Министерство образования и науки Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Утверждаю: Зав.каф. ПМиИ, профессор

_____ (Тимченко С. В.)

Мещеряков П. С.

ПРИКЛАДНАЯ ИНФОРМАТИКА

Методические указания по практическим занятиям и самостоятельной работе студентов для бакалавров по направлению 210100 «Электроника и наноэлектроника» и студентов по специальности 210106 «Промышленная электроника»

Томск 2012

Введение

Как и математический аппарат является инструментом в руках инженера профессионала, так и информатика предоставляет огромные инструментальные возможности для решения различных прикладных задач в различных предметных областях. В том числе множество инструментов для облегчения и ускорения процесса математических вычислений. Практически все пакеты прикладных программ, каждой предметной области содержат реализацию основных математических действий, применяемых в данной области. Существуют на сегодня и отдельные, очень мощные по своим вычислительным возможностям специальные математические пакеты, которые становятся одним из основных рабочих инструментов исследователя, инженера, разработчика и т.д.

Важно понимать, что расчеты с использованием компьютера осуществляются не в аналитической форме, а в численном виде, что накладывает дополнительные условия и ограничения. Поэтому необходимо не только разбираться в инструментах математического аппарата, но и иметь представление о их реализации. Какой из них имеет большую скорость сходимости, какие методы устойчивы, а какие нет и т.д

Основы алгоритмизации

Структурное программирование

В настоящее время большинство программистов признают, что для эффективного и быстрого проектирования надежных, «правильных» программ необходимо использовать стиль *структурного программирования.* Основные принципы, заложенные в его основу, состоят в том, чтобы при записи алгоритмов пользоваться ограниченным набором инструкций, а именно структурными операторами:

- оператором присваивания «**:=**»;
- условным оператором «**if**...**then**...**else**»;
- оператором цикла «**do**...**while**».

Хотя этих трех операторов достаточно для построения программ любой сложности, вышесказанное не надо воспринимать как догму.[1] Разумеется, вполне допустимы и другие структурные операторы - оператор выбора, другие операторы цикла и т.д. Структурное программирование предполагает при построении программ *метод пошаговой детализации*, при котором, вначале разрабатывается общая структура программы, состоящая из записи алгоритма не с помощью элементарных операций, а более крупными блоками. Впоследствии, на следующих этапах каждый из этих блоков детализируется на более мелкие, пока в результате не получатся элементарные действия, которые можно выразить непосредственно средствами языка программирования.

Другими словами, *структурное программирование* - это метод программирования, обеспечивающий создание программ, структура которых ясна и непосредственно связана со структурой решаемых задач.

Пошаговая разработка программ

Как уже отмечалось выше, программирование неразрывно связано с проектированием алгоритмов. При этом часто удобно сочетать поэтапную разработку алгоритма с написанием собственно программы. Этот метод носит название *пошаговой разработки программ* **.**

Конструирование алгоритма требует тщательного осмысливания и исследования возможных решений. На ранних стадиях обращают внимание главным образом на глобальные проблемы, упуская из виду многие детали. По мере продвижения процесса проектирования задача разбивается на подзадачи, и постепенно все большее внимание уделяется подробному описанию проблемы и характеристикам имеющихся инструментов программирования.

Вероятно, наиболее общая тактика программирования состоит в разбиении задачи на небольшое число подзадач, каждая из которых представляет собой более простую самостоятельную задачу, а соответствующих программ – на отдельные инструкции. При этом на каждом таком шаге этой декомпозиции решения частных задач должны приводить к решению общей задачи, а выбранная последовательность индивидуальных действий должна быть разумна и позволять получить инструкции более близкие к языку, на котором, в конечном счете, будет сформулированна программа.

Рекуррентные алгоритмы

При решении различного рода математических задач часто приходится иметь дело с алгоритмами, построенными на рекуррентных соотношениях, т.е. выражениях вида

$$v_i = f(v_{i-1}) \text{ для всех } i > 0.$$

При этом последовательность $v_0, v_1, \ldots, v_n$ называется рекуррентной последовательностью. Часто используют более общее определение элементов рекуррентной последовательности при помощи формул вида

$$v_i = f(i, v_{i-1}, v_{i-2}, \ldots, v_{i-k}), i \geq k$$

$$v_0 = x_0,$$

$$v_1 = x_1,$$

$$\ldots,$$

$$v_{k-1} = x_{k-1}.$$

Однако не трудно видеть, что на самом деле это определение становится частным случаем определения , если в качестве v_i рассматривать

вектор, одним из компонентов которого может быть собственно номер элемента рекуррентной последовательности *i*.

С самым простым примером использования рекуррентных соотношений мы сталкиваемся при определении факториала

$$f(n) = n! = 1 \cdot 2 \cdot \ldots \cdot n,\ n \geq 0,$$

который можно вычислить при помощи рекуррентных соотношений

$$f(0) = 1,$$

$$f(i) = i \cdot f(i-1).$$

При реализации этого алгоритма на языке Free Pascal удобно заменить рекуррентные формулы системой рекуррентных соотношений

$$\left.\begin{array}{l} f_i = k_i \cdot f_{i-1} \\ k_i = k_{i-1} + 1 \end{array}\right\} \text{ для } i>0$$

с начальными значениями

$$f_0 = 1,\ k_0 = 0.$$

Рекурсия

Элегантным и универсальным методом, заслуживающим специального рассмотрения, является рекурсия. Рекурсия первоначально была разработана математиками как средство для определения функций и широко использовалась в математической логике.

Рекурсивные определения (будь то функции или алгоритмы) характеризуются тем, что определяемый объект сам фигурирует в определении. Или, другими словами, объект называется рекурсивным, если он содержит сам себя или определен с помощью самого себя.

Самый простой пример - рекурсивное определение факториала:

0!=1; если $n > 0$, то $n! = n\,(n\text{-}1)!$

Мощность рекурсии связана с тем, что она позволяет определить бесконечное множество объектов с помощью конечного высказывания.

Структуры данных

На первых этапах развития вычислительной техники компьютеры предназначались для проведения сложных и длительных расчетов. Однако впоследствии оказалось, что на первый план вышла их способность хранить и обрабатывать большие объемы информации. Представляя некоторую *абстракцию* какой-то части реального мира, информация, доступная компьютеру, состоит из некоторых *данных* о действительности, которые *считаются* относящимися к решаемой задаче и из *которых* предполагается получить требуемый результат.

Концепция структур данных чрезвычайно важна в информатике. Не случайно одна из самых известных книг в этой области называется «Алгоритмы + структуры данных = программы» (ее автор – Н. Вирт – является основным разработчиком языка Паскаль). Правда, с математической

точки зрения это название не совсем точно, так как мы помним, что входные и выходные данные алгоритма являются его неотъемлемой частью. В конечном счете, *программы* представляют собой конкретные формулировки абстрактных *алгоритмов*, основанные на конкретных представлениях и структурах *данных*. При этом данные представляют собой абстракции реальных объектов, сформулированные в терминах конкретного языка программирования.

Машинная арифметика

Появление компьютеров изменило характер вычислений по двум причинам – во-первых, резко вырос объем вычислений, во-вторых, качественно изменился их характер, т.к. машинная арифметика существенно отличается от арифметики ручной.

Это отличие объясняется, в основном, следующими причинами:

1) множество *чисел с плавающей точкой* (т.е. вещественных чисел, имеющих *различное* машинное представление) конечно;

2) существуют наименьшие и наибольшие *числа с плавающей точкой*;

3) существует вещественное число (машинная точность) равное разнице между 1.0 и следующим по величине вещественным числом;

4) арифметические операции редко приводят к точно представимым результатам, поэтому результат округляется до ближайшего *числа с плавающей точкой*;

5) *числа с плавающей точкой* расположены между нулем и наибольшим значением неравномерно – их больше около нуля.

Указания для самостоятельной работы

При самостоятельной проработке материала данной темы следует уделить особое внимание технологиям программирования, ознакомиться с различными существующими подходами, позволяющими минимизировать ошибки при проектировании программ. Изучить вопрос влияния типов данных языков программирования и архитектуры вычислительных систем на точность вычисляемых значений.

Решение СЛАУ

Обусловленность матрицы

Ошибки округлений, совершенных в процессе вычислений, почти всегда приводят к тому, что вычисленное решение (которое мы будем обозначать через $\mathbf{x}_*$) в определенной степени отличается от теоретического решения $\mathbf{x} = A^{-1}\mathbf{b}$. В действительности оно и должно отличаться, поскольку компоненты вектора *x* обычно не являются числами с плавающей точкой.

Имеются две общепринятые меры отклонения для $\mathbf{x}_*$:

Ошибка $e = \mathbf{x} - \mathbf{x}_*$

и

Невязка $r = \mathbf{b} - A\mathbf{x}_*$.

Из теории матриц известно, что если одна из этих величин равна нулю, то и другая также должна быть равна нулю. Но они совсем не обязательно одновременно будут «малыми».

Рассмотрим следующий простой (хотя и специально сконструированный) пример:

$$\begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.217 \\ 0.254 \end{pmatrix}.$$

Если провести Гауссово исключение с частичным выбором на гипотетической *трехзначной* десятичной машине (т.е. результатом любой арифметической операции является число с тремя десятичными значащими цифрами), то получим решение

$$\mathbf{x}_* = \begin{pmatrix} -0.443 \\ 1.000 \end{pmatrix}.$$

Если бы мы не знали правильного ответа, то с целью оценить полученную точность могли бы (точно) вычислить невязку:

$$r = \mathbf{b} - A\mathbf{x}_* = \begin{pmatrix} -0.000460 \\ -0.000541 \end{pmatrix}.$$

Все невязки меньше, чем 10^{-3}. Едва ли можно ожидать лучшего на трехзначной машине. Однако легко видеть, что точное решение этой системы есть

$$\mathbf{x} = \begin{pmatrix} 1.000 \\ -1.000 \end{pmatrix}.$$

Таким образом, ошибка больше, чем решение.

Метод прогонки

Модификация метода исключения Гаусса, в которой учитывается структура матрицы A, называется *методом прогонки*.

Пусть дана система линейных алгебраических уравнений вида:

$$y_0 + \lambda_1 y_1 = \mu_1;$$
$$a_i y_{i-1} + c_i y_i + b_i y_{i+1} = f_i; \qquad i = 1..n-1,$$
$$y_n + \lambda_2 y_{n-1} = \mu_2.$$

Это обычный вид таких систем, которые появляются при решении практических задач. Здесь λ , μ , a, b, c– константы.

Запишем ее в развернутом виде.

$$\begin{pmatrix} 1 & \lambda_1 & 0 & 0 & \dots & 0 & 0 \\ a_1 & c_1 & b_1 & 0 & \dots & 0 & 0 \\ 0 & a_2 & c_2 & b_2 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{n-1} & c_{n-1} & b_{n-1} \\ 0 & 0 & 0 & \dots & 0 & \lambda_2 & 1 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \dots \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} \mu_1 \\ f_1 \\ f_2 \\ \dots \\ f_{n-1} \\ \mu_2 \end{pmatrix}.$$

Будем искать решение этой системы в виде рекуррентной формулы, связывающей y_i и y_{i+1} :

$$y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1}, \quad i = 0,1,...,n-1,$$

где α_1 и β_1 – известные константы ($\alpha_1 \equiv -\lambda_1$; $\beta_1 \equiv \mu_1$) , а α_2, ..., α_n и β_2,..., β_n – пока неизвестные константы.

Понизим индекс на 1:

$$y_{i-1} = \alpha_i y_i + \beta_i.$$

Теперь, подставив во второе уравнение системы:

$$a_i(\alpha_i y_i + \beta_i) + c_i y_i + b_i y_{i+1} = f_i,$$

где i=1, ..., n–1, приходим к уравнению вида

$$y_i + \frac{b_i}{a_i \alpha_i + c_i} y_{i+1} = \frac{f_i - a_i \beta_i}{a_i \alpha_i + c_i}.$$

Сравнивая полученное равенство, находим

$$\alpha_{i+1} = \frac{-b_i}{\alpha_i a_i + c_i}; \quad \beta_{i+1} = \frac{f_i - a_i \beta_i}{\alpha_i a_i + c_i}.$$

Эти рекуррентные соотношения называются *формулами прямой прогонки*. Они позволяют по заданным значениям α_1 и β_1 последовательно определить все пары прогоночных коэффициентов (α_2,β_2), (α_3,β_3),..., (α_n,β_n).

Затем начинается второй этап метода прогонки – обратная прогонка.

Запишем уравнение для i=n–1 и добавим еще не использованное третье уравнение системы. Получаем систему из двух уравнений с двумя неизвестными y_n и y_{n-1}:

$$\begin{cases} y_{n-1} = \alpha_n y_n + \beta_n; \\ y_n = -\lambda_2 y_{n-1} + \mu_2. \end{cases}$$

Ее решение: $$y_n = \frac{\lambda_2 \beta_n + \mu_2}{1 + \lambda_2 \alpha_n}.$$

Все остальные значения y_i ($i = n–1, n–2,\ldots,0$) находим по рекуррентной формуле. Формулы и называются *формулами обратной прогонки*. Метод прогонки является одним из наиболее эффективных и распространенных численных алгоритмов линейной алгебры.

Указания для самостоятельной работы

При самостоятельной проработке материала данной темы следует уделить внимание вопросу различия матрицы коэффициентов СЛАУ, как вид матрицы влияет на способ ее представления и хранения в памяти компьютера. А так же какие особенности матрицы коэффициентов влияют на выбор методов решения СЛАУ.

Собственные значения и собственные вектора

Полную проблему собственных значений для матриц невысокого порядка ($n \le 10$) можно решить методом непосредственного развертывания.

Если раскрыть определитель, то он превратится в полином степени n относительно λ:

$$P(\lambda) = \lambda^n + p_{n-1} \cdot \lambda^{n-1} + p_{n-2} \cdot \lambda^{n-2} + \cdots + p_1 \cdot \lambda + p_0,$$

где n - размер матрицы A, а коэффициенты p_k зависят только от значений элементов матрицы A. Уравнение примет вид:

$$P(\lambda) = \lambda^n + p_{n-1} \cdot \lambda^{n-1} + p_{n-2} \cdot \lambda^{n-2} + \cdots + p_1 \cdot \lambda + p_0 = 0.$$

Данное уравнение еще называется характеристическим уравнением. Таким образом, задача о поиске собственных чисел матрицы размера $n \times n$ сводится к поиску корней полинома степени n.

В общем случае полином может быть представлен в виде произведения:

$$P(\lambda) = (\lambda - \lambda_1)^{m_1} (\lambda - \lambda_2)^{m_2} \cdots (\lambda - \lambda_K)^{m_K},$$

где λ_i — i - ый корень полинома;

m_i — кратность корня λ_i ;

K — число различных корней

В математике есть т.н. *основная теорема алгебры*, которая утверждает: *всякий полином степени* ***n*** *имеет в поле комплексных чисел ровно* ***n*** *корней, причем каждый корень считается столько раз, какова его кратность*. Это означает, что $m_1+m_2+\ldots+m_K = n$.

Задача поиска корней полинома в аналитическом виде решена лишь для $n \le 4$, для $n > 4$ возможен только их численный поиск.

С собственным числом матрицы связано понятие *собственный вектор*. Собственным вектором матрицы A, соответствующим собственному числу λ_i называют вектор t_i, для которого справедливо соотношение:

$$A \cdot t_i = \lambda_i \cdot t_i \text{ , где } i = 1 \ldots n.$$

Допустим, что матрица A имеет n различных собственных чисел и, соответственно, n собственных векторов. Составим матрицу T, столбцы которой образованы векторами t_i:

$$\mathrm{T} = \begin{pmatrix} \mathrm{t}_1 & \mathrm{t}_2 & \cdots & \mathrm{t}_\mathrm{n} \end{pmatrix},$$

и запишем уравнения в матричной форме:

$$\mathrm{A} \cdot \mathrm{T} = \mathrm{T} \cdot \begin{pmatrix} \lambda_1 & 0 & \cdots & \\ 0 & \lambda_2 & \cdots & \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \lambda_\mathrm{n} \end{pmatrix}.$$

При определении собственных векторов, для каждого собственного числа (корня характеристического уравнения) необходимо составить систему:

$$(A-\lambda_i * E) * X_i = 0, \qquad i = 1,2,\ldots,n,$$

и найти собственные векторы X_i.

Указания для самостоятельной работы

При самостоятельной проработке материала данной темы следует уделить внимание альтернативным алгоритмам нахождения собственных чисел и собственных векторов. В каких случаях предпочтителен тот или иной метод.

Интерполяция

Достаточно часто в реальной практике приходится сталкиваться данными полученными эмпирически, в результате каких либо наблюдений. Как правило, наблюдения осуществляются в дискретные интервалы времени. Нам доступны только те значения, которые мы наблюдали в эти моменты, а что происходило в промежутке между моментами регистрации данных нам не известно, а зачастую эти данные необходимы.

Предположим, что задано множество вещественных абсцисс $x_1, \ldots, x_n$ $(x_1 < x_2 < \ldots < x_n)$ и им соответствующие ординаты $y_1, \ldots, y_n$. Задача одномерной интерполяции состоит в построении функции f, такой, что $f(x_i) = y_i$ для всех i, и при этом $f(x)$ должна принимать «разумные» значения для x, лежащих между заданными точками. Критерий разумности слабо формализован, существенно зависит от задачи и ему невозможно дать точное определение.

Интерполяция часто встречается как при работе на компьютере, так и в обычной жизни. Например, если стрелка спидометра автомобиля (или часов, или любого другого прибора) находится между делениями, мы мысленно интерполируем, чтобы получить скорость. Если у нас есть данные, полученные с большими затратами всего в нескольких точках (например, результаты переписи населения, проводимой раз в десять лет), то может потребоваться определить величины между этими точками. Если ординаты $\{y_i\}$ происходят от гладкой математической функции, и ошибки в них не превосходят уровня округлений, то можно рассчитывать, что задача имеет удовлетворительное решение.

Если точки $(x_i, y_i,)$ получены из очень точных экспериментальных наблюдений, то зачастую их можно считать лишенными ошибок, и тогда вполне разумно интерполировать их гладкой функцией. Если, с другой стороны, эти точки проистекают из сравнительно грубых экспериментов, то неправомерно требовать от интерполирующей функции точно удовлетворять таким данным. Позволяя значениям $f(x_i)$ отличаться от y_i, можно очень хорошо отразить характер изменения данных и даже поправить некоторые из содержащихся в них ошибки.

Цели интерполяции разнообразны, но почти всегда в ее основе - желание иметь быстрый алгоритм вычисления значений $f(x)$ для x, не содержащихся в таблице данных (x_i, y_i).

Для задачи интерполирования очень важно определение того, как должна вести себя приемлемая функция между заданными точками. В конце концов эти точки могут быть интерполированы бесконечным множеством различных функций, и нужно иметь некоторый критерий выбора. Обычно критерии формулируются в терминах гладкости и простоты; например, функция f должна быть аналитична и максимальное значение $|f(x)|$ по всему интервалу должно быть насколько возможно мало, или f должна быть полиномом наименьшей степени, и т. п.

Многие интерполирующие функции генерируются линейными комбинациями элементарных функций. Линейные комбинации одночленов $\{x_k\}$ приводят к полиномам. Линейные комбинации тригонометрических функций $\{\cos kx, \sin kx\}$ ведут к тригонометрическим полиномам. Используются также, хотя и реже, линейные комбинации экспонент $\{\exp(b_k x)\}$ или рациональные функции вида:

$$\frac{a_0 + a_1 x + \ldots + a_m x^m}{b_0 + b_1 x + \ldots + b_n x^n}.$$

Указания для самостоятельной работы

При самостоятельной проработке материала данной темы следует уделить внимание закреплению понимания различий между полиномиальной и кусочно-полиномиальной интерполяцией. Какими положительными и отрицательными моментами обладают эти подходы, в какой момент предпочтительны первые методы, а когда вторые.

Численное интегрирование

Рассмотрим решение следующей задачи – вычисление определенного интеграла:

$$I = \int_a^b f(x)dx .$$

Это одна из фундаментальных задач математического анализа, тесно связанная, в частности, с решением дифференциальных уравнений.

Если нет возможности выразить интеграл в известных элементарных или специальных функциях, то применяется приближенное численное интегрирование. С другой стороны, при решении многих инженерных задач удачно выбранный численный метод может оказаться экономичней вычисления точного значения интеграла, выраженного через тригонометрические и прочие специальные функции.

Рассмотрим квадратурные формулы (методы численного интегрирования), основанные на интерполировании по небольшому числу точек. Заменяя подынтегральную функцию, например, интерполяционным полиномом Лагранжа, получаем приближенную формулу:

$$I = \int_a^b f(x)dx \approx \int_a^b L_n(x)dx .$$

При этом предполагается, что отрезок $[a, b]$ разбит на n частей точками $x_0 = a, x_1, x_2, \ldots, x_{n-1}, x_n = b$, по которым строят интерполяционный полином. Подставляя выражение для интерполяционного полинома Лагранжа, получим, что определенный интеграл можно представить в виде:

$$\int_a^b f(x)dx \approx \sum_{k=0}^{n} C_k y_k ,$$

причем коэффициенты $C_к$ не зависят от подынтегральной функции $f(x)$, а только от значений узлов интерполяции.

Это приближенное равенство называется *квадратурной формулой*. Точки $x_к$ называются *узлами квадратурной формулы*, а числа $C_к$ – *коэффициентами квадратурной формулы*.

Разность

$$R_n(f) = \int_a^b f(x)dx - \sum_{k=0}^{n} C_k y_k$$

называется *погрешностью квадратурной формулы*.

Отметим, что:

1) погрешность зависит как от расположения узлов, так и от выбора коэффициентов;

2) если функция $f(x)$ – полином степени n , то тогда квадратурная формула будет точной (то есть $R_n(f)\equiv 0$), так как в этом случае $L_n(x)\equiv f(x)$;

Получим некоторые простые (и в то же время весьма распространенные) формулы численного интегрирования, используя интерполяционный полином Лагранжа.

Для получения формул численного интегрирования на некотором отрезке $[a, b]$, достаточно построить квадратурную формулу для интеграла на элементарном отрезке $[x_i, x_{i+1}]$, а затем ее просуммировать, т.к.

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx .$$

Указания для самостоятельной работы

При самостоятельной проработке материала данной темы следует уделить внимание оценкам порядка точности и погрешности квадратурных формул.

Численное решение задачи Коши для ОДУ

Обыкновенное дифференциальное уравнение (ОДУ) первого порядка можно записать в виде

$$y'=f(y, t).$$

Это уравнение имеет семейство решений $y(t)$. Например, если $f(y, t)=y$, то для произвольной константы C функция $y(t) = Ce^t$ является решением. Выбор начального значения, скажем $y(0)$, служит для выделения одной из кривых семейства. Начальное значение зависимой переменной может быть задано для любого значения t_0 независимой переменной. Однако часто считают, что выполнено преобразование, обеспечивающее, чтобы $t_0=0$. Это не влияет на решение или методы, используемые для приближения решения.

Зачастую имеется более чем одна зависимая переменная, и тогда задача заключается в решении системы уравнений первого порядка; например,

$$\frac{dy}{dt} = f(t, y, z),$$

$$\frac{dz}{dt} = g(t, y, z).$$

Решение этой системы содержит две постоянные интегрирования, и,

следовательно, нужны два дополнительных условия, чтобы определить эти константы. Если значения y и z указаны при одном и том же значении независимой переменной t_0, то система будет иметь единственное решение. Задача определения значений y и z для (будущих) значений $t > t_0$ называется *задачей Коши*.

Любое обыкновенное дифференциальное уравнение порядка n, которое можно записать так, что его левая часть есть производная наивысшего порядка, а в правой части эта производная не встречается, может быть записано и в виде системы из n уравнений первого порядка путем введения n - 1 новых переменных. Например, уравнение второго порядка:

$$u'' = g(u, u', t)$$

можно записать как систему:

$$v' = g(u, v, t),$$

$$u' = v.$$

В векторных обозначениях это выглядит так:

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}, t),$$

где:

$$\mathbf{y} = \begin{pmatrix} v \\ u \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix};$$

$$\mathbf{f}(\mathbf{y}, t) = \begin{pmatrix} g(y_2, y_1, t) \\ y_1 \end{pmatrix}.$$

При обсуждении методов для задачи Коши удобно представлять себе единственное уравнение:

$$y' = f(y, t);$$

$$y(t_0) = y_0.$$

Однако методы с равным успехом применимы и к системам уравнений.

Лишь очень немногие дифференциальные уравнения могут быть решены точными или приближенными аналитическими методами.

Поэтому (особенно после появления компьютеров) широкое распространение получили *численные методы*, в основе которых лежит замена исходного уравнения его дискретным *аналогом – разностным* уравнением.

Область непрерывного изменения аргумента заменяется дискретным множеством точек (узлов): t_0, t_1, t_2, t_3, … , возможно, с переменной длиной шага $h_n = t_{n+1} - t_n$. Эти узлы составляют *разностную сетку*. Искомая функция непрерывного аргумента приближенно заменяется функцией дискретного аргумента на заданной сетке, т.е. в каждой точке t_n решение $y(t_n)$ заменяется

приближенным значением (аппроксимируется) y_n, которое вычисляется по предыдущим значениям. Эта функция *называется сеточной функцией*. Она определена только в узлах разностной сетки.

Разностный метод, дающий формулу для вычисления y_{n+1} по k предыдущим значениям y_n, y_{n-1}, y_{n-2}, ..., y_{n-k+1}, называется k-шаговым методом. Если k=1, то это одношаговый метод, а при k>1 это многошаговый метод.

Указания для самостоятельной работы

При самостоятельной проработке материала данной темы следует уделить особое внимание вопросу устойчивости и точности получаемого решения.

Решение нелинейных уравнений

Пусть f - некоторая функция одной переменной. Задача состоит в том, чтобы найти одно или более решений уравнения $f(x) = 0$. Используемые алгоритмы являются итерационными, и здесь важны такие вопросы: много ли требуется вычислений функции $f(x)$, нужны ли вычисления производных $f'(x)$ и $f''(x)$ и т. д. Чтобы начать поиск нуля $f(x)$, предположим, что можно найти интервал $[a, b]$, на котором $f(x)$ меняет знак. Однако если мы ничего не знаем относительно f, то мы не можем быть уверены, что она имеет нуль на этом интервале. Даже если *математическая* функция непрерывна и изменяет знак в $[a, b]$, *вычисляемая* функция переменного с плавающей точкой, значения которой также являются числами с плавающей точкой, принимает лишь дискретное множество значений, среди которых, возможно, нет нуля. Поэтому в конечном счете более практично искать не нуль f, а малый интервал $[\alpha,\beta]$, в котором f меняет знак. Такой интервал всегда можно найти и можно сузить его настолько, насколько позволяет система чисел с плавающей точкой, т. е. так, чтобы концевыми точками были два соседних числа этой системы. Если о функции f ничего не известно, то наиболее надежным алгоритмом является метод бисекции или половинного деления.

Указания для самостоятельной работы

При самостоятельной проработке материала данной темы следует уделить внимание вопросу скорости сходимости различных алгоритмов уточнения корня уравнения.

Литература

1. Численные методы / Н.С. Бахвалов, Н.П. Жидков, Г.М. Кобельников. – 4-е изд. – М. БИНОМ. Лаборатория знаний, 2006. – 636 с.
2. Фармалеев В. Ф., Ревизников Д.Л. Численные методы. – Изд. 2-е, испр., доп. – М.: ФИЗМАТЛИТ, 2006. – 400 с.

3. Численные методы в примерах и задачах: Учеб. Пособие/В.И. Киреев, А.В. Пантелеев. – 2-е изд. стер. – М.: Высш. шк., 2006. – 480 с.