

5.4.4. Программная реализация численных методов решения задачи Коши

```
//файл включения для векторного и матричного типов
#include "matrix.h"
/*для удобства переопределим шаблонные типы векторов и матриц для работы с числовыми объектами - вещественными числами двойной точности */
typedef vector<double> dvector;
typedef matrix<double> dmatrix;

/*Класс дифференциальных уравнений */

class DEqu
{
  double t0, //начальное время
  t1, //конечное время
  step; //шаг по времени
  dvector x0; //вектор начальных условий
```

/*Будем рассматривать класс уравнений, которые удается разрешить относительно старшей производной. В этом случае при представлении уравнения *n*-го порядка в виде системы уравнений первого порядка и решении этой системы нам нужна только одна функция - для правой части последнего уравнения в системе, вычисляющая его правую часть по уже вычисленным до этого значениям младших производных и самой функции.

Для первого уравнения правая часть - в нулевой момент времени -это начальное значение первой производной, для второго уравнения - второй и т.д., а правая часть для последнего уравнения может включать зависимость от времени, значение функции и всех младших производных. Для унификации ввода этой функции в интерактивном режиме надо программировать интерпретатор. Так как нам этого делать не хотелось, то использовать настоящий класс сможет только программирующий пользователь - ему придется запрограммировать функцию правой части и указать ее имя при вызове соответствующего метода решения системы уравнений. Прототип этой функции объявлен через указатель на функцию, чтобы можно было использовать в качестве аргумента другой функции */

```
dvector (*dxbydt)(double t, dvector x);

public:
  //конструктор принимает в качестве параметров все вышеперечисленное
  DEqu(double T0, double Tmax, double Step,
  dvector X0, dvector (*Dxbydt)(double T, dvector
X)):
  t0(T0),     t1(Tmax),     step(Step),     x0(X0),
dxbydt(Dxbydt)
  {
  }

  //конструктор копирования
  DEqu(DEqu &x): t0(x.t0), t1(x.t1), step(x.step),
                 x0(x.x0), dxbydt(x.dxbydt)
  {
  }
```

/*Прототипы функций, реализующих методы Эйлера, Рунге-Кутты второго и четвертого порядка точности и Адамса. Функции возвращают матрицы с количеством столбцов, равным порядку системы, то есть в нулевом столбце - значения функции, а в последующих – значения производных соответствующего номеру столбца порядка */

```
  dmatrix EilerSol();
  dmatrix RK2Sol();
  dmatrix RK4Sol();
  dmatrix Adams4Sol();

  //набор функций для доступа и модификации параметров системы
  double &TStart() { return t0; }//начальное время
  double &TEnd() { return t1; }//конечное время
  dvector &XStart() { return x0; }//вектор начальных
условий
  double &StepByT() { return step; }//шаг по времени
};


//Метод Эйлера
dmatrix DEqu::EilerSol()
{
  //определяем количество точек, которые нам необходимо вычислить
  long StepCount=(TEnd()-TStart())/StepByT();

/*создаём матрицу, количество строк в которой равно количеству точек
решения с учётом начальной (нулевой) точки*/
  dmatrix x(StepCount+1,1);

  x[0]=XStart();//записываем в нулевую точку начальные усло-
вия

  //итеративно вычисляем все послеедующие точки решения
  for(long i=0;i<StepCount;i++)

x[i+1]=x[i]+StepByT()*dxbydt(TStart()+i*StepByT(),
                                    x[i]);
  return x;//возвращаем результирующую матрицу
}

//Решение по методу Рунге-Кутта второго порядка (метод Хойна)
dmatrix DEqu::RK2Sol()
{
  //определяем количество вычисляемых точек
  long StepCount=(TEnd()-TStart())/StepByT();
```

```
  //создаём матрицу для хранения решений
  dmatrix x(StepCount+1, 1);

  //заносим начальные условия в 0-й вектор матрицы
  x[0]=XStart();

  for(long i=0;i<StepCount;i++)
  {
    //вычисляем вектор-коэффициенты следующей точки решения
    dvector k1=dxbydt(TStart()+i*StepByT(),x[i]);
    dvector k2=dxbydt(TStart()+(i+1)*StepByT(),
                      x[i]+StepByT()*k1);

    //вычисляем вектор решения на следующем шаге
    x[i+1]=x[i]+(StepByT()/2)*(k1+k2);
  }
  return x;//возвращаем матрицу-результат
}


//Решение по методу Рунге-Кутта 4-го порядка точности
dmatrix DEqu::RK4Sol()
{
  //количество точек решения
  long StepCount=(TEnd()-TStart())/StepByT();

  dmatrix x(StepCount+1, 1);//матрица решения

  x[0]=XStart();//начальные условия

  for(long i=0;i<StepCount;i++)
  {
    //вычисляем вектор-коэффициенты решения
    dvector k1=dxbydt(TStart()+i*StepByT(),x[i]);
    dvector k2=dxbydt(TStart()+(i+0.5)*StepByT(),
                      x[i]+(StepByT()/2)*k1);
    dvector k3=dxbydt(TStart()+(i+0.5)*StepByT(),
                      x[i]+(StepByT()/4)*(k1+k2));
    dvector k4=dxbydt(TStart()+(i+1)*StepByT(),
                      x[i]+StepByT()*(-k2+2*k3));

  x[i+1]=x[i]+(StepByT()/6)*(k1+4*k3+k4);//следующая
точка
```

```
  }
  return x;//возвращаем матрицу решений
}

//Метод Адамса
dmatrix DEqu::Adams4Sol()
{
  //определяем количество точек решения
  long StepCount=(TEnd()-TStart())/StepByT();

  dmatrix x(StepCount+1,1);//матрица решения - массив векторов

/*преобразуем указатель на данный объект в указатель на объект для решения системы дифференциальных уравнений методом Рунге-Кутта четвёртого порядка и получаем объект соответствующего типа вызовом конструктора копирования */
  DEqu obj=*this;

/*если количество точек решения таково, что данную систему нельзя решить методом Адамса-Башфорта, возвращаем решение по методу Рунге-Кутта*/
  if(StepCount<=3) return obj.RK2Sol();

/*В противном случае модифицируем параметры системы (конечное время) так, чтобы по методу Рунге-Кутта четвёртого порядка находились только первые четыре решения */
  obj.TEnd()=TStart()+3*StepByT();

  dmatrix tempx=obj.RK4Sol();//получаем первые четыре точки

  for(long i=0;i<4;i++)//и переписываем их в векторы
  x[i]=tempx[i]; //результирующей матрицы

  //начиная с пятой точки, работает собственно метод Адамса
  for(long i=3;i<StepCount;i++)
    x[i+1]=x[i]+(StepByT()/24)*
    (
    55*dxbydt(TStart()+i*StepByT(),x[i])-
    59*dxbydt(TStart()+(i-1)*StepByT(),x[i-1])+
    37*dxbydt(TStart()+(i-2)*StepByT(),x[i-2])-
```

```
    9*dxbydt(TStart()+(i-3)*StepByT(),x[i-3])
    );
  return x;//возвращаем результирующую матрицу
}


//Тестовые функции
dvector testfunc(double x, dvector y)
{
  //Решение уравнения Бесселя
  dvector dy=2;
  dy[0]=y[1];//начальное условие для первой производной
  dy[1]=-y[0]-y[1]/x;


/*решение уравнений движения
  dvector dy=4;
  dy[0]=y[2];
  dy[1]=y[3];
  dy[2]=-y[0]/hypot(y[0],y[1]);
  dy[3]=-y[1]/hypot(y[0],y[1]);*/
  return dy;
}


void main()
{
  dvector StartCond=2;
  StartCond[0]=0.9384698;
  StartCond[1]=-0.2422685;
  /* dvector StartCond=4;
     StartCond[0]=0.5;
     StartCond[1]=0;
     StartCond[2]=0;
     StartCond[3]=1.63;*/
  DEqu testobj(0.5, 1.1, 0.1,/*0, 20.3, 0.1,*/
               StartCond, testfunc);
  dmatrix resEiler=testobj.EilerSol();
  dmatrix resRK2=testobj.RK2Sol();
  dmatrix resRK4=testobj.RK4Sol();
  dmatrix resAdams4=testobj.Adams4Sol();
  cout<<resEiler<<endl<<resRK2<<endl;
}
```