ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ РФ
ГОУ ВПО «ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

В.Н. ТАРАСОВ, Н.Ф. БАХАРЕВА

ЧИСЛЕННЫЕ МЕТОДЫ ТЕОРИЯ АЛГОРИТМЫ ПРОГРАММЫ

ИЗДАНИЕ ВТОРОЕ,
ПЕРЕРАБОТАННОЕ

Рекомендовано ГОУ ВПО МГТУ им. Н.Э. Баумана в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки «Информатика и вычислительная техника».
Регистрационный № рецензии 119 от16.07.2008 г. МГУП

Самара 2008

ББК 22.19я7
Т19
УДК [519.95+004.421](075.8)

Рецензенты:
Заведующий кафедрой «Информационные системы и технологии» СГАУ, Заслуженный работник высшей школы РФ, Академик МАИ, д.т.н., профессор ***С.А. Прохоров***; кафедра «Программное обеспечение ЭВМ и информационные технологии» МГТУ им. Н.Э. Баумана, д.т.н., профессор ***В.М. Градов***.

Тарасов В.Н., Бахарева Н.Ф.

Численные методы. Теория, алгоритмы, программы. – Оренбург: ИПК ОГУ, 2008. – 264 с.

Т19

ISBN 5-7410-0451-2

Учебное пособие предназначено для студентов специальностей направления 230100 – Информатика и вычислительная техника.

ISBN 5-7410-0451-2

Содержание

ПРЕДИСЛОВИЕ

Данное издание учебного пособия «Численные методы. Теория, алгоритмы, программы» включает все основные (классические) разделы дисциплины «Вычислительная математика», предусмотренные государственным образовательным стандартом для студентов направления подготовки 230100 – Информатика и вычислительная техника. Учебное пособие рассчитано на стандартный семестровый курс.

Наряду с теоретическими основами численных методов, пособие содержит также полный комплекс лабораторных работ, включающий схемы алгоритмов методов, коды программ, решения контрольных примеров и варианты заданий. На взгляд авторов, такое представление материала учебного пособия наиболее полно отражает специфику направления подготовки – Информатика и вычислительная техника, т.к. простое использование закрытых математических пакетов типа MathCAD и Matlab с точки зрения построения алгоритмов вычислительных методов мало информативно. Наоборот, умение программировать вычислительные методы поможет лучше понять содержимое математических пакетов программ и их работу.

Все результаты расчетов контрольных примеров лабораторной части учебного пособия проверены в пакете MathCAD.

Читатель, заинтересованный в более глубоком и детальном изучении курса численных методов, должен обратиться к более полным руководствам. Некоторые из них приведены в списке литературы. Так же можно обратиться к Internet ресурсам. Например, современные достижения в этой области можно увидеть на Web – сайтах Института вычислительной математики РАН: www.inm.ras.ru и научного журнала «Вычислительные методы и программирование. Новые вычислительные технологии» - www.num-meth.srcc.msu.ru.

Введение

Математическое моделирование и вычислительный эксперимент

1. Схема вычислительного эксперимента. Эффективное решение крупных естественнонаучных и народнохозяйственных задач сейчас невозможно без применения быстродействующих электронно-вычислительных машин (ЭВМ). В настоящее время выработалась технология исследования сложных проблем, основанная на построении и анализе с помощью ЭВМ математических моделей изучаемого объекта. Такой метод исследования называют *вычислительным экспериментом.*

Пусть, например, требуется исследовать какой-то физический объект, явление, процесс. Тогда схема вычислительного эксперимента выглядит так, как показано на рисунке 1. Формулируются основные законы, управляющие данным объектом исследования (I) и строится соответствующая *математическая модель* (II), представляющая обычно запись этих законов в форме системы уравнений (алгебраических, дифференциальных, интегральных и т. д.).

Рисунок 1 – Этапы построения и анализа с помощью ЭВМ математической модели объекта

При выборе физической и, следовательно, математической модели мы пренебрегаем факторами, не оказывающими существенного влияния на ход изучаемого процесса. Типичные математические модели, соответствующие физическим явлениям, формулируются в виде уравнений математической физики. Большинство реальных процессов описывается нелинейными уравнениями и лишь в первом приближении (при малых значениях параметров, малых отклонениях от равновесия и др.) эти уравнения можно заменить линейными.

После того как задача сформулирована в математической форме, необходимо найти ее решение. Но что значит решить математическую задачу? Только в исключительных случаях удается найти решение в явном виде, например в виде ряда. Иногда утверждение «задача решена» означает, что доказано существование и единственность решения. Ясно, что этого недостаточно для практических приложений. Необходимо еще изучить качественное поведение решения и найти те или иные количественные характеристики.

Именно на этом этапе требуется привлечение ЭВМ и, как следствие, развитие численных методов (см. III на рис. 1). Под *численным методом* здесь понимается такая интерпретация математической модели («дискретная модель»), которая доступна для реализации на ЭВМ. Например, если математическая модель представляет собой дифференциальное уравнение, то численным методом может быть аппроксимирующее его разностное уравнение совместно с алгоритмом, позволяющим отыскать решение этого разностного уравнения. Результатом реализации численного метода на ЭВМ является число или таблица чисел. Отметим, что в настоящее время помимо собственно численных методов имеются также методы, которые позволяют проводить на ЭВМ аналитические выкладки. Однако аналитические методы для ЭВМ не получили пока достаточно широкого распространения.

Чтобы реализовать численный метод, необходимо составить программу для ЭВМ (см. IV на рис. 1) или воспользоваться готовой программой.

После отладки программы наступает этап проведения вычислений и анализа результатов (V). Полученные результаты изучаются с точки зрения их соответствия исследуемому явлению и, при необходимости, вносятся исправления в численный метод и уточняется математическая модель.

Такова в общих чертах схема вычислительного эксперимента. Его основу составляет триада: *модель — метод* (*алгоритм*) — *программа*. Опыт решения крупных задач показывает, что метод математического моделирования и вычислительный эксперимент соединяют в себе преимущества традиционных теоретических и экспериментальных методов исследования. Можно указать такие крупные области применения вычислительного эксперимента, как энергетика, аэрокосмическая техника, обработка данных натурного эксперимента, совершенствование технологических процессов.

2. Вычислительный алгоритм. Предметом данной книги является изложение вопросов, отражающих этапы III, IV, V вычислительного эксперимента. Таким образом, здесь не обсуждаются исходные задачи и их математическая постановка.

Необходимо подчеркнуть, что процесс исследования исходного объекта методом математического моделирования и вычислительного эксперимента неизбежно носит приближенный характер, потому что на каждом этапе вносятся те или иные погрешности. Так, построение математической модели связано с упрощением исходного явления, недостаточно точным заданием коэффициентов уравнения и других входных данных. По отношению к численному методу, реализующему данную математическую модель, указанные погрешности являются *неустранимыми,* поскольку они неизбежны в рамках данной модели.

При переходе от математической модели к численному методу возникают погрешности, называемые *погрешностями метода*. Они связаны с тем, что всякий численный метод воспроизводит исходную математическую модель приближенно.

Наиболее типичными погрешностями метода являются *погрешность дискретизации* и *погрешность округления*.

Поясним причины возникновения таких погрешностей.

Обычно построение численного метода для заданной математической модели разбивается на два этапа: а) формулирование дискретной задачи, б) разработка вычислительного алгоритма, позволяющего отыскать решение дискретной задачи. Например, если исходная математическая задача сформулирована в виде системы дифференциальных уравнений, то для численного решения необходимо заменить ее системой конечного, может быть, очень большого числа линейных или разностных алгебраических уравнений. В этом случае говорят, что проведена *дискретизация исходной математической задачи.* Простейшим примером дискретизации является построение *разностной схемы,* путем замены дифференциальных выражений конечно-разностными отношениями. В общем случае дискретную модель можно рассматривать как конечномерный аналог исходной математической задачи. Ясно, что решение дискретизированной задачи отличается от решения исходной задачи. Разность соответствующих решений и называется *погрешностью дискретизации.* Обычно дискретная модель зависит от некоторого параметра (или множества параметров) дискретизации, при стремлении которого к нулю должна стремиться к нулю и погрешность дискретизации. При этом число алгебраических уравнений, составляющих дискретную модель, неограниченно возрастает. В случае разностных методов таким параметром является шаг сетки.

Как уже отмечалось, дискретная модель представляет собой систему большого числа алгебраических уравнений. Невозможно найти решение такой системы точно и в явном виде. Поэтому приходится использовать тот или иной численный алгоритм решения системы алгебраических уравнений. Входные данные этой системы, а именно коэффициенты и правые части, задаются в ЭВМ не точно, а с округлением.

В процессе работы алгоритма погрешности округления обычно накапливаются, и в результате решение, полученное на ЭВМ, будет отличаться от точного решения дискре-

тизированной задачи. Результирующая погрешность называется *погрешностью округления* (иногда ее называют *вычислительной погрешностью*).

Величина этой погрешности определяется двумя факторами: точностью представления вещественных чисел в ЭВМ и чувствительностью данного алгоритма к погрешностям округления.

Алгоритм называется *устойчивым,* если в процессе его работы вычислительные погрешности возрастают незначительно, и *неустойчивым* — в противоположном случае. При использовании неустойчивых вычислительных алгоритмов накопление погрешностей округления приводит в процессе счета к переполнению арифметического устройства ЭВМ.

Итак, следует различать погрешности модели, метода и вычислительную. Какая же из этих трех погрешностей является преобладающей? Ответ здесь неоднозначен. Видимо, типичной является ситуация, возникающая при решении задач математической физики, когда погрешность модели значительно превышает погрешность метода, а погрешностью округления в случае устойчивых алгоритмов можно пренебречь по сравнению с погрешностью метода. С другой стороны, при решении, например, систем обыкновенных дифференциальных уравнений возможно применение столь точных методов, что их погрешность будет сравнима с погрешностью округления. В общем случае нужно стремиться, чтобы все указанные погрешности имели один и тот же порядок. Например, нецелесообразно пользоваться разностными схемами, имеющими точность 10^{-6}, если коэффициенты исходных уравнений задаются с точностью 10^{-2}.

3. Требования к вычислительным методам. Одной и той же математической задаче можно поставить в соответствие множество различных дискретных моделей. Однако далеко не все из них пригодны для практической реализации.

Вычислительные алгоритмы, предназначенные для быстродействующих ЭВМ, должны удовлетворять многообразным и зачастую противоречивым требованиям.

Попытаемся здесь сформулировать основные из этих требований в общих чертах.

Можно выделить две группы требований к численным методам. Первая группа связана с адекватностью дискретной модели исходной математической задаче, и вторая группа – с реализуемостью численного метода на ЭВМ.

К первой группе относятся такие требования, как сходимость численного метода, выполнение дискретных аналогов законов сохранения, качественно правильное поведение решения дискретной задачи.

Поясним эти требования. Предположим, что дискретная модель математической задачи представляет собой систему большого, но конечного числа алгебраических уравнений. Обычно, чем точнее мы хотим получить решение, тем больше уравнений приходится брать. Говорят, что численный метод *сходится,* если при неограниченном увеличении числа уравнений решение дискретной задачи стремится к решению исходной задачи.

Поскольку реальная ЭВМ может оперировать лишь с конечным числом уравнений, на практике сходимость, как правило, не достигается. Поэтому важно уметь оценивать погрешность метода в зависимости от числа уравнений, составляющих дискретную модель. По этой же причине стараются строить дискретную модель таким образом, чтобы она правильно отражала качественное поведение решения исходной задачи даже при сравнительно небольшом числе уравнений.

Например, дискретной моделью задачи математической физики может быть разностная схема. Для ее построения область изменения независимых переменных заменяется дискретным множеством точек – *сеткой,* а входящие в исходное уравнение производные заменяются, на сетке, конечно-разностными отношениями. В результате получаем систему алгебраических уравнений относительно значений искомой функции в точках сетки.

Число уравнений этой системы равно числу точек сетки. Известно, что дифференциальные уравнения математической физики являются следствиями интегральных законов сохранения. Поэтому естественно требовать, чтобы для

разностной схемы выполнялись аналоги таких законов сохранения. Разностные схемы, удовлетворяющие этому требованию, называются *консервативными.* Оказалось, что при одном и том же числе точек сетки консервативные разностные схемы более правильно отражают поведение решения исходной задачи, чем неконсервативные схемы.

Сходимость численного метода тесно связана с его корректностью. Предположим, что исходная математическая задача поставлена корректно, т.е. ее решение существует, единственно и непрерывно зависит от входных данных. Тогда дискретная модель этой задачи должна быть построена таким образом, чтобы свойство корректности сохранилось. Таким образом, в понятие *корректности численного метода* включаются свойства однозначной разрешимости соответствующей системы уравнений и ее устойчивости по входным данным. Под *устойчивостью* понимается непрерывная зависимость решения от входных данных, равномерная относительно числа уравнений, составляющих дискретную модель.

Вторая группа требований, предъявляемых к численным методам, связана с возможностью реализации данной дискретной модели на данной ЭВМ, т. е. с возможностью получить на ЭВМ решение соответствующей системы алгебраических уравнений за приемлемое время. Основным препятствием для реализации корректно поставленного алгоритма является ограниченный объем оперативной памяти ЭВМ и ограниченные ресурсы времени счета. Реальные вычислительные алгоритмы должны учитывать эти обстоятельства, т. е. они должны быть экономичными как по числу арифметических действий, так и по требуемому объему памяти.

Численные методы алгебры и анализа

1 Решение систем линейных алгебраических уравнений

Рассмотрим систему линейных алгебраических уравнений:

$$\begin{cases} a_{11}x_1+a_{12}x_2+...+a_{1m}x_m=b_1 \\ a_{21}x_1+a_{22}x_2+...+a_{2m}x_m=b_2 \\ \cdots\cdots\cdots \\ a_{m1}x_1+a_{m2}x_2+...+a_{mm}x_m=b_m \end{cases} \qquad (1.1)$$

или в матричной форме:

$$\boldsymbol{Ax}=\boldsymbol{b}, \qquad (1.2)$$

где: $\boldsymbol{A}=\{a_{ij}\}$ квадратная матрица размерности $(m\times m,)$;
$\boldsymbol{x}=(x_1,\ldots.,\ x_m)^T$; T – операция транспонирования;
$\boldsymbol{b}=(b_1,\ldots.,b_m)^T$; $det\boldsymbol{A}\neq 0$.

Предположим, что определитель матрицы $\boldsymbol{A}$ не равен нулю. Тогда решение $\boldsymbol{x}$ существует и единственно. На практике встречаются системы, имеющие большой порядок. Методы решения системы (1.1) делятся на две группы:

1) прямые (точные методы);

2) итерационные методы (приближенные).

1.1 Точные методы

В точных методах решение $\boldsymbol{x}$ находится за конечное число действий, но из-за погрешности округления и их накопления прямые методы можно назвать точными, только отвлекаясь от погрешностей округления.

1.1.1 Метод Гаусса

Вычисления с помощью метода Гаусса (который называют также методом последовательного исключения неизвестных) состоят из двух основных этапов: прямого хода и обратного хода. Прямой ход метода заключается в последовательном исключении неизвестных из системы для преобразования ее к эквивалентной системе с треугольной

матрицей. На этапе обратного хода производят вычисления значений неизвестных. Рассмотрим простейший вариант метода Гаусса, называемый схемой единственного деления.

Прямой ход метода

1-й шаг. Предположим, что $a_{11} \neq 0$. Поделим первое уравнение на этот элемент, который назовем *ведущим* элементом первого шага :

$$x_1 + c_{12}x_2 + ... + c_{1m}x_m = y_1. \qquad (1.3)$$

Остальные уравнения системы (1.1) запишем в виде

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{im}x_m = b_i, \qquad (1.4)$$

где $i = \overline{2,m}$.

Уравнение (1.3) умножаем на a_{i1} и вычитаем из i-го уравнения системы (1.4). Это позволит обратить в нуль коэффициенты при x_1 во всех уравнениях, кроме первого.

Получим эквивалентную систему вида:

$$\begin{cases} x_1 + c_{12}x_2 + ... + c_{1m}x_m = y_1 \\ \quad a_{22}^{(1)}x_2 + ... + a_{2m}^{(1)}x_1 = b_2^{(1)} \\ \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\ \quad a_{m2}^{(1)}x_2 + ... + a_{mm}^{(1)}x_1 = b_m^{(1)} \end{cases}. \qquad (1.5)$$

$$\begin{aligned} a_{ij}^{(1)} &= a_{ij} - c_{1j}a_{i1} \\ b_i^{(1)} &= b_i - y_1 a_{i1} \end{aligned},$$

где $i,j = \overline{2,m}$. Система (1.5) имеет матрицу вида:

$$\begin{pmatrix} 1 & x & ... & x \\ 0 & x & ... & x \\ ... & ... & ... & ... \\ 0 & x & ... & x \end{pmatrix}.$$

Дальше работаем с укороченной системой, т.к. x_1 входит только в 1-ое уравнение

$$\begin{cases} a_{22}^{(1)} x_2 + ... + a_{2m}^{(1)} x_m = b_2^{(1)} \\ . \quad . \quad . \quad . \quad . \quad . \quad . \\ a_{m2}^{(1)} x_2 + ... + a_{mm}^{(1)} x_m = b_m^{(1)} \end{cases}.$$

2-й шаг. На этом шаге исключаем неизвестное x_2 из уравнений с номерами i=3,4,…,m. Если ведущий элемент второго шага $a_{22}^{(1)} \neq 0$, то из укороченной системы аналогично исключаем неизвестное x_2 и получаем матрицу коэффициентов такого вида:

$$\begin{pmatrix} 1 & x & x & ... & x \\ 0 & 1 & x & ... & x \\ 0 & 0 & x & ... & x \\ ... & ... & ... & ... & ... \\ 0 & 0 & x & ... & x \end{pmatrix}.$$

Аналогично повторяем указанные действия для неизвестных $x_3, x_4, \ldots, x_{m-1}$ и приходим к системе :

$$\begin{cases} x_1 + c_{12} x_2 + ... + c_{1m} x_m = y_1 \\ \qquad x_2 + ... + c_{2m} x_m = y_2 \\ \quad . \quad . \quad . \quad . \quad . \quad . \quad .. \\ x_{m-1} + c_{m-1,m} x_m = y_{m-1} \\ \qquad\qquad c_{mm} x_m = y_m \end{cases} \qquad (1.6)$$

Эта система с верхней треугольной матрицей:

$$\begin{pmatrix} 1 & x & x & ... & x & x \\ 0 & 1 & x & ... & x & x \\ 0 & 0 & 1 & x & ... & x \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & 1 & x \\ 0 & 0 & 0 & ... & ... & x \end{pmatrix}.$$

Обратный ход метода. Из последнего уравнения системы (1.6) находим x_m, из предпоследнего x_{m-1}, …, из первого уравнения – x_1.

Общая формула для вычислений:

$$x_m = y_m / c_{mm},$$

$$x_i = y_i - \sum_{j=i+1}^{m} c_{ij} x_j \text{, } (i = m\text{-}1, \ldots, 1).$$

Для реализации метода Гаусса требуется примерно $(1/3)m^3$ арифметических операций, причем большинство из них приходится на прямой ход.

Ограничение метода единственного деления заключается в том, что ведущие элементы на k-ом шаге исключения не равны нулю, т.е. $a_{kk}^{k-1} \neq 0$.

Но если ведущий элемент близок к нулю, то в процессе вычисления может накапливаться погрешность. В этом случае на каждом шаге исключают не x_k, а x_j (при $j \neq k$). Такой подход называется методом выбора главного элемента. Для этого выбирают неизвестные x_j с наибольшим по абсолютной величине коэффициентом либо в строке, либо в столбце, либо во всей матрице. Для его реализации требуется $\frac{m(m^2+3m-1)}{3}$ – арифметических действий.

1.1.2 Связь метода Гаусса с разложением матрицы на множители. Теорема об *LU* разложении.

Пусть дана система $\boldsymbol{Ax=b}$ (1.1), которая при прямом ходе преобразуется в эквивалентную систему (1.6) и запишем ее в виде

$$\boldsymbol{Cx=y}, \qquad (1.6^*)$$

где $\boldsymbol{C}$ – верхняя треугольная матрица с единицами на главной диагонали, полученная из (1.6) делением последнего уравнения системы на c_{mm}.

Как связаны в системе (1.1) элементы $\boldsymbol{b}$ и элементы $\boldsymbol{y}$ из (1.6*)?

Если внимательно посмотреть на прямой ход метода Гаусса, то можно увидеть, что

$$b_1 = a_{11} y_1$$
$$b_2 = a_{21} y_1 + a_{22}^{(1)} y_2 .$$

Для произвольного j имеем

$$b_j = d_{j1}y_1 + d_{j2}y_2 + ... + d_{jj}y_i, \qquad (1.7)$$

где $j = \overline{1,m}$, d_{ji} – числовые коэффициенты:

$$d_{jj} = a_{jj}^{(j-1)}. \qquad (1.8)$$

Можно записать систему:

$$\boldsymbol{b} = \boldsymbol{D}\boldsymbol{y},$$

где $\boldsymbol{D}$ – нижняя треугольная матрица с элементами $a_{jj}^{(j-1)}$ на главной диагонали $(j = \overline{1,m}, a_{11}^{(0)} = a_{11})$.

В связи с тем, что в методе Гаусса угловые коэффициенты не равны нулю $a_{jj}^{(j-1)} \neq 0$, то на главной диагонали матрицы $\boldsymbol{D}$ стоят не нулевые элементы. Следовательно, эта матрица имеет обратную, тогда $\boldsymbol{y} = \boldsymbol{D}^{-1}\boldsymbol{b}$, $\boldsymbol{C}\boldsymbol{x} = \boldsymbol{D}^{-1}\boldsymbol{b}$.

Тогда

$$\boldsymbol{D} \times \boldsymbol{C}\boldsymbol{x} = \boldsymbol{b}. \qquad (1.9)$$

В результате использования метода Гаусса, получили разложение матрицы $\boldsymbol{A}$ на произведение двух матриц

$$\boldsymbol{A} = \boldsymbol{D} \times \boldsymbol{C},$$

где $\boldsymbol{D}$ – нижняя треугольная матрица, у которой элементы на главной диагонали не равны нулю, а $\boldsymbol{C}$ – верхняя треугольная матрица с единичной диагональю.

Таким образом, если задана матрица $\boldsymbol{A}$ и вектор $\boldsymbol{b}$, то в методе Гаусса сначала производится разложение этой матрицы $\boldsymbol{A}$ на произведение $\boldsymbol{D}$ и $\boldsymbol{C}$, а затем последовательно решаются две системы:

$$\begin{aligned} \boldsymbol{D}\boldsymbol{y} &= \boldsymbol{b}, \\ \boldsymbol{C}\boldsymbol{x} &= \boldsymbol{y}. \end{aligned} \qquad (1.10)$$

Из последней системы находят искомый вектор $\boldsymbol{x}$. При этом разложение матрицы $\boldsymbol{A}$ на произведение $\boldsymbol{CD}$ – есть прямой ход метода Гаусса, а решение систем (1.10) обратный ход. Обозначим нижнюю треугольную матрицу через $\boldsymbol{L}$, верхнюю треугольную матрицу – $\boldsymbol{U}$.

Теорема об LU разложении

Введем обозначения: Δ_j – угловой минор порядка j матрицы $\boldsymbol{A}$, т.е.

$$\Delta_1 = a_{11},$$

$$\Delta_2 = det\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix},$$

. . . . . .

$$\Delta_m = det(\boldsymbol{A}).$$

Теорема. *Пусть все угловые миноры матрицы* $\boldsymbol{A}$ *не равны нулю* ($\Delta_j \neq 0$ *для* $j = \overline{1,m}$). *Тогда матрицу* $\boldsymbol{A}$ *можно представить единственным образом в виде произведения* $\boldsymbol{A} = \boldsymbol{L} * \boldsymbol{U}$.

Идея доказательства. Рассмотрим матрицу $\boldsymbol{A}$ второго порядка и будем искать разложение этой матрицы в виде $\boldsymbol{L}$ и $\boldsymbol{U}$.

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} * \begin{pmatrix} 1 & u_{12} \\ 0 & 1 \end{pmatrix}.$$

Сопоставляя эти два равенства, определяем элементы матриц $\boldsymbol{L}$ и $\boldsymbol{U}$ (перемножим и приравняем неизвестные). Система имеет единственное решение. Методом математической индукции сказанное можно обобщить для матрицы размерности $m \times m$.

Следствие. Метод Гаусса (схему единственного деления) можно применять только в том случае, когда угловые миноры матрицы $\boldsymbol{A}$ не равны нулю.

1.1.3 Метод Гаусса с выбором главного элемента

Может оказаться так, что система (1.1) имеет единственное решение, хотя какой либо из миноров матрицы $\boldsymbol{A}$ равен нулю. Заранее неизвестно, что все угловые миноры матрицы $\boldsymbol{A}$ не равны нулю. В этом случае можно использовать метод Гаусса с выбором главного элемента.

1. Выбор главного элемента по столбцу, когда на k-ом шаге исключения в качестве главного элемента выбирают максимальный по модулю коэффициент при неизвестном x_k в уравнениях с номерами $i = k, k+1, \ldots, m$. Затем уравнение,

соответствующее выбранному коэффициенту, меняют местами с k-ым уравнением системы, чтобы ведущий элемент занял место коэффициента a_{kk}^{k-1}.После перестановки исключение неизвестного x_k выполняют как в схеме единственного деления.

ПРИМЕР. Пусть дана система второго порядка

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

Предположим, что $|a_{21}|>|a_{11}|$, тогда переставим уравнения

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

и применяем первый шаг прямого хода метода Гаусса. В этом случае имеет место перенумерация строк.

2. Выбор главного элемента по строке, т.е. производится перенумерация неизвестных системы.

При $|a_{12}|>|a_{11}|$, на первом шаге вместо неизвестного x_1 исключают x_2:

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

К этой системе применяем первый шаг прямого хода метода Гаусса.

3. Поиск главного элемента по всей матрице заключается в совместном применении методов 1 и 2. Всё это приводит к уменьшению вычислительной погрешности, но может замедлить процесс решения задачи.

1.1.4 Метод Холецкого (метод квадратных корней)

Пусть дана система

$$\boldsymbol{Ax=b}, \qquad (1.11)$$

где $\boldsymbol{A}$ – симметричная положительно определенная матрица.

Тогда решение системы (1.11) проводится в два этапа:

1. Симметричная матрица $\boldsymbol{A}$ представляется как произведение двух матриц

$$\boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{L}^T.$$

Рассмотрим метод квадратных корней на примере системы 4-го порядка:

$$\begin{pmatrix} a_{11} & \dots & a_{14} \\ \dots & \dots & \dots \\ a_{41} & \dots & a_{44} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} * \begin{pmatrix} l_{11} & l_{21} & l_{31} & l_{41} \\ 0 & l_{22} & l_{32} & l_{42} \\ 0 & 0 & l_{33} & l_{34} \\ 0 & 0 & 0 & l_{44} \end{pmatrix}.$$

Перемножаем матрицы в правой части разложения и сравниваем с элементами в левой части:

$$l_{11} = \sqrt{a_{11}}\,,\ l_{21} = \frac{a_{12}}{\sqrt{a_{11}}},\ l_{31} = \frac{a_{13}}{\sqrt{a_{11}}},\ l_{41} = \frac{a_{14}}{\sqrt{a_{11}}},\ l_{22} = \sqrt{a_{22} - l_{21}^2}\,,$$

$$l_{32} = \frac{a_{32} - l_{21} l_{31}}{l_{22}},\ l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2}\,,\ l_{44} = \sqrt{a_{44} - l_{41}^2 - l_{42}^2 - l_{43}^2}\,.$$

2. Решаем последовательно две системы

$$\boldsymbol{L}\boldsymbol{y} = \boldsymbol{b},$$
$$\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{y}.$$

Замечания

1) Под квадратным корнем может получиться отрицательное число, следовательно в программе необходимо предусмотреть использование правил действия с комплексными числами.

2) Возможно переполнение, если угловые элементы близки к нулю.

1.2 Итерационные методы решений систем алгебраических уравнений

Итерационные методы обычно применяются для решения систем большой размерности и они требуют приведения исходной системы к специальному виду.

Суть итерационных методов заключается в том, что решение $\boldsymbol{x}$ системы (1.1) находится как предел последовательности $\lim\limits_{n\to\infty} x(n)$.

Так как за конечное число итераций предел не может быть достигнут, то задаётся малое число ε – точность, и последовательные приближения вычисляют до тех пор, пока не будет выполнено неравенство

$$\left\| \boldsymbol{x}^{n} - \boldsymbol{x}^{n-1} \right\| < \varepsilon,$$

где $n=n(\varepsilon)$ – функция ε, $\|\boldsymbol{x}\|$ – норма вектора.

Определения основных норм в пространстве векторов и матриц.

Для вектора $\boldsymbol{x}=(x_1,x_2,\ldots,x_n)^{\mathrm{T}}$ нормы вычисляются по следующим формулам:

$$\|\boldsymbol{x}\|_1 = \max_{1\le i\le n} |x_i|;$$

$$\|\boldsymbol{x}\|_2 = \sum_{i=1}^{n} |x_i|;$$

$$\|\boldsymbol{x}\|_3 = \sqrt{\sum_{i=1}^{n} |x_i|^2}.$$

Согласованные с ними нормы в пространстве матриц:

$$\|\boldsymbol{A}\|_1 = \max_{1\le i\le n} \left(\sum_{j=1}^{n} |a_{ij}| \right);$$

$$\|\boldsymbol{A}\|_2 = \max_{1\le j\le n} \left(\sum_{i=1}^{n} |a_{ij}| \right);$$

$\|\boldsymbol{A}\|_3 = \sqrt{\sum_{i,j=1}^{n} |a_{ij}|^2}$ – величина, называемая евклидовой нормой матрицы $\boldsymbol{A}$.

Прямые методы рассчитаны для решения систем, порядок которых не больше 100, иначе для практических вычислений используются итерационные методы.

1.2.1 Метод Якоби (простых итераций)

Исходную систему (1.11)

$$\boldsymbol{Ax=b}$$

преобразуем к виду:

$$x_i = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j - \sum_{j=i+1}^{m} \frac{a_{ij}}{a_{ii}} x_j + \frac{b_i}{a_{ii}}, \qquad (1.12)$$

где $i=1,2,\ldots,m$; $a_{ii} \neq 0$.

Первая сумма равна нулю, если верхний предел суммирования меньше нижнего.

Так (1.12) при $i=1$ имеет вид

$$x_1 = -\sum_{j=2}^{m} \frac{a_{1j}}{a_{11}} x_j + \frac{b_1}{a_{11}}.$$

По методу Якоби x_i^{n+1} ($n+1$ приближение x_i) ищем по формуле

$$x_i^{n+1} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^n - \sum_{j=i+1}^{m} \frac{a_{ij}}{a_{ii}} x_j^n + \frac{b_i}{a_{ii}}. \qquad (1.13)$$

где n – номер итерации $(0,1,\ldots)$; $i=\overline{1,m}$.

Итерационный процесс (1.13) начинается с начальных значений x_i^0, которые в общем случае задаются произвольно, но предпочтительнее за x_i^0 взять свободные члены исходной системы.

Условие окончания счета:

$$\max_i \left| x_i^{n+1} - x_i^n \right| < \varepsilon, \qquad \text{где } i=\overline{1,m}.$$

1.2.2 Метод Зейделя

Система (1.11) преобразуется к виду (1.12) и организуется итерационная процедура, где неизвестные x_i на $n+1$ шаге определяются по формулам

$$x_i^{n+1} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{n+1} - \sum_{j=i+1}^{m} \frac{a_{ij}}{a_{ii}} x_j^n + \frac{b_i}{a_{ii}}. \qquad (1.14)$$

Например,

$$x_1^{n+1} = -\sum_{j=2}^{m} \frac{a_{1j}}{a_{11}} x_j^n + \frac{b_1}{a_{11}}, \qquad (1.15)$$

$$x_2^{n+1} = -\sum_{j=3}^{m} \frac{a_{2j}}{a_{22}} x_j^n + \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}} x_1^{n+1}, \qquad (1.16)$$

и так далее.

Итерационные процессы (1.13) и (1.14) сходятся, если норма матрицы $\boldsymbol{A}$ ($\boldsymbol{A}$ – матрица коэффициентов при неизвестных в правой части систем (1.13) и (1.14)) удовлетворяет условию:

$$||\boldsymbol{A}||<1.$$

1.2.3 Матричная запись методов Якоби и Зейделя

Исходную матрицу системы (1.11) представим в виде суммы трёх матриц

$$\boldsymbol{A}=\boldsymbol{A}_1+\boldsymbol{D}+\boldsymbol{A}_2,$$

где $\boldsymbol{D}$ – диагональная матрица;

$\boldsymbol{D}=diag[a_{11}a_{22}...a_{mm}]$;

$\boldsymbol{A}_1$ – нижняя треугольная матрица;

$\boldsymbol{A}_2$ – верхняя треугольная матрица.

Пример: Дана матрица размерности (3×3):

$$\underbrace{\begin{pmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & a_{32} & 0 \end{pmatrix}}_{\boldsymbol{A}_1}+\underbrace{\begin{pmatrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{pmatrix}}_{\boldsymbol{A}_2}+\underbrace{\begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{pmatrix}}_{\boldsymbol{D}}=A.$$

Тогда исходную систему (1.11) можно записать в виде

$$\boldsymbol{x}=-\boldsymbol{D}^{-1}\boldsymbol{A}_1\,\boldsymbol{x}-\boldsymbol{D}^{-1}\boldsymbol{A}_2\,\boldsymbol{x}+\boldsymbol{D}^{-1}\,\boldsymbol{b}.$$

Тогда метод Якоби можно записать в виде:

$$\boldsymbol{x}^{n+1}=-\boldsymbol{D}^{-1}\boldsymbol{A}_1\boldsymbol{x}^{n}-\boldsymbol{D}^{-1}\boldsymbol{A}_2\boldsymbol{x}^{n}+\boldsymbol{D}^{-1}\boldsymbol{b}$$

или

$$\boldsymbol{D}\,\boldsymbol{x}^{n+1}+(\boldsymbol{A}_1+\boldsymbol{A}_2)\,\boldsymbol{x}^{n}=\boldsymbol{b}\,. \qquad (1.17)$$

В матричной форме метод Зейделя будет выглядеть:

$$\boldsymbol{x}^{n+1}=-\boldsymbol{D}^{-1}\boldsymbol{A}_1\boldsymbol{x}^{n+1}-\boldsymbol{D}^{-1}\boldsymbol{A}_2\boldsymbol{x}^{n}+\boldsymbol{D}^{-1}\boldsymbol{b}$$

или

$$(\boldsymbol{D}+\boldsymbol{A}_1)\,\boldsymbol{x}^{n+1}+\boldsymbol{A}_2\,\boldsymbol{x}^{n}=\boldsymbol{b}\,. \qquad (1.18)$$

Преобразуем формулы (1.17) и (1.18):

$$\boldsymbol{D}(\boldsymbol{x}^{n+1}-\boldsymbol{x}^{n})+\boldsymbol{A}\boldsymbol{x}^{n}=\boldsymbol{b}, \qquad (1.19)$$

$$(\boldsymbol{D}+\boldsymbol{A}_1)(\boldsymbol{x}^{n+1}-\boldsymbol{x}^{n})+\boldsymbol{A}\boldsymbol{x}^{n}=\boldsymbol{b}. \qquad (1.20)$$

Из (1.19) и (1.20) видно, что если итерационный метод сходится, то он сходится к точному решению. Иногда при решении задач большой размерности, в итерационные методы вводятся числовые параметры, которые могут зависеть от номера итерации.

Пример для метода Якоби.

$$\boldsymbol{D}\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^{n}}{t_{n+1}}+\boldsymbol{A}\boldsymbol{x}^{n}=\boldsymbol{b},$$

где t – числовой параметр.

Возникают вопросы:

1) При каких значениях t сходимость будет наиболее быстрой?

2) При каких значениях t метод сходится?

На примере двух методов просматривается вывод о том, что одни и те же методы можно записывать несколькими способами. Поэтому вводят каноническую (стандартную) форму записи:

$$\boldsymbol{D}_{n+1}\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^{n}}{\boldsymbol{t}_{n+1}}+\boldsymbol{A}\boldsymbol{x}^{n}=\boldsymbol{b}. \qquad (1.21)$$

Формула (1.21) получена путем объединения (1.19) и (1.20).

Матрица $\boldsymbol{D}_{n+1}$ здесь задает тот или иной метод. Если существует обратная матрица к этой матрице, то из последней системы мы можем найти все неизвестные.

1. Метод (1.21) – явный, если матрица $\boldsymbol{D}_n$ совпадает с единичной матрицей и неявный – в противном случае.

2. Метод (1.21) – стационарный, если матрица $\boldsymbol{D}_{n+1}=\boldsymbol{D}$, и параметр t не зависит от номера итерации и нестационарный – в противном случае.

1.2.4 Метод Ричардсона

Явный метод с переменным параметром t:

$$\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^{n}}{t_{n+1}}+\boldsymbol{A}\boldsymbol{x}^{n}=\boldsymbol{b}, \qquad (1.21а)$$

называется методом Ричардсона.

1.2.5 Метод верхней релаксации (обобщённый метод Зейделя)

$$(\boldsymbol{D}+\omega\boldsymbol{A}_1)\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^{n}}{\omega}+\boldsymbol{A}\boldsymbol{x}^{n}=\boldsymbol{b}, \qquad (1.21б)$$

где ω – числовой параметр.

Если матрица $\boldsymbol{A}$ – симметричная и положительно определена, то последний метод сходится при ($0 < \omega < 2$). Последнюю формулу запишем в следующем виде:

$$(\boldsymbol{E}+\omega\boldsymbol{D}^{-1}\boldsymbol{A}_1)\boldsymbol{x}^{n+1}=((1-\omega)\boldsymbol{E}-\omega\boldsymbol{D}^{-1}\boldsymbol{A}_2)\boldsymbol{x}^{\mathrm{n}}+\omega\boldsymbol{D}^{-1}\boldsymbol{b}, \qquad (1.22)$$

где $\boldsymbol{E}$ – единичная матрица.

Тогда для вычисления неизвестных x_i ($i=\overline{1,m}$) можно записать итерационную процедуру в виде:

$$x_{\mathrm{i}}^{\mathrm{n+1}}+\omega\sum_{j=1}^{i-1}\frac{a_{ij}}{a_{ii}}x_j^{n+1}=(1-\omega)x_i^{n}-\omega\sum_{j=i+1}^{m}\frac{a_{ij}}{a_{ii}}x_j^{n}+\omega\frac{b_i}{a_{ii}}. \qquad (1.23)$$

Например, для x_1 это будет такое выражение:

$$x_1^{n+1}=(1-\omega)x_1^{n}-\omega\sum_{j=2}^{m}\frac{a_{1j}}{a_{11}}x_j^{n}+\omega\frac{b_1}{a_{11}}.$$

1.2.6 Сходимость итерационных методов

Рассмотрим систему

$$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b},$$

где $\boldsymbol{A}$ – невырожденная действительная матрица.

Для решения системы рассмотрим одношаговый стационарный метод

$$\boldsymbol{D}\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^{n}}{t}+\boldsymbol{A}\boldsymbol{x}^{n}=\boldsymbol{b}, \qquad (1.24)$$

при n=0,1,2….

Предположим, что задан начальный вектор решения. Тогда метод (1.24) сходится, если норма вектора

$$\left\|\boldsymbol{x}-\boldsymbol{x}^{n}\right\|_{n\to\infty}\to 0.$$

Теорема. *Условие сходимости итерационного метода.*

Пусть $\boldsymbol{A}$ *– симметричная положительно определенная матрица и выполнено условие* $\boldsymbol{D} - 0.5t\boldsymbol{A} > 0$ (*где* $t > 0$). *Тогда итерационный метод* (1.24) *сходится.*

Следствие 1. Пусть $\boldsymbol{A}$ – симметричная и положительно определенная матрица с диагональным преобладанием, то есть

$$|a_{jj}| > \sum_{\substack{i=1 \\ i \neq j}}^{m} |a_{ij}|,$$

при j=1,2,…,m. Тогда метод Якоби сходится.

Следствие 2. Пусть $\boldsymbol{A}$ – симметричная и положительно определенная матрица с диагональным преобладанием, тогда метод верхней релаксации сходится при
(0< ω<2).

Проверяется, при каком значении ω метод достигает заданной точности быстрее.

В частности, при ω = 1 метод верхней релаксации превращается в метод Зейделя, следовательно, при ω = 1 метод Зейделя сходится.

Теорема. *Итерационный метод* (*1.24*) *сходится при любом начальном векторе* $\boldsymbol{x}^0$ *тогда и только тогда, когда все собственные значения матрицы*

$$\boldsymbol{S} = \boldsymbol{E} - t\boldsymbol{D}^{-1}\boldsymbol{A}$$

по модулю меньше единицы.

2 Плохо обусловленные системы линейных алгебраических уравнений

Дана система линейных алгебраических уравнений

$$\boldsymbol{Ax}=\boldsymbol{b} \qquad (2.1)$$

Если система плохо обусловлена, то это значит, что погрешности коэффициентов матрицы $\boldsymbol{A}$ и правых частей $\boldsymbol{b}$ или же погрешности их округления сильно искажают решение системы.

Для оценки обусловленности системы вводят число обусловленности M_A

$$M_A=\| \boldsymbol{A}^{-1} \| \cdot \| \boldsymbol{A} \|.$$

Чем больше значение M_A,тем система хуже обусловлена.

Свойства числа обусловленности:

1) $M_E = 1$;

2) $M_A \geq 1$;

3) $M_A \geq |\lambda_{max}| / |\lambda_{min}|$, где $\lambda_{\max}$, $\lambda_{\min}$ – соответственно максимальное и минимальное собственные числа матрицы $\boldsymbol{A}$;

4) $M_{AB} \leq M_A * M_B$;

5) Число обусловленности матрицы $\boldsymbol{A}$ не меняется при умножении матрицы на произвольное число $\alpha \neq 0$.

Найдем выражение для полной оценки погрешности решения системы.

Пусть в системе (2.1) возмущены коэффициенты матрицы $\boldsymbol{A}$ и правая часть $\boldsymbol{b}$, т.е.

$\delta \boldsymbol{A}=\tilde{\boldsymbol{A}}-\boldsymbol{A}$, $\delta \boldsymbol{b}=\tilde{\boldsymbol{b}}-\boldsymbol{b}$, $\delta \boldsymbol{x}=\tilde{\boldsymbol{x}}-\boldsymbol{x}$.

Теорема. *Пусть матрица* $\boldsymbol{A}$ *имеет обратную матрицу, и выполняется условие* $\|\delta \boldsymbol{A}\| < \|\boldsymbol{A}^{-1}\|^{-1}$. *Тогда матрица* $\tilde{A}=\delta A+A$ *имеет обратную и справедлива следующая оценка относительной погрешности:*

$$\frac{\|\delta \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{M_A}{1-M_A \dfrac{\|\delta \boldsymbol{A}\|}{\|\boldsymbol{A}\|}} \left(\frac{\|\delta \boldsymbol{A}\|}{\|\boldsymbol{A}\|} + \frac{\|\delta \boldsymbol{b}\|}{\|\boldsymbol{b}\|} \right).$$

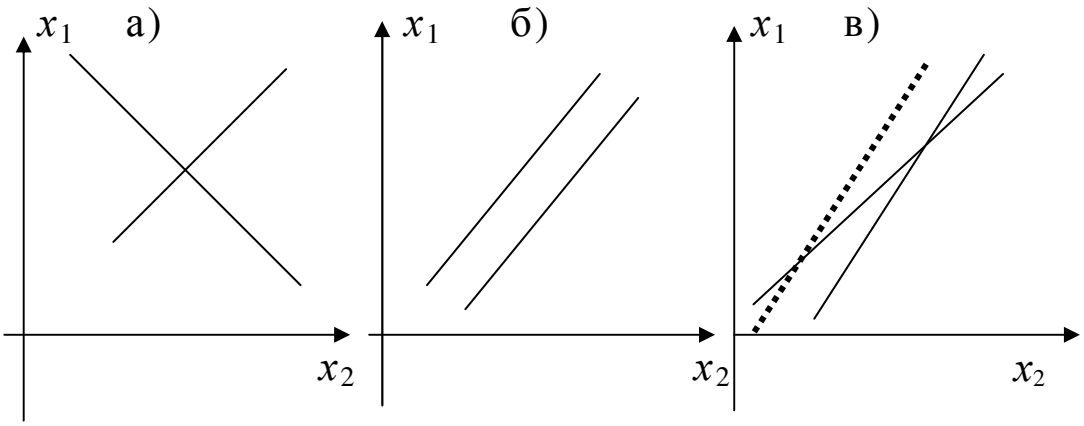


Рисунок 2 – а) система имеет единственное решение;
б) система не имеет решения;
в) система плохо обусловлена.

В случае в) малейшее возмущение системы сильно меняет положение точки пересечения прямых.

В качестве примера рассмотрим систему

$$\begin{cases} 1.03x_1 + 0.991x_2 = 2.51 \\ 0.991x_1 + 0.943x_2 = 2.41. \end{cases}$$

Решение этой системы

$$x_1 \approx 1.981$$
$$x_2 \approx 0.4735.$$

Оценим влияние погрешности правых частей на результат. Рассмотрим “возмущенную” систему с правой частью $\boldsymbol{b}^* = (2.505\ ,\ 2.415)$ и решим эту систему:

$$x_1^* \approx 2.877$$
$$x_2^* \approx -0.4629.$$

Относительная погрешность правой части

$\delta(\boldsymbol{b}) = 0.005/2.51 \approx 0.28\%$ привела к относительной погрешности решения $\delta(\boldsymbol{x}^*) = 0.9364/1.981 \approx 47.3\%$.

Погрешность возросла примерно в 237 раз. Число обусловленности системы (2.1) приблизительно равно 237.

Подобные системы называются плохо обусловленными. Возникает вопрос: какими методами можно решать такие системы?

2.1 Метод регуляризации для решения плохо обусловленных систем

Рассмотрим систему

$$\boldsymbol{Ax}=\boldsymbol{b}. \qquad (2.1)$$

Для краткости перепишем эту систему в эквивалентный форме

$$(\boldsymbol{Ax}-\boldsymbol{b},\ \ \boldsymbol{Ax}-\boldsymbol{b})=0. \qquad (2.2)$$

Для примера рассмотрим систему

$$\begin{cases} 2x_1 - x_1 = 1 \\ x_1 - 2x_2 = 2 \end{cases}.$$

Тогда ее можно представить как

$$(2x_1\text{-}x_2\text{-}1)^2+(x_1\text{-}2x_2\text{-}2)^2=0. \qquad (2.2^*)$$

Решение системы (2.2) совпадает с решением системы (2.2*).

Если коэффициенты $\boldsymbol{A}$ или $\boldsymbol{b}$ известны неточно, то решение также является не точным, поэтому вместо равенства $(\boldsymbol{Ax}-\boldsymbol{b}, \boldsymbol{Ax}-\boldsymbol{b})=0$ можем потребовать приближенного выполнения равенства $(\boldsymbol{Ax}-\boldsymbol{b}, \boldsymbol{Ax}-\boldsymbol{b})\approx 0$ и в этом виде задача становится не определенной и нужно добавить дополнительные условия.

В качестве дополнительного условия вводят требование, чтобы решение как можно меньше отклонялось от заданного $\boldsymbol{x}_0$ т.е. $(\boldsymbol{x}\text{-}\boldsymbol{x}_0,\ \boldsymbol{x}\text{-}\boldsymbol{x}_0)$ было минимальным. Следовательно, приходим к регуляризованной задаче вида

$$(\boldsymbol{Ax}\text{-}\boldsymbol{b},\ \boldsymbol{Ax}\text{-}\boldsymbol{b})+\alpha(\boldsymbol{x}\text{-}\boldsymbol{x}_0,\ \boldsymbol{x}\text{-}\boldsymbol{x}_0)=min, \qquad (2.3)$$

где $\alpha>0$.

Используя свойства скалярного произведения, выражение (2.3) перепишем в виде

$$(\boldsymbol{x},\boldsymbol{A}^T\boldsymbol{Ax})\text{-}2(\boldsymbol{x},\boldsymbol{A}^T\boldsymbol{b})+(\boldsymbol{b},\boldsymbol{b})+\alpha[(\boldsymbol{x},\boldsymbol{x})\text{-}2(\boldsymbol{x},\boldsymbol{x}_0)+(\boldsymbol{x}_0,\boldsymbol{x}_0)]=min. \qquad (2.4)$$

Варьируя $\boldsymbol{x}$ в уравнении (2.4), получим уравнение вида

$$(\boldsymbol{A}^T\boldsymbol{A}+\alpha\boldsymbol{E})\boldsymbol{x}=\boldsymbol{A}^T\boldsymbol{b}+\alpha\boldsymbol{x}_0. \qquad (2.5)$$

Система (2.5) – система линейных алгебраических уравнений, эквивалентная системе (2.1). Систему (2.5) решаем с помощью метода Гаусса или с помощью метода квадратных корней. Решая систему (2.5) найдем решение, которое зависит от числа α.

Выбор управляющего параметра α. Если α=0, то система (2.5) перейдет в плохо обусловленную систему (2.1).

Если же α– велико, то система (2.5) переходит в хорошо обусловленную систему и решение этой системы может сильно отличаться от решения системы (2.1).

Оптимальное значение α – это такое число, при котором система (2.5) удовлетворительно обусловлена.

На практике пользуются невязкой вида $\boldsymbol{r}_\alpha = \boldsymbol{A}\boldsymbol{x}_\alpha - \boldsymbol{b}$, и эту невязку сравнивают по норме с известной погрешностью правых частей $\delta\boldsymbol{b}$ и с влиянием погрешности коэффициентов матрицы $\delta\boldsymbol{A}$.

Если α– слишком велико, то $\boldsymbol{r}_\alpha >> \delta\boldsymbol{b}$ или $\delta\boldsymbol{A}$. Если α– мало, то $\boldsymbol{r}_\alpha << \delta\boldsymbol{b}$ или $\delta\boldsymbol{A}$.

Поэтому проводят серию расчетов, при различных α и в качестве оптимального значения выбирают то значение α, когда выполнено следующее условие

$$\|\boldsymbol{r}_\alpha\| \approx \|\delta\boldsymbol{b}\| + \|\delta\boldsymbol{A}\cdot\boldsymbol{x}\| \ .$$

Для выбора вектора $\boldsymbol{x}_0$ нужно знать приближенное решение или же, если приближенное решение трудно определить, то $\boldsymbol{x}_0$ =0.

2.2 Метод вращения (Гивенса)

Метод Гивенса, как и метод Гаусса состоит из прямого и обратного ходов.

Прямой ход метода. Исключаем неизвестное x_1 из всех уравнений, кроме первого. Для исключения x_1 из 2-го уравнения вычисляют числа

$$\alpha_{12} = \frac{a_{11}}{\sqrt{a_{11}^2 + a_{21}^2}}, \qquad \beta_{12} = \frac{a_{21}}{\sqrt{a_{11}^2 + a_{21}^2}},$$

где α и β такие, что $\alpha_{12}^2+\beta_{12}^2=1$, $-\beta_{12}a_{11}+\alpha_{12}a_{21}=0$.

Первое уравнение системы заменяем линейной комбинацией первого и второго уравнений с коэффициентами α_{12} и β_{12}, а второе уравнение такой же комбинацией с α_{12} и $-\beta_{12}$. В результате получим систему

$$\begin{cases} a_{11}^{(1)}x_1+a_{12}^{(1)}x_2+...+a_{1m}^{(1)}x_m=b_1^{(1)} \\ \qquad a_{22}^{(1)}x_2+...+a_{2m}^{(1)}x_m=b_2^{(1)} \\ a_{31}x_1+a_{32}x_2+...+a_{3m}x_m=b_3. \\ \qquad\qquad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ a_{m1}x_1+a_{m2}x_2+...+a_{mm}x_m=b_m \end{cases} \qquad (2.6)$$

Здесь

$a_{1j}^{(1)}=\alpha_{12}a_{1j}+\beta_{12}a_{2j}$, $a_{2j}^{(1)}=\alpha_{12}a_{2j}-\beta_{12}a_{1j}$, $b_1^{(1)}=\alpha_{12}b_1+\beta_{12}b_2$,

$b_2^{(1)}=\alpha_{12}b_2-\beta_{12}b_1$,

где $j=\overline{1,m}$.

Преобразование системы (2.1) к системе (2.6) эквивалентно умножению слева матрицы $\boldsymbol{A}$ и вектора $\boldsymbol{b}$ на матрицу $\boldsymbol{C}_{12}$ вида

$$\boldsymbol{C}_{12}=\begin{pmatrix} \alpha_{12} & \beta_{12} & 0 & ... & 0 \\ -\beta_{12} & \alpha_{12} & 0 & ... & 0 \\ 0 & 0 & 1 & ... & 0 \\ . & . & . & . & . \\ 0 & 0 & 0 & ... & 1 \end{pmatrix}.$$

Аналогично для исключения x_1 из третьего уравнения вычисляем числа

$$\alpha_{13}=\frac{a_{11}^{(1)}}{\sqrt{(a_{11}^{(1)})^2+(a_{31}^{(1)})^2}} \text{ и } \beta_{13}=\frac{a_{31}}{\sqrt{(a_{11}^{(1)})^2+(a_{31}^{(1)})^2}},$$

такие, что $\alpha_{13}^2+\beta_{13}^2=1$, $\alpha_{13}a_{31}-\beta_{13}a_{11}^{(1)}=0$.

Затем первое уравнение системы (2.6) заменяем линейной комбинацией первого и третьего уравнений с коэффициентами α_{13},β_{13}, а третье уравнение системы (2.6) заменяем линейной комбинацией тех же уравнений, но с коэффициентами α_{13} и $-\beta_{13}$. Это преобразование эквивалентно умножению слева на матрицу

$$\boldsymbol{C}_{13}=\begin{pmatrix} \alpha_{13} & 0 & \beta_{13} & 0 & ... & 0 \\ 0 & 1 & 0 & 0 & ... & 0 \\ -\beta_{13} & 0 & \alpha_{13} & 0 & ... & 0 \\ 0 & 0 & 0 & 1 & ... & 0 \\ . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & ... & 1 \end{pmatrix}.$$

Исключая неизвестное x_1 из всех последующих уравнений получим систему

$$\boldsymbol{A}^{(1)}\,\boldsymbol{x}=\boldsymbol{b}^{(1)},$$

где матрица на первом шаге $\boldsymbol{A}^{(1)}=\boldsymbol{C}_{1m}...\boldsymbol{C}_{13}\boldsymbol{C}_{12}\boldsymbol{A}$, , а вектор правых частей $\boldsymbol{b}^{(1)}=\boldsymbol{C}_{1m}...\boldsymbol{C}_{13}\boldsymbol{C}_{12}\boldsymbol{b}$.

Здесь и далее через $\boldsymbol{C}_{kj}$ обозначена матрица элементарного преобразования, отличающаяся от единичной матрицы $\boldsymbol{E}$ только четырьмя элементами.

Действие матрицы $\boldsymbol{C}_{kj}$ на вектор $\boldsymbol{x}$ эквивалентно повороту вектора $\boldsymbol{x}$ вокруг оси, перпендикулярной плоскости OX_kX_j на угол φ_{kj} такой, что

$$\alpha_{kj}=\cos\varphi_{kj},\ \beta_{kj}=\sin\varphi_{kj}.$$

Операцию умножения на матрицу $\boldsymbol{C}_{kj}$ называют плоским вращением или преобразованием Гивенса.

Первый этап состоит из m-1 шагов, в результате чего получается система

$$\begin{cases} a_{11}^{(m-1)}x_1+a_{12}^{(m-1)}x_2...+a_{1m}^{(m-1)}x_m=b_1^{(m-1)} \\ \qquad\qquad a_{22}^{(1)}x_2+...+a_{2m}^{(1)}x_m=b_2^{(1)} \\ \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ \qquad\qquad a_{m2}^{(1)}x_2+...+a_{mm}^{(1)}x_m=b_m^{(1)}. \end{cases} \qquad (2.7)$$

В матричной форме получаем $\boldsymbol{A}^{(1)}\,\boldsymbol{x}=\boldsymbol{b}^{(1)}$.

На втором этапе, состоящем из m-2 шагов, из уравнений системы (2.7) с номерами 3,4,…,m исключают неизвестное x_2. В результате получим систему

$$\begin{cases} a_{11}^{(m-1)}x_1 + a_{12}^{(m-1)}x_2 + a_{13}^{(m-1)}x_3 ... + a_{1m}^{(m-1)}x_m = b_1^{(m-1)} \\ a_{22}^{(m-1)}x_2 + a_{23}^{(m-1)}x_3 ... + a_{2m}^{(m-1)}x_m = b_2^{(m-1)} \\ a_{33}^{(2)}x_3 + ... + a_{3m}^{(2)}x_m = b_m^{(2)} \\ \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\ a_{m3}^{(2)}x_3 + ... + a_{mm}^{(2)}x_m = b_m^{(2)}. \end{cases}$$

В матричной форме получаем $\boldsymbol{A}^{(2)}\boldsymbol{x}=\boldsymbol{b}^{(2)}$, где $\boldsymbol{A}^{(2)}=\boldsymbol{C}_{2m}...\boldsymbol{C}_{24}\boldsymbol{C}_{23}\boldsymbol{A}^{(1)}$, $\boldsymbol{b}^{(2)}=\boldsymbol{C}_{2m}...\boldsymbol{C}_{24}\boldsymbol{C}_{23}\boldsymbol{b}^{(1)}$.

После завершения (m-1)-го шага придем к системе с верхней треугольной матрицей вида

$$\boldsymbol{A}^{(m-1)}\boldsymbol{x}=\boldsymbol{b}^{(m-1)},$$

где $\boldsymbol{A}^{(m-1)}=\boldsymbol{C}_{m-1,m}\boldsymbol{A}^{(m-2)}$, $\boldsymbol{b}^{(m-1)}=\boldsymbol{C}_{m-1,m}\boldsymbol{b}^{(m-2)}$.

Обратный ход метода вращений проводится точно так же, как и для метода Гаусса.

3 Решение нелинейных уравнений и систем нелинейных уравнений

Рассмотрим систему нелинейных уравнений с m неизвестными вида

$$\begin{cases} f_1(x_1,...,x_m)=0 \\ f_2(x_1,...,x_m)=0 \\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot \\ f_m(x_1,...,x_m)=0 \end{cases}. \qquad (3.1)$$

Задача решения такой системы является более сложной, чем нахождение корней одного нелинейного уравнения, и чем задача решения линейных алгебраических уравнений. В отличие от систем линейных уравнений здесь использование прямых методов исключено и решение находится с использованием итерационных методов, т.е. находится приближенное решение

$$\boldsymbol{x}^* = (x_1^*, \ldots, x_m^*),$$

удовлетворяющее при заданном $\varepsilon > 0$ условию $\left\|\boldsymbol{x}^* - \boldsymbol{x}\right\| < \varepsilon$.

Задача (3.1) совсем может не иметь решения или же число решений может быть произвольным. Введем векторную запись решения задачи:

$$\begin{gathered} \boldsymbol{x}=(x_1,\ldots,x_m)^T, \\ \boldsymbol{f}=(f_1,\ldots,f_m)^T, \\ \boldsymbol{f}(\boldsymbol{x})=0. \end{gathered} \qquad (3.2)$$

Будем считать, что функции f_i непрерывно дифференцируемы в некоторой окрестности точки $\boldsymbol{x}$. Введем матрицу Якоби

$$\boldsymbol{f}'(\boldsymbol{x})=\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \cdot & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_m} \end{pmatrix}.$$

Как и в случае решения одного уравнения начинаем с этапа локализации решения (отделения корней).

Пример. Дана система 2-х уравнений с двумя неизвестными

$$\begin{cases} x_1^3 + x_2^3 = 8x_1x_2 \\ x_1 \ln x_2 = x_2 \ln x_1 \end{cases}.$$

Найдем на плоскости место расположения решения.

Строим графики уравнений этой системы: а) – график 1-го уравнения, б) – график 2-го уравнения, в) – совмещенные графики.

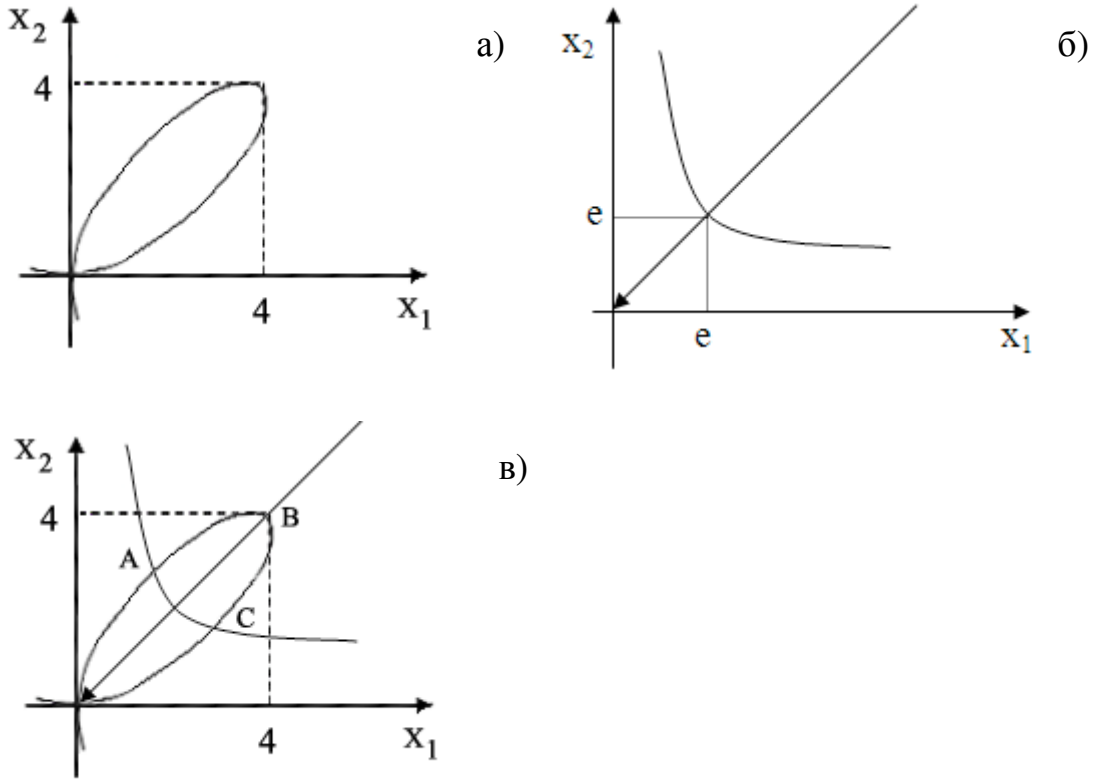


Рисунок 3 – Графики уравнений системы

Определяем границы координат пересечения графиков. Данная система имеет три решения. Координаты точек (B, C, A):

B: $x_1=4$, $x_2=4$

C: $3{,}5 < x_1 < 4$; $1{,}5 < x_2 < 2{,}5$.

Точки A и C симметричны относительно прямой $x_1=x_2$. Координаты точки C определим приближенно: $x_1 \approx 3{,}8$, $x_2 \approx 2$.

Обусловленность и корректность решения системы (3.1). Предположим что система (3.1) имеет решение $\boldsymbol{x}$ и в некоторой окрестности этого решения матрица Якоби не вырождена. Это означает, что в указанной окрестности нет других решений системы.

В одномерном случае нахождение корня нелинейного уравнения приводит к определению интервала неопределенности $(x^*-\delta,\ x^*+\delta)$. Так как значения функции $f(x)$ чаще всего вычисляются на ЭВМ с использованием приближенных методов нельзя ожидать, что в окрестности корня относительная погрешность окажется малой. Сама погрешность корня ведет себя крайне нерегулярно и в первом приближении может восприниматься как некоторая случайная величина. На рисунке 4а представлена идеальная ситуация, отвечающая исходной математической постановке задачи, а на рисунке 4б – реальная, соответствующая вычислениям значений функции f на ЭВМ.

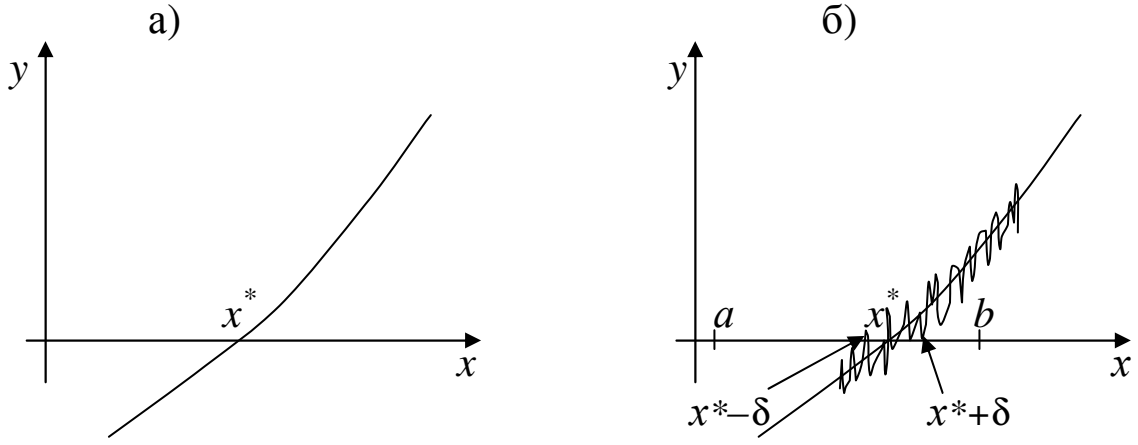


Рисунок 4 – Графическое изображение определения интервала неопределенности

В этом случае мы не можем определить, какая же точка в интервале неопределённости является решением. Радиус интервала неопределенности δ прямо пропорционален погрешности вычисления значения f. Кроме того, δ возрастает (обусловленность задачи ухудшается) с уменьшением $\left|f'(x^*)\right|$. Оценить величину δ довольно сложно, но выполнить это необходимо по следующим причинам:

– не имеет смысла ставить задачу о вычислении корня с точностью $\varepsilon < \delta$;
– после попадания очередного приближения в интервал неопределенности или близко от него, вычисления следует прекратить (этот момент для итерационных методов определяется крайне нерегулярным поведением приближений).

Если случай многомерный, то получаем некоторую область неопределённости D, и можем получить оценку радиуса ε этой области:

$$\bar{\varepsilon} \le \left\| (f'(x))^{-1} \right\| \cdot \Delta(f)$$

$$\left\| f(x) - f(x^*) \right\| \le \Delta(f).$$

Роль абсолютного числа обусловленности играет норма матрицы, обратной матрице Якоби $f'(x)$. Чем число обусловленности больше, тем хуже эта система обусловлена.

3.1 Метод простых итераций

Систему (3.1) преобразуем к следующему эквивалентному виду:

$$\begin{cases} x_1 = \varphi_1(x_1,...,x_m) \\ x_2 = \varphi_2(x_1,...,x_m) \\ \cdots\cdots\cdots \\ x_m = \varphi_m(x_1,...,x_m) \end{cases}. \qquad (3.3)$$

Или в векторной форме

$$x = \varphi(x) \qquad (3.4)$$

Пусть задано начальное приближение $x^{(0)} = (x_1^{(0)} ...,x_m^{(0)})^T$. Подставляем его в правую часть системы (3.4) и получаем $x^{(1)} = \varphi(x^{(0)})$, продолжая подстановку, находим $x^{(2)}$ и т.д. Получим последовательность точек $\{x^{(0)}, x^{(1)},...,x^{(\kappa+1)}\}$, которая приближается к искомому решению x.

3.1.1 Условия сходимости метода.

Пусть $\boldsymbol{\varphi}'(\boldsymbol{x})$ – матрица Якоби, соответствующая системе (3.4) и в некоторой Δ-окрестности решения $\boldsymbol{x}$ функции $\varphi_i(x)$ (i=1,2,…,m) дифференцируемы и выполнено неравенство вида:

$$\left\|\boldsymbol{\varphi}'(\boldsymbol{x})\right\| \le q\,,$$

где ($0 \le q < 1$), q – постоянная.

Тогда независимо от выбора $\boldsymbol{x}^{(0)}$ из Δ-окрестности корня итерационная последовательность $\{\boldsymbol{x}^k\}$ не выходит за пределы данной окрестности, метод сходится со скоростью геометрической прогрессии и справедлива оценка погрешности

$$\left\|\boldsymbol{x}^{(n)} - \boldsymbol{x}\right\| \le q^n \left\|\boldsymbol{x}^{(0)} - \boldsymbol{x}\right\|.$$

3.1.2 Оценка погрешности.

В данной окрестности решения системы, производные функции $\boldsymbol{\varphi}_i(\boldsymbol{x})$ (i=1,…,m) должны быть достаточно малы по абсолютной величине. Таким образом, если неравенство $\left\|\boldsymbol{\varphi}'(\boldsymbol{x})\right\| \le q$ не выполнено, то исходную систему (3.1) следует преобразовать к виду (3.3).

Пример. Рассмотрим предыдущий пример и приведем систему к удобному для итераций виду

$$x_1 = \sqrt[3]{8x_1x_2 - x_2^3},$$
$$x_2 = x_2 + \frac{x_2}{\ln x_2} - \frac{x_1}{\ln x_1}.$$

Проверяем условие сходимости вблизи точки С. Вычислим матрицу Якоби

$$\boldsymbol{\varphi}'(x_1,x_2)=\begin{pmatrix} \dfrac{8x_2}{3(8x_1x_2-x_2^3)^{2/3}} & \dfrac{8x_1-3x_2^2}{3(8x_1x_2-x_2^3)^{2/3}} \\ \dfrac{1}{\ln^2 x_1}-\dfrac{1}{\ln x_1} & 1+\dfrac{1}{\ln x_2}-\dfrac{1}{\ln^2 x_2} \end{pmatrix}.$$

Так как $x_1\approx3{,}8$, $x_2\approx2$, то при этих значениях вычисляем норму матрицы $\boldsymbol{\varphi}'(\boldsymbol{x})$

$$\|\boldsymbol{\varphi}'(\boldsymbol{x})\|\approx\|\boldsymbol{\varphi}'(3{,}8\ ;\ 2)\ \|\approx0{,}815.$$

Запишем итерационную процедуру

$$x_1^{(k+1)}=\sqrt[3]{8x_1^{(k)}x_2^{(k)}-(x_2^{(k)})^3},$$

$$x_2^{(k+1)}=x_2^{(k)}+\frac{x_2^{(k)}}{\ln x_2^{(k)}}-\frac{x_1^{(k)}}{\ln x_1^{(k)}}.$$

Следовательно, метод простых итераций будет сходиться со скоростью геометрической прогрессии, знаменатель которой $q\approx0{,}815$. Вычисления поместим в таблице 1.

Таблица 1 Решение системы нелинейных уравнений

k	0	1	…	8	9
$x_1^{(k)}$	3,80000	3,75155	….	3,77440	x_1=3,77418
$x_2^{(k)}$	2,00000	2,03895	…	2,07732	x_2=2,07712

При k=9 критерий окончания счета выполняется при $\varepsilon=10^{-3}$ и можно положить $x_1=3{,}774\pm0{,}001$ и $x_2=2{,}077\pm0{,}001$.

3.2 Метод Ньютона

Суть метода состоит в том, что система нелинейных уравнений сводится к решению систем линейных алгебраических уравнений. Пусть дана система (3.1) и задано начальное приближение $\boldsymbol{x}^{(0)}$. Приближение к решению $\boldsymbol{x}$ строим в виде последовательности $\boldsymbol{x}^{(0)}$, $\boldsymbol{x}^{(1)}$, …, $\boldsymbol{x}^{(n)}$.

В исходной системе (3.1) каждую функцию $f_i(x_1, x_2, ..., x_n)$, где $i=\overline{1,m}$, раскладывают в ряд Тейлора в точке $\boldsymbol{x}^{(n)}$ и заменяют линейной частью её разложения

$$f_i(\boldsymbol{x}) \approx f_i(\boldsymbol{x}^{(n)}) + \sum_{j=1}^{m} \frac{\partial f_i(\boldsymbol{x}^{(n)})}{\partial x_j} (x_j - x_j^{(n)}).$$

В результате получим систему линейных алгебраических уравнений

$$\begin{cases} f_1(\boldsymbol{x}^{(n)}) + \sum_{j=1}^{m} \frac{\partial f_1(\boldsymbol{x}^{(n)})}{\partial x_j} (x_j - x_j^{(n)}) = 0 \\ \cdots\cdots\cdots\cdots \\ f_m(\boldsymbol{x}^{(n)}) + \sum_{j=1}^{m} \frac{\partial f_m(\boldsymbol{x}^{(n)})}{\partial x_j} (x_j - x_j^{(n)}) = 0 \end{cases}. \qquad (3.5)$$

В матричной форме

$$\boldsymbol{f}(\boldsymbol{x}^{(n)}) + \boldsymbol{f}'(\boldsymbol{x}^{(n)}) * (\boldsymbol{x} - \boldsymbol{x}^{(n)}) = 0, \qquad (3.6)$$

где $\boldsymbol{f}'$ – матрица Якоби.

Предположим, что матрица невырожденная, то есть существует обратная матрица $\left[\boldsymbol{f}'(\boldsymbol{x}^{(n)})\right]^{-1}$.

Тогда система (3.6) имеет единственное решение, которое и принимается за очередное приближение $\boldsymbol{x}^{(n+1)}$. Отсюда выражаем решение $\boldsymbol{x}^{(n+1)}$ по итерационной формуле:

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} - \left[\boldsymbol{f}'(\boldsymbol{x}^{(n)})\right]^{-1} * \boldsymbol{f}(\boldsymbol{x}^{(n)}). \qquad (3.7)$$

Формула (3.7) и есть итерационная формула метода Ньютона для приближенного решения системы нелинейных уравнений.

Замечание. В таком виде формула (3.7) используется редко в виду того, что на каждой итерации нужно находить обратную матрицу. Поэтому поступают следующим образом: вместо системы (3.6) решают эквиваленту ей систему линейных алгебраических уравнений вида

$$\boldsymbol{f}'(\boldsymbol{x}^{(n)}) * \Delta\boldsymbol{x}^{(n+1)} = -\boldsymbol{f}(\boldsymbol{x}^{(n)}). \quad (3.8)$$

Это система линейных алгебраических уравнений относительно поправки $\Delta\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n+1)} - \boldsymbol{x}^{(n)}$. Затем полагают

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \Delta\boldsymbol{x}^{(n+1)}. \quad (3.9)$$

3.2.1 Сходимость метода

Теорема. *Пусть в некоторой окрестности решения* $\boldsymbol{x}$ *системы* (3.1) *функции* f_i (*при* $i = \overline{1,m}$) *дважды непрерывно дифференцируемы и матрица Якоби не вырождена. Тогда найдется такая малая* δ *окрестность вокруг решения* $\boldsymbol{x}$, *что при выборе начального приближения* $\boldsymbol{x}^0$ *из этой окрестности итерационный метод* (3.7) *не выйдет за пределы этой окрестности решения и справедлива оценка вида*

$$\left\| \boldsymbol{x}^{(n+1)} - \boldsymbol{x} \right\| \le \frac{1}{\delta} \left\| \boldsymbol{x}^{(n)} - \boldsymbol{x} \right\|^2,$$

где n *– номер итерации.*

Метод Ньютона сходится с квадратичной скоростью. На практике используется следующий критерий остановки:

$$\left\| \boldsymbol{x}^{(n)} - \boldsymbol{x}^{(n+1)} \right\| < \varepsilon.$$

4 Решение проблемы собственных значений

Пусть дана квадратная матрица $\boldsymbol{A}$ размерностью ($m*m$) и существует такое число λ, что выполняется равенство

$$\boldsymbol{A}\cdot\boldsymbol{x}=\lambda\cdot\boldsymbol{x},\ \boldsymbol{x}\neq 0,$$

тогда такое число λ называется собственным значением матрицы $\boldsymbol{A}$, а $\boldsymbol{x}$– соответствующим ему собственным вектором.

Перепишем это равенство в эквивалентной форме

$$(\boldsymbol{A} - \lambda\boldsymbol{E})\boldsymbol{x} = 0\ . \qquad (4.1)$$

Система (4.1) – однородная система линейных алгебраических уравнений. Для существования нетривиального решения системы (4.1) должно выполняться условие

$$det(\boldsymbol{A} - \lambda\boldsymbol{E}) = 0\ . \qquad (4.2)$$

Определитель в левой части уравнения является многочленом m-ой степени относительно λ, его называют – характеристическим определителем (характеристическим многочленом). Следовательно, уравнение (4.2) имеет m корней или m собственных значений. Среди них могут быть как действительные, так и комплексные корни.

Задача вычисления собственных значений сводится к нахождению корней характеристического многочлена (4.2). Корни могут быть найдены одним из итерационных методов (в частности методом Ньютона).

Если найдено некоторое собственное значение матрицы $\boldsymbol{A}$, то, подставив это число в систему (4.1) и решив эту систему однородных уравнений, находим собственный вектор $\boldsymbol{x}$, соответствующий данному собственному значению.

Собственные вектора будем при нахождении нормировать (вектор $\boldsymbol{x}$ умножаем на $||\boldsymbol{x}||^{-1}$, и таким образом они будут иметь единичную длину), нахождение собственных значений матрицы $\boldsymbol{A}$ и соответствующих им собственных

векторов и есть полное решение проблемы собственных значений. А нахождение отдельных собственных значений и соответствующих им векторов – называется решением частной проблемы собственных значений.

Эта проблема имеет самостоятельное значение на практике. Например, в электрических и механических системах собственные значения отвечают собственным частотам колебаний, а собственные вектора характеризуют соответствующие формы колебаний.

Эта задача легко решается для некоторых видов матриц: диагональных, треугольных и трехдиагональных матриц.

К примеру, определитель треугольной или диагональной матрицы равен произведению диагональных элементов, тогда и собственные числа равны диагональным элементам.

Пример. Матрица $\boldsymbol{A}$ – диагональная $\boldsymbol{A}=\begin{pmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{pmatrix}$.

Тогда $det(\boldsymbol{A}-\lambda\boldsymbol{E})=(a-\lambda)^3$, а характеристическое уравнение $(a-\lambda)^3=0$ имеет трехкратный корень $\lambda=a$.

Собственными векторами для матрицы $\boldsymbol{A}$ будут единичные векторы

$$\boldsymbol{e}_1=\begin{pmatrix}1\\0\\0\end{pmatrix},\quad \boldsymbol{e}_2=\begin{pmatrix}0\\1\\0\end{pmatrix},\quad \boldsymbol{e}_3=\begin{pmatrix}0\\0\\1\end{pmatrix}.$$

Пример. Найдем собственные числа матрицы

$$\boldsymbol{A}=\begin{pmatrix} 2 & -9 & 5 \\ 1{,}2 & -5{,}999 & 6 \\ 1 & -1 & -7{,}5 \end{pmatrix}.$$

Составим характеристический многочлен

$$P_3(\lambda)=det\,(\boldsymbol{A}-\lambda\boldsymbol{E})=det\begin{pmatrix} 2-\lambda & -9 & 5 \\ 1{,}2 & -5{,}3999-\lambda & 6 \\ 1 & -1 & -7{,}5-\lambda \end{pmatrix}=$$

$$=-\lambda^3-10{,}8999\,\lambda^2-26{,}49945\,\lambda-21{,}002\,.$$

Используя метод Ньютона, определим один из корней уравнения $P_3(\lambda)=0$, а именно $\lambda\approx-7{,}87279$.

Разделив многочлен $P_3(\lambda)$ на $(\lambda-\lambda_1)$ получим многочлен второй степени: $P_2(\lambda)=\lambda^2+3{,}02711\lambda+2{,}66765$. Решив квадратное уравнение, находим оставшиеся два корня:

$\lambda_{2,3}\approx-1{,}51356\pm0{,}613841*i$ (комплексные сопряженные корни).

Существуют прямые методы нахождения собственных значений и итерационные методы. Прямые методы неудобны для нахождения собственных значений для матриц высокого порядка. В таких случаях с учетом возможностей компьютера более удобны итерационные методы.

4.1 Прямые методы нахождения собственных значений

4.1.1 Метод Леверрье

Метод разделяется на две стадии:

- раскрытие характеристического уравнения,
- нахождение корней многочлена.

Пусть $det(\boldsymbol{A}-\lambda\boldsymbol{E})$ – есть характеристический многочлен матрицы $\boldsymbol{A}=\{a_{ij}\}$ $(i,j=1,2,\ldots,m)$,
т.е. $det(\boldsymbol{A}-\lambda\boldsymbol{E})=\lambda^m+p_1\lambda^{m-1}+\cdots+p_m$, и $\lambda_1,\lambda_2,\ldots,\lambda_m$ – есть полная совокупность корней этого многочлена (полный спектр собственных значений).

Рассмотрим суммы вида

$$S_k=\lambda_1^k+\lambda_2^k+\cdots+\lambda_m^k \quad (k=1,2,\ldots,m), \text{ т.е.}$$

$$\begin{aligned}
&S_1=\lambda_1+\lambda_2+\ldots+\lambda_m=SpA\\
&S_2=\lambda_1^2+\lambda_2^2+\cdots+\lambda_m^2=SpA^2\\
&\ldots\ldots\ldots\\
&S_m=\lambda_1^m+\lambda_2^m+\cdots+\lambda_m^m=SpA^m
\end{aligned}, \qquad (4.3)$$

где $SpA=\sum_{i=1}^{m}a_{ii}$ – след матрицы.

В этом случае при $k\le m$ справедливы формулы Ньютона для всех $(1\le k\le m)$

$$S_k + p_1 S_{k-1} + \cdots + p_{k-1} S_1 = -k p_k, \qquad (4.4)$$

Откуда получаем

при k=1 $p_1 = -S_1$,

при k=2 $p_2 = -1/2 \cdot (S_2 + p_1 \cdot S_1)$,

. . . . . (4.5)

при k=m $p_m = -1/n \cdot (S_m + p_1 \cdot S_{m-1} + p_2 \cdot S_{m-2} + \ldots + p_{m-1} \cdot S_1)$.

Следовательно, коэффициенты характеристического многочлена p_i можно определить, если известны суммы $S_1, S_2, \ldots, S_m$. Тогда схема алгоритма раскрытия характеристического определителя методом Леверрье будет следующей:

1) вычисляют степень матрицы: $\boldsymbol{A}^k = \boldsymbol{A}^{k-1} \cdot \boldsymbol{A}$ для $k=1,\ldots,m$;

2) определяют S_k – суммы элементов, стоящих на главной диагонали матриц $\boldsymbol{A}^k$;

3) по формулам (4.5) находят коэффициенты характеристического уравнения $p_i (i=1,2,\ldots,m)$.

4.1.2 Усовершенствованный метод Фадеева

Алгоритм метода:

1) вычисляют элементы матриц $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_m$:

$$\boldsymbol{A}_1 = \boldsymbol{A}; \quad SpA_1 = q_1; \quad \boldsymbol{B}_1 = \boldsymbol{A}_1 - q_1 \cdot \boldsymbol{E};$$

$$\boldsymbol{A}_2 = \boldsymbol{A} * \boldsymbol{B}_1; \quad \frac{SpA_2}{2} = q_2; \quad \boldsymbol{B}_2 = \boldsymbol{A}_2 - q_2 \cdot \boldsymbol{E};$$

. . . . . .

$$\boldsymbol{A}_m = \boldsymbol{A} * \boldsymbol{B}_{m-1}; \quad \frac{SpA_m}{m} = q_m; \quad \boldsymbol{B}_m = \boldsymbol{A}_m - q_m \cdot \boldsymbol{E},$$

(в конце подсчета $\boldsymbol{B}_m$ – нулевая матрица для контроля);

2) определяют коэффициенты характеристического уравнения p_i: $q_1 = -p_1$, $q_2 = -p_2, \ldots, q_m = -p_m$.

Существуют и другие методы раскрытия характеристического определителя: метод Крылова, Данилевского и др.

4.1.3 Метод Данилевского

Две матрицы $\boldsymbol{A}$ и $\boldsymbol{B}$ называются подобными, если одна получается из другой путем преобразования с помощью некоторой не вырожденной матрицы $\boldsymbol{S}$:

$$\boldsymbol{B}=\boldsymbol{S}^{-1}\cdot\boldsymbol{A}\boldsymbol{S},$$

если это равенство справедливо, то матрицы $\boldsymbol{A}$ и $\boldsymbol{B}$ подобны, а само преобразование называется преобразованием подобия (переход к новому базису в пространстве m - мерных векторов).

Пусть $\boldsymbol{y}$ – результат применения матрицы $\boldsymbol{A}$ к вектору $\boldsymbol{x}$

$$\boldsymbol{y}=\boldsymbol{A}\cdot\boldsymbol{x}.$$

Сделаем замену переменных:

$$\boldsymbol{x}=\boldsymbol{S}\cdot\boldsymbol{x}'\ ,\ \boldsymbol{y}=\boldsymbol{S}\cdot\boldsymbol{y}'.$$

Тогда равенство $\boldsymbol{y}=\boldsymbol{A}\cdot\boldsymbol{x}$ преобразуется к виду

$$\boldsymbol{y}'=\boldsymbol{S}^{-1}\cdot\boldsymbol{A}\cdot\boldsymbol{S}\cdot\boldsymbol{x}'.$$

В этом случае матрица $\boldsymbol{B}$ и матрица $\boldsymbol{A}$ имеют одни и те же собственные числа. Это можно легко увидеть раскрыв определитель

$$det(\boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}-\lambda\boldsymbol{E})=det(\boldsymbol{S}^{-1}(\boldsymbol{A}-\lambda\boldsymbol{E})\boldsymbol{S})=$$
$$=det(\boldsymbol{S}^{-1})\cdot det(\boldsymbol{A}-\lambda\boldsymbol{E})\cdot det(\boldsymbol{S})=det(\boldsymbol{A}-\lambda\boldsymbol{E}).$$

Следовательно, матрицы $\boldsymbol{A}$ и $\boldsymbol{B}$ – подобные, имеют одни и те же собственные значения. Но собственные векторы $\boldsymbol{x}$ и $\boldsymbol{x}'$ не совпадают, они связаны между собой простым соотношением

$$\boldsymbol{x}\ =\ \boldsymbol{S}\cdot\boldsymbol{x}'.$$

Такую матрицу $\boldsymbol{A}$ с помощью преобразования подобия или же последовательности таких преобразований можно привести к матрице Фробениуса вида:

$$\boldsymbol{F}=\begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1,m-1} & f_{1m} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}.$$

Детерминант матрицы $\boldsymbol{F}$ – $det(\boldsymbol{F})$ можно разложить по элементам первой строки:

$$det(\boldsymbol{F}-\lambda\boldsymbol{E})=(-1)^m(\lambda^m-p_1\lambda^{m-1}-\cdots-p_m).$$

Тогда коэффициенты характеристического многочлена матрицы $\boldsymbol{A}$ будут

$$p_1 = f_{11}\,,\; p_2 = f_{12},\ldots,\; p_m = f_{1m}\cdot.$$

Второй случай. Матрицу $\boldsymbol{A}$ преобразованием подобия можно привести к матрице $\boldsymbol{B}$ верхнего треугольного вида

$$\boldsymbol{B}=\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ 0 & b_{22} & \cdots & b_{2m} \\ . & . & . & . \\ 0 & 0 & \cdots & b_{mm} \end{pmatrix}.$$

Тогда собственными числами будут диагональные элементы матрицы $\boldsymbol{B}$:

$$det(\boldsymbol{B}-\lambda\boldsymbol{E})=(b_{11}-\lambda)(b_{22}-\lambda)\cdots(b_{mm}-\lambda).$$

Третий случай. Матрицу $\boldsymbol{A}$ с помощью преобразования подобия можно привести к Жордановой форме $\boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}=\boldsymbol{\Lambda}$

$$\boldsymbol{\Lambda}=\begin{pmatrix} \lambda_1 & S_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & S_2 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda_{\mathrm{m}} \end{pmatrix},$$

где λ_i – собственные числа матрицы $\boldsymbol{A}$; S_i – константы (0 или 1); если S_i=1, то $\lambda_i=\lambda_{i+1}$.

К четвёртому случаю относятся матрицы, которые с помощью преобразования подобия можно привести к диагональному виду (матрица простой структуры):

$$\boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}=\boldsymbol{D}=\begin{pmatrix}\lambda_1 & 0 & \cdots & 0\\ 0 & \lambda_2 & \cdots & 0\\ \cdots & \ldots & \cdots & \ldots\\ 0 & 0 & \cdots & \lambda_n\end{pmatrix},$$

у которой, как известно, собственными числами являются диагональные элементы.

4.1.4 Метод итераций определения первого собственного числа матрицы.

Пусть дано характеристическое уравнение:

$$det(\boldsymbol{A}-\lambda\cdot\boldsymbol{E}) = 0,$$

где $\lambda_1, \lambda_2,\ldots, \lambda_n$ – собственные значения матрицы $\boldsymbol{A}$.

Предположим, что $|\lambda_1|>|\lambda_2|\geq|\lambda_3|\geq\ldots\geq|\lambda_m|$, т.е. λ_1 – наибольшее по модулю собственное число.

Тогда для нахождения приближенного значения λ_1 используется следующая схема:

1) выбирают произвольно начальный вектор $\boldsymbol{y}^{(0)}$;

2) строят последовательность итераций вида:

$$\begin{aligned}
&\boldsymbol{y}^{(1)}=\boldsymbol{A}\boldsymbol{y}^{(0)},\\
&\boldsymbol{y}^{(2)}=\boldsymbol{A}\cdot\boldsymbol{A}\boldsymbol{y}^{(0)}=\boldsymbol{A}^2\boldsymbol{y}^{(0)},\\
&\ldots\ldots\ldots\\
&\boldsymbol{y}^{(m)}=\boldsymbol{A}\cdot\boldsymbol{A}^{m-1}\boldsymbol{y}^{(0)}=\boldsymbol{A}^m\boldsymbol{y}^{(0)},\\
&\boldsymbol{y}^{(m+1)}=\boldsymbol{A}\cdot\boldsymbol{A}^m\boldsymbol{y}^{(0)}=\boldsymbol{A}^{m+1}\boldsymbol{y}^{(0)}.
\end{aligned}$$

3) выбирают $\boldsymbol{y}^{(m)}=\boldsymbol{A}^m\boldsymbol{y}^{(0)}$ и $\boldsymbol{y}^{(m+1)}=\boldsymbol{A}^{m+1}\boldsymbol{y}^{(0)}$, тогда

$$\lambda_1=\lim_{n\to\infty}\frac{y_i^{(m+1)}}{y_i^{(m)}} \text{ или } \lambda_1\approx\frac{y_i^{(m+1)}}{y_i^{(m)}},$$

где $y_i^{(m)}$ и $y_i^{(m+1)}$ – соответствующие координаты векторов $\boldsymbol{y}^{(m)}$ и $\boldsymbol{y}^{(m+1)}$.

Возникает вопрос выбора начального вектора $\boldsymbol{y}^{(0)}$. При неудачном выборе можем не получить значения нужного корня, или же предела может не существовать. Этот факт при вычислении можно заметить по прыгающим значениям этого отношения, следовательно, нужно изменить $\boldsymbol{y}^{(0)}$. В качестве первого собственного вектора можно взять вектор $\boldsymbol{y}^{(n+1)}$ и пронормировать его.

Пример. Найти наибольшее по модулю собственное значение и соответствующий ему собственный вектор матрицы $\boldsymbol{A}$

$$\boldsymbol{A}=\begin{pmatrix}3 & 1 & 0\\ 1 & 2 & 2\\ 0 & 1 & 1\end{pmatrix}.$$

1) Выбираем начальный вектор $\boldsymbol{y}^{(0)}=\begin{pmatrix}1\\1\\1\end{pmatrix}$.

2) Вычисляем последовательно векторы $\boldsymbol{y}^{(1)}$, $\boldsymbol{y}^{(2)}$, ..., $\boldsymbol{y}^{(10)}$. Вычисления помещаем в таблицу 2.

Таблица 2 – Вычисление векторов $\boldsymbol{y}^{(n+1)}$

$\boldsymbol{y}^{(0)}$	$\boldsymbol{A}\cdot\boldsymbol{y}^{(0)}$	$A^2\cdot\boldsymbol{y}^{(0)}$	$\boldsymbol{A}^3\cdot\boldsymbol{y}^{(0)}$	.....	$\boldsymbol{A}^9\cdot\boldsymbol{y}^{(0)}$	$\boldsymbol{A}^{10}\cdot\boldsymbol{y}^{(0)}$
1	4	17	69		243569	941370
1	5	18	67		210663	812585
1	2	7	25		73845	284508

3) Вычисляем отношения координат векторов $y_i^{(10)}$ и $y_i^{(9)}$

$$\lambda_1^{(1)}=\frac{y_1^{(10)}}{y_1^{(9)}}=3{,}865;\ \lambda_1^{(2)}=\frac{y_2^{(10)}}{y_2^{(9)}}=3{,}857;\ \lambda_1^{(3)}=\frac{y_3^{(10)}}{y_3^{(9)}}=3{,}853.$$

4)Вычисляем λ_1 как среднее арифметическое $\lambda_1^{(1)}$, $\lambda_1^{(2)}$, $\lambda_1^{(3)}$.

$$\lambda_1=\frac{\lambda_1^{(1)}+\lambda_1^{(2)}+\lambda_1^{(3)}}{3}=3{,}858.$$

5) Определим соответствующий числу λ_1собственный вектор:

$$\boldsymbol{y}^{(10)}=\boldsymbol{A}^{(10)}\cdot\boldsymbol{y}^{(0)}=\begin{pmatrix}941370\\812585\\284508\end{pmatrix}.$$

Нормируем вектор $\boldsymbol{y}^{(10)}$, разделив на его длину

$$\left\|\boldsymbol{y}^{(10)}\right\|_3=\sqrt{941370^2+812585^2+284508^2}=1{,}28\cdot10^6$$

получим вектор

$$\boldsymbol{x}_1 = \begin{pmatrix} 0{,}74 \\ 0{,}64 \\ 0{,}22 \end{pmatrix}.$$

Далее можем определить второе собственное число

$$\lambda_2 \approx \frac{y_i^{(n+1)} - \lambda_1 y_i^{(n)}}{y_i^{(n+1)} - \lambda_1 y_i^{(n-1)}},$$

где $i=1,2,\ldots,n$.

При вычислении собственных чисел подобным образом, будет накапливаться ошибка. Данная методика позволяет приближенно оценить собственные значения матрицы.

5 Задача приближения функции

Постановка задачи.

Пусть на отрезке $[a,b]$ функция $\boldsymbol{y}=f(\boldsymbol{x})$ задана таблицей своих значений $y_0 = f(x_0),..., y_n = f(x_n)$.

Допустим, что вид функции $f(x)$ неизвестен. На практике часто встречается задача вычисления значений функции $y=f(x)$ в точках x, отличных от $x_0,...,x_n$. Кроме того, в некоторых случаях, не смотря на то, что аналитическое выражение $y=f(x)$ известно, оно может быть слишком громоздким и неудобным для математических преобразований (например, специальные функции). Кроме этого значения y_i могут содержать ошибки эксперимента.

Определение . Точки $x_0, ..., x_n$ называются узлами интерполяции.

Требуется найти аналитическое выражение функции $F(x)$, совпадающей в узлах интерполяции со значениями данной функции, т.е.

$$F(x_0) = y_0, F(x_1) = y_1, ..., F(x_n) = y_n.$$

Определение. Процесс вычисления значений функции $F(x)$ в точках отличных от узлов интерполирования называется интерполированием функции $f(x)$. Если $x \in [x_0, x_n]$, то задача вычисления приближенного значения функции в т. x называется интерполированием, иначе – экстраполированием.

Геометрически задача интерполирования функции одной переменной означает построение кривой, проходящей через заданные точки $(x_0, y_0),(x_1, y_1),...,(x_n, y_n)$ (рисунок 5). То есть задача в такой постановке может иметь бесконечное число решений.

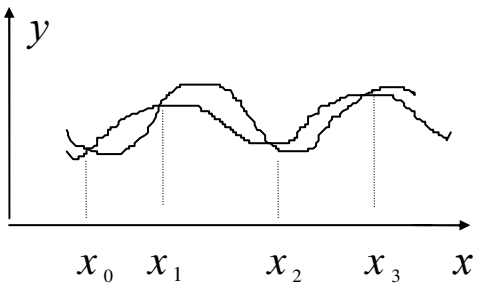


Рисунок 5 – Геометрическая иллюстрация задачи интерполирования функции

Задача становится однозначной, если в качестве $F(x)$ выбрать многочлен степени не выше n, такой что:

$$F_n(x_0)=y_0,\ F_n(x_1)=y_1,\ \ldots,\ F_n(x_n)=y_n.$$

Определение. Многочлен F_n(x), отвечающий вышеназванным условиям, называется интерполяционным многочленом.

Знание свойств функции f позволяет осознанно выбирать класс G аппроксимирующих функций. Широко используется класс функций вида

$$\Phi_m(x) = c_0\varphi_0(x) + c_1\varphi_1(x) + \ldots + c_m\varphi_m(x), \qquad (5.1)$$

являющихся линейными комбинациями некоторых базисных функций $\varphi_0(x)$, …, $\varphi_m(x)$.

Будем искать приближающую функцию в виде многочлена степени m, с коэффициентами c_0, …, c_m, которые находятся в зависимости от вида приближения. Функцию $\Phi_m(x)$ называют обобщенным многочленом по системе функций $\varphi_0(x)$, $\varphi_1(x)$, …,$\varphi_m(x)$, а число m – его степенью. Назовем обобщенный многочлен $\Phi_m(x)$ интерполяционным, если он удовлетворяет условию

$$\Phi_m(x_i)=y_i,\ (i=0,1,\ldots,n). \qquad (5.2)$$

Покажем, что условие (5.2) позволяет найти приближающую функцию единственным образом

$$\begin{cases} c_0\varphi_0(x_0)+c_1\varphi_1(x_0)+...+c_m\varphi_m(x_0)=y_0 \\ c_0\varphi_0(x_1)+c_1\varphi_1(x_1)+...+c_m\varphi_m(x_1)=y_1 \\ \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ c_0\varphi_0(x_n)+c_1\varphi_1(x_n)+...+c_m\varphi_m(x_n)=y_{n,} \end{cases} \qquad (5.3)$$

Система (5.3) есть система линейных алгебраических уравнений относительно коэффициентов $c_0, c_1, \ldots, c_m$.

Эта система n линейных уравнений имеет единственное решение, если выполняется условие $m=n$ и определитель квадратной матрицы $\boldsymbol{P}$

$$detP = \begin{vmatrix} \varphi_0(x_0), \varphi_1(x_0), ..., & \varphi_n(x_0) \\ \varphi_0(x_1), \varphi_1(x_1), ..., & \varphi_n(x_1) \\ \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . & \\ \varphi_0(x_n), \varphi_1(x_n), ..., & \varphi_n(x_n) \end{vmatrix} \neq 0.$$

Определение. Система функций $\varphi_0(x), \ldots, \varphi_n(x)$ называется Чебышевской системой функций на $[a,b]$,если определитель матрицы отличен от нуля $detP \neq 0$ при любом расположении узлов $x_i \in [a,b]$, $i=0,1,\ldots n$, когда среди этих узлов нет совпадающих.

Если мы имеем такую систему функций, то можно утверждать, что существует единственный для данной системы функций интерполяционный многочлен $\Phi_m(x)$, коэффициенты которого определяются единственным образом из системы (5.3).

Пример. При $m \leq n$ система функций 1, x, $x^2, \ldots, x^m$ линейно независима в точках x_0, x_1, ..., x_n, если они попарно различны.

5.1 Интерполяционный многочлен Лагранжа

Рассмотрим случай, когда узлы интерполирования не равноотстоят друг от друга на отрезке $[a,b]$.

Тогда шаг $h=x_{i+1}-x_i \neq \text{const}$. Задача имеет единственное решение, если в качестве интерполирующей функции $F(x)$ взять алгебраический многочлен

$$L_n(x)=a_0+a_1\,x+a_2\,x^2+\ldots+a_nx^n,$$

где a_i неизвестные постоянные коэффициенты.

Используя условие (5.2) можем записать

$$L_n(x_0)=y_0,\ L_n(x_1)=y_1,\ldots,\ L_n(x_n)=y_n. \tag{5.4}$$

Запишем это в виде:

$$\begin{cases} a_0+a_1x_0+a_{12}x_0^2+\ldots+a_{1n}x_0^n=y_0 \\ a_0+a_1x_1+a_{12}x_1^2+\ldots+a_{1n}x_1^n=y_1 \\ \ldots\ldots\ldots\ldots\ldots \\ a_0+a_1x_n+a_{12}x_n^2+\ldots+a_{1n}x_n^n=y_n. \end{cases} \tag{5.5}$$

Эта система однозначно разрешима, так как система функций $1,x,x^2,\ldots,x^n$ линейно независима в точках $x_0,x_1,\ldots,x_n$. Однозначная разрешимость следует из того факта, что определитель этой системы (определитель Вандермонда)

$$\begin{vmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{vmatrix} = \prod_{0\le j<i\le n}(x_i-x_j)\neq 0.$$

Без вывода приведем одну из форм записи интерполяционного многочлена Лагранжа

$$L_n(x)=y_0\cdot\frac{(x-x_1)\ldots(x-x_n)}{(x_0-x_1)\ldots(x_0-x_n)}+ y_1\cdot\frac{(x-x_0)(x-x_2)\ldots(x-x_n)}{(x_1-x_0)(x_1-x_2)\ldots(x_1-x_n)}+\ldots+y_n\cdot\frac{(x-x_0)\ldots(x-x_{n-1})}{(x_n-x_0)\ldots(x_n-x_{n-1})}. \tag{5.6}$$

Определение. Этот многочлен называется интерполяционным многочленом Лагранжа и сокращенно записывается в виде

$$L_n(x)=\sum_{i=0}^{n} y_i \frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)}. \qquad (5.7)$$

На практике часто пользуются линейной и квадратичной интерполяцией. В этом случае формула Лагранжа имеет вид

$L_1(x)= y_0\frac{(x-x_0)}{(x_0-x_1)}+y_1\frac{(x-x_0)}{(x_1-x_0)}$ – при линейной интерполяции;

$L_2(x)= y_0\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}+y_1\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}+y_2\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$ – при квадратичной интерполяции.

Рассмотрим теперь случай с равноотстоящими узлами. Тогда интерполяционная формула Лагранжа заметно упрощается. В этом случае шаг $h=x_{i+1}-x_i=\text{const}$. Введем в рассмотрение многочлен вида

$$Q_i(x)=\frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})\ldots(x_i-x_n)}.$$

Введем обозначение $q=\dfrac{x-x_0}{h}$, отсюда следует, что

$$\begin{aligned}
x-x_0&= q\cdot h,\\
x-x_1&= q\cdot h-h=h\cdot(q-1),\\
&\ldots\ldots\ldots\ldots\ldots\\
x-x_i&= q\cdot h-i\cdot h=h\cdot(q-i),\\
x-x_n&= q\cdot h-n\cdot h=h\cdot(q-n).
\end{aligned}$$

Тогда многочлен Q_i примет вид

$$Q_i(x)=\frac{q(q-1)\cdot[q-(i-1)]\cdot[q-(i+1)]\ldots(q-n)\cdot h^n}{i\cdot h(i-1)\cdot h\ldots h(-h)\ldots[-(n-i)\cdot h]}.$$

Произведя простейшие преобразования, получим выражение вида:

$$Q_i(q)=\frac{q\cdot(q-1)...(q-n)\cdot(-1)^{n-i}}{(q-i)\cdot i!(n-i)!}=(-1)^{n-i}\cdot\frac{C_n^i}{q-i}\cdot\frac{q(q-1)...(q-n)}{n!},$$

где C_n^i – число сочетаний из n элементов по i $C_n^i=\frac{n!}{i!(n-i)!}$.

Тогда интерполяционный многочлен Лагранжа для равноотстоящих узлов имеет вид:

$$L_n(x)=\frac{q(q-1)...(q-n)}{n!}\cdot\sum_{i=0}^{n}(-1)^{n-i}\cdot\frac{C_n^i}{q-i}\cdot y_i.$$

5.1.1 Оценка погрешности интерполяционного многочлена

Оценить погрешность интерполяционной формулы Лагранжа можно только тогда, когда известно аналитическое выражение интерполируемой функции, а точнее, если известно максимальное значение (n+1)-ой производной функции $f(x)$ на отрезке $[a,b]$. Пусть

$$|R_n(x)| = |f(x) - L_n(x)|,$$

где $R_n(x)$ –погрешность;

$f(x)$ – точное значение функции в точке x;

$L_n(x)$ – приближенное значение, полученное по полиному Лагранжа.

Если обозначить через $M_{n+1}=f^{(n+1)}(\xi)=\max\limits_{x\in[a,b]}\left|f^{(n+1)}(x)\right|$, где $\xi\in[a,b]$, причем $x_0=a$, $x_n=b$, то

$$\left|R_n(x)\right| \le \frac{M_{n+1}}{(n+1)!}\left|(\xi-x_0)(\xi-x_1)....(\xi-x_n)\right|.$$

5.2 Интерполяционные полиномы Ньютона

5.2.1 Интерполяционный многочлен Ньютона для равноотстоящих узлов

Вычисление значений функции для значений аргумента, лежащих в начале таблицы удобно проводить, пользуясь первой интерполяционной формулой Ньютона. Для этого введем понятие конечной разности.

Определение. Конечной разностью перового порядка называется разность между значениями функции в соседних узлах интерполяции. Тогда конечные разности в точках $x_0, x_1, \ldots, x_{n-1}$

$$\Delta y_0 = y_1 - y_0 = f(x_1) - f(x_0) = \Delta f(x_0),$$

$$\Delta y_1 = y_2 - y_1 = f(x_2) - f(x_1) = \Delta f(x_1),$$

. . . . . .

$$\Delta y_{n-1} = y_n - y_{n-1} = f(x_n) - f(x_{n-1}) = \Delta f(x_{n-1}).$$

Конечная разность второго порядка имеет вид:

$$\Delta^2 y_i = \Delta y_{i+1} - \Delta y_i,$$

. . . . . .

$$\Delta^n y_i = \Delta(\Delta^{n-1} y_i).$$

Рассмотрим некоторые свойства конечных разностей. Вторая конечная разность в точке x_i

$$\Delta^2 y_i = [f(x_{i+1}+\Delta x) - f(x_i+\Delta x)] - [f(x_{i+1}) - f(x_i)] = \\ = f(x_{i+2}) - 2\cdot f(x_{i+1}) + f(x_i) = y_{i+2} - 2\cdot y_{i+1} + y_i.$$

Аналогично третья конечная разность

$$\Delta^3 y_i = y_{i+3} - 3\cdot y_{i+2} + 3\cdot y_{i+1} - y_i.$$

Общее выражение для конечной разности n-го порядка имеет вид

$$\Delta^n y_i = y_{n+i} - C_n^1 y_{n+i-1} + C_n^2 y_{n+i-2} - \ldots \\ \ldots + (-1)^m C_n^m y_{n+i-m} + \ldots (-1)^n y_i,$$

а вообще, конечная разность порядка m от конечной разности порядка n

$$\Delta^m(\Delta^n y) = \Delta^{m+n} y.$$

Конечные разности n-го порядка от многочлена степени n – есть величины постоянные, а конечные разности $n+1$-го порядка равны нулю.

Для вычисления значений функции в начале таблицы требуется построить интерполяционный многочлен степени n такой, что выполнены условия интерполяции

$$P_n(x_0)=y_0,...,P_n(x_n)=y_n.$$

В силу единственности многочлена степени n, построенного по $n+1$ значениям функции $f(x)$ многочлен $P_n(x)$, в конечном счете, совпадает с многочленом Лагранжа. Найдем этот многочлен в виде:

$$P_n(x)=a_0+a_1\cdot(x-x_0)+a_2(x-x_0)(x-x_1)+...+a_n(x-x_0)...(x-x_{n-1}),$$

где $a_i(i=0,1,\ldots,n)$ – неизвестные коэффициенты. Для нахождения a_0 положим $x=x_0$. Тогда $P(x_0)=a_0$, отсюда $a_0=y_0$.

Для вычисления a_1 рассмотрим первую конечную разность для многочлена $P_n(x)$ в точке x.

$$\Delta P_n(x)=P_n(x+h)-P_n(x)=[a_0+a_1(x-x_0+h)+...+a_n(x-x_0+h)\cdot...$$
$$\cdot...(x-x_{n-1}+h)]-[a_0+a_1(x-x_0)+...+a_n(x-x_0)...(x-x_{n-1})].$$

В результате преобразований получим

$$\Delta P_n(x)=h\cdot a_1+2ha_2(x-x_0)+...+n\cdot ha_n(x-x_0)...(x-x_{n-1}).$$

Вычислим первую конечную разность многочлена $P_n(x)$ в точке x_0

$\Delta P_n(x_0)=a_1\cdot h$, но $\Delta P_n(x_0)=f(x_1)-f(x_0)=y_1-y_0=\Delta y_0$,

откуда $a_1=\dfrac{\Delta y_0}{h}$.

Чтобы определить коэффициент a_2, составим конечную разность второго порядка $\Delta^2 P_n(x)=\Delta P_n(x+h)-\Delta P_n(x)$. Отсюда после преобразования получим $a_2=\dfrac{\Delta^2 y_0}{2!h^2}$. Вычисляя конечные разности более высоких порядков и полагая $x=x_0$, придем к общей формуле для определения коэффициентов: $a_i=\dfrac{\Delta^i y_0}{i!h^i}$ $(i=0,1,2,\ldots,n)$.

Подставим значения a_i в многочлен, в результате получим первую интерполяционную формулу Ньютона:

$$P_n(x)=y_0+\frac{\Delta y_0}{1!h}\cdot(x-x_0)+...+\frac{\Delta^n y_0}{n!h^n}\cdot(x-x_0)...(x-x_{n-1}).$$

Первую интерполяционную формулу можно записать в том виде, в котором ее удобнее использовать для интерполирования в начале таблицы. Для этого введем переменную $q=(x-x_0)/h$, где h– шаг интерполирования. Тогда первая формула примет вид

$$P_n(x)=y_0+q\cdot\Delta y_0+\frac{q(q-1)}{2!}\cdot\Delta^2 y_0+...+\frac{q(q-1)...(q-n+1)}{n!}\cdot\Delta^n y_0.$$

5.2.2 Вторая интерполяционная формула Ньютона

Эта формула используется для интерполирования в конце таблицы. Построим интерполяционный многочлен вида

$$P_n(x)=a_0+a_1(x-x_n)+a_2(x-x_n)(x-x_{n-1})+...+a_n(x-x_n)...(x-x_1).$$

Неизвестные коэффициенты $a_0,a_1,...,a_n$ подберем так, чтобы были выполнены равенства

$$P_n(x_0)=y_0,\ P_n(x_1)=y_1,...,\ P_n(x_n)=y_n.$$

Для этого необходимо и достаточно, чтобы

$$\Delta^i P_n(x_{n-i})=\Delta^i y_{n-i}\ (i=0,1,\ldots,n).$$

В случае, если положить $x=x_n$, то сразу определяется коэффициент a_0

$$P_n(x) = y_n = a_0.$$

Из выражения для первой конечной разности найдем a_1:

$$\Delta P_n(x) = 1\cdot ha_1+2\cdot ha_2(x-x_{n-1})+...+n\cdot ha_n(x-x_{n-1})(x-x_{n-2})...(x-x_1).$$

Отсюда, полагая $x=x_{n-1}$, получим $a_1=\frac{\Delta y_{n-1}}{h}$. Из выражения

для второй конечной разности найдем a_2: $a_2 = \frac{\Delta^2 y_{n-2}}{2!h^2}$. Общая формула для коэффициента a_i имеет вид $a_i = \frac{\Delta^i y_{n-i}}{i!h^i}$.

Подставим эти коэффициенты в формулу многочлена и получим вторую интерполяционную формулу Ньютона:

$$P_n(x)=y_n+\frac{\Delta y_{n-1}}{h}(x-x_n)+...+\frac{\Delta^n y_0}{n!h^n}(x-x_n)...(x-x_1).$$

На практике используют формулу Ньютона в другом виде. Положим $q=(x-x_n)/h$. Тогда

$$P_n(x)=y_n+q\Delta y_{n-1}+\frac{q(q+1)}{2!}\Delta^2 y_{n-2}+...+\frac{q(q+1)...(q+n-1)}{n!}\Delta^n y_0.$$

5.3 Интерполирование сплайнами

Многочлен Лагранжа или Ньютона на всем отрезке $[a,b]$ с использованием большого числа узлов интерполирования часто приводит к плохому приближению, что объясняется накоплением погрешностей в ходе вычислений. Кроме того, из-за расходимости процесса интерполирования увеличение числа узлов не обязательно приводит к повышению точности вычислений.

Поэтому построим такой вид приближения, который:

- позволяет получить функцию, совпадающую с табличной функцией в узлах;
- приближающая функция в узлах таблицы имеет непрерывную производную до нужного порядка;

В силу вышесказанного на практике весь отрезок $[a,b]$ разбивается на частичные интервалы и на каждом из них приближающая функция $f(x)$ заменяется многочленом невысокой степени. Такая интерполяция называется кусочно-полиномиальной интерполяцией.

Определение. Сплайн – функцией называют кусочно-полиномиальную функцию, определенную на отрезке $[a,b]$ и имеющую на этом отрезке некоторое число непрерывных производных.

Слово сплайн означает гибкую линейку, которую используют для проведения гладких кривых через определенное число точек на плоскости. Преимущество сплайнов – сходимость и устойчивость процесса вычисления. Рассмотрим частный случай (часто используемый на практике), когда сплайн определяется многочленом третьей степени.

5.3.1 Построение кубического сплайна

Пусть на отрезке $[a,b]$ в узлах сетки заданы значения некоторой функции $f(x)$, т.е. $a = x_0 < x_1 < x_2 ... < x_n = b$, $y_i = f(x_i)$ $(i = 0,1,\ldots, n)$.

Сплайном, соответствующим этим узлам функции $f(x)$ называется функция $S(x)$, которая:

1) на каждом частичном отрезке является многочленом третьей степени;

2) функция $S(x)$ и ее две первые производные $S'(x), S''(x)$ непрерывны на $[a,b]$;

3) $S(x_i) = f(x_i)$.

На каждом частичном отрезке $[x_{i-1}, x_i]$ будем искать сплайн $S(x) = S_i(x)$, где $S_i(x)$ многочлен третьей степени

$$S_i(x) = a_i + b_i(x - x_i) + \frac{c_i}{2} \cdot (x - x_i)^2 + \frac{d_i}{6} \cdot (x - x_i)^3. \qquad (5.8)$$

То есть для $x \in [x_{i-1}, x_i]$ нужно построить такую функцию $S_i(x)$, где a_i, b_i, c_i, d_i подлежат определению. Для всего отрезка интерполирования $[a,b]$, таким образом, необходимо определить $4 \cdot n$ неизвестных коэффициента.

$$S'(x) = b_i + c_i(x - x_i) + \frac{d_i}{2} \cdot (x - x_i)^2,$$
$$S''(x) = c_i + d_i(x - x_i),$$
$$S_i(x) = a_i = y_i.$$

Доопределим $a_0 = f(x_0) = y_0$. Требование непрерывности функции $S(x)$ приводит к условия $S_i(x_i) = S_{i+1}(x_i)$, $(i = 0,1,\ldots,n-1)$.

Отсюда из (5.8) получаем следующие уравнения:

$$a_i = a_{i+1} + b_{i+1}(x_i - x_{i+1}) + \frac{c_{i+1}}{2}(x_i - x_{i+1})^2 + \frac{d_{i+1}}{6}(x_i - x_{i+1})^3 \ (\text{i} = 1,2,\ldots,n-1).$$

Введем шаг интерполирования $h_i = x_i - x_{i-1}$.

Тогда последнее равенство можно переписать в виде

$$h_i \cdot b_i - \frac{h_i^2}{2} \cdot c_i + \frac{h_i^3}{6} \cdot d_i = f_i - f_{i-1} (i = 1,2,\ldots,n).$$

Из непрерывности первой производной следует

$$h_i \cdot c_i - \frac{h_i^2}{2} \cdot d_i = b_i - b_{i-1} \ (i=2,3,\ldots,n),$$

а из непрерывности второй производной

$$h_i d_i = c_i - c_{i-1} \ (i=2,3,\ldots,n).$$

Объединив все три вида уравнений, получим систему из $3n$-2 уравнений относительно $3n$ неизвестных b_i, c_i, d_i. Два недостающих уравнения получим, задав граничные условия для функции $S(x)$. Для этого воспользуемся граничными условиями для сплайн-функции в виде $S''(a) = S''(b) = 0$ (концы гибкой линейки свободны).

Тогда получим систему уравнений

$$\begin{cases} h_i \cdot d_i = c_i - c_{i-1}, c_0 = c_n = 0, (i = 1,2,..., n) \\ h_i \cdot c_i - \frac{h_i^2}{2} \cdot d_i = b_i - b_{i-1}, (i = 2,3,..., n) \\ h_i \cdot b_i - \frac{h_i^2}{2} \cdot c_i + \frac{h_i^3}{6} \cdot d_i = f_i - f_{i-1}, (i = 1,2,..., n). \end{cases} \qquad (5.9)$$

Решая систему методом подстановки (исключаем из (5.9) неизвестные b_i, d_i), получим систему:

$$\begin{cases} h_i c_{i-1} + 2(h_i + h_{i+1}) \cdot c_i + h_{i+1} c_{i+1} = 6 \cdot \left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i} \right), \ (i = 1,2,...,n) \\ c_0 = c_n = 0 \end{cases} \qquad (5.10)$$

Система (5.10) имеет трехдиагональную матрицу. Эта система может быть решена методом прогонки или Гаусса. Метод прогонки рассматривается в пункте 6.7.2.

После решения системы коэффициенты сплайна d_i, b_i определим через коэффициенты c_i с помощью явных формул

$$d_i = \frac{c_i - c_{i-1}}{h_i},$$

$$b_i = \frac{h_i}{2}\cdot c_i - \frac{h_i^2}{6}\cdot d_i + \frac{y_i - y_{i-1}}{h_i} \quad (i = 1,2,\ldots,n).$$

Существуют специальные виды записи сплайнов на каждом из промежутков $[x_i, x_{i+1}]$ [9], которые позволяют уменьшить число неизвестных коэффициентов сплайна. Вводятся обозначения

$S'(x_i) = m_i$, $i = 0,1,...,n$,

$h_i = x_{i+1} - x_i$ и $t = (x - x_i)/h_i$.

На отрезке $[x_i, x_{i+1}]$ кубический сплайн записывается в виде

$$S(x) = y_i(1-t)^2(1+2t) + y_{i+1}t^2(3-2t) + m_i h_i t(1-t)^2 - m_{i+1}t^2(1+t)h_i.$$

Кубический сплайн, записанный в таком виде, на каждом из промежутков $[x_i, x_{i+1}]$ непрерывен вместе со своей первой производной на $[a,b]$.

Выберем m_i таким образом, чтобы и вторая производная была непрерывна во всех внутренних узлах. Отсюда получим систему уравнений:

$$\lambda_i m_{i-1} + 2m_i + \mu_i m_{i+1} = 3\left(\mu_i \frac{y_{i+1} - y_i}{h_i} + \lambda_i \frac{y_i - y_{i-1}}{h_{i-1}}\right),$$

где $\mu_i = \frac{h_{i-1}}{h_{i-1} + h_i}$, $\lambda_i = 1 - \mu_i = \frac{h_i}{h_{i-1} + h_i}$, $i = 1,2,...,n-1$.

К этим уравнениям добавим уравнения, полученные из граничных условий

$$2m_0 + m_1 = 3\frac{y_1 - y_0}{h_0}, \quad m_{n-1} + 2m_n = 3\frac{y_n - y_{n-1}}{h_{n-1}}.$$

В результате получаем аналогичную систему с трехдиагональной матрицей. Решаем систему линейных уравнений относительно коэффициентов m_i методом пргонки.

5.3.2 Сходимость процесса интерполирования кубическими сплайнами

Доказывается, что при неограниченном увеличении

числа узлов на одном и том же отрезке $[a,b]$ $S(x) \to f(x)$. Оценка погрешности интерполяции $R(x) = f(x) - S(x)$ зависит от выбора сетки и степени гладкости функции $f(x)$.

При равномерной сетке

$$x_i = a + i \cdot h (i=0,1,\ldots,n)$$

$$\left| f(x) - S_h(x) \right| \le \frac{M_4 \cdot h^4}{8},$$

$$\text{где } M_4 = \max_{[a,b]} | f^{IV}(x) |.$$

Другие постановки задачи интерполирования функций.

1. Если функция периодическая, то используется тригонометрическая интерполяция с периодом l, которая строится с помощью тригонометрического многочлена

$$T_n(x) = a_0 + \sum_{k=1}^{n} (a_k \cos \frac{\pi kx}{l} + b_k \sin \frac{\pi kx}{l}),$$

коэффициенты которого находятся из системы уравнений

$$T_n(x_i) = f(x_i) \ (i= 1,2,\ldots, 2n+1).$$

2. Выделяют приближение функций рациональными, дробно – рациональными и другими функциями. В данном пособии эти вопросы не рассматриваются.

5.4 Аппроксимация функций методом наименьших квадратов

К такой задаче приходят при статистической обработке экспериментальных данных с помощью регрессионного анализа. Пусть в результате исследования некоторой величины x значениям $x_1, x_2, x_3, \ldots, x_n$ поставлены в соответствие значения $y_1, y_2, y_3, \ldots, y_n$ некоторой величины y.

Требуется подобрать вид аппроксимирующей зависимости $y=f(x)$, связывающей переменные x и y. Здесь могут иметь место следующие случаи. Во-первых: значения функции $f(x)$ могут быть заданы в достаточно большом количестве узлов; во-вторых: значения таблично заданной функции отягощены погрешностями. Тогда проводить приближения функции с помощью интерполяционного многочлена нецелесообразно, т.к.

- число узлов велико и пришлось бы строить несколько интерполяционных многочленов;

- построив интерполяционные многочлены, мы повторили бы те же самые ошибки, которые присущи таблице.

Будем искать приближающую функцию из следующих соображений:

1) приближающая функция не проходит через узлы таблицы и не повторяет ошибки табличной функции;

2) чтобы сумма квадратов отклонений приближающей функции от таблично заданной в узлах таблицы была минимальной.

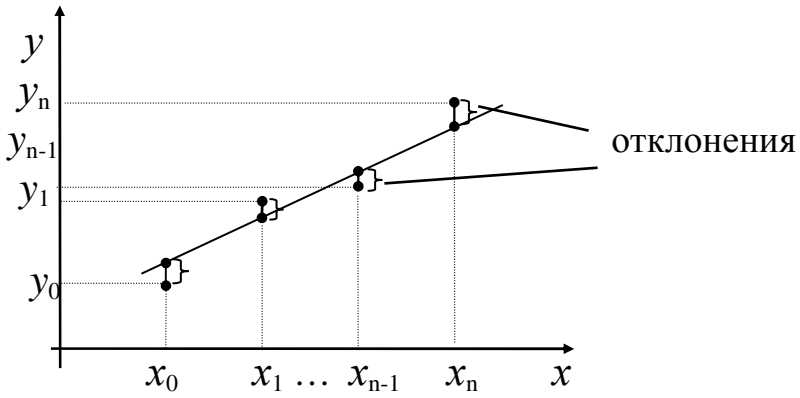


Рисунок 6 – Графическое изображение отклонений приближающей функции от таблично заданной

Рассмотрим линейную задачу наименьших квадратов.

Определение. Уровень погрешности, допускаемый при снятии характеристики измеряемой величины, называется шумом таблицы.

Пусть функция $y=f(x)$ задана таблицей приближенных значений $y_i \approx f(x_i)$, $i=0,1,\ldots,n$, полученных с ошибками $\varepsilon_i = y_i^0 - y_i$, где $y_i^0 = f(x_i)$.

Пусть даны функции $\varphi_0(x),\varphi_1(x),..\varphi_m(x)$, назовем их базисными функциями.

Будем искать приближающую (аппроксимирующую) функцию в виде линейной комбинации базисных функций

$$y = \Phi_m(x) \equiv c_0\varphi_0(x) + c_1\varphi_1(x) + \ldots + c_m\varphi_m(x). \qquad (5.11)$$

Такая аппроксимация называется линейной, а $\Phi_m(x)$ – обобщенным многочленом.

Будем определять коэффициенты обобщенного многочлена $c_0,\ldots,c_m$ используя критерий метода наименьших квадратов. Согласно этому критерию вычислим сумму квадратов отклонений таблично заданной функции от искомого многочлена в узлах:

$$\delta_m = \sum_{i=0}^{n} (y_i - \Phi_m(x_i))^2 = \sum_{i=0}^{n} (y_i - c_0\varphi_0(x_i) - \ldots - c_m\varphi_m(x_i))^2 . \qquad (5.12)$$

Выражение для δ_m можно рассматривать как функцию от неизвестных $c_0,\ldots, c_m$. Нас интересует, при каких значениях $c_0,\ldots, c_m$, значение δ_m будет минимально.

Для этого воспользуемся условием существования экстремума функции, а именно, найдем частные производные от δ_m по всем переменным $c_0,\ldots, c_m$ и приравняем их к нулю. Получим систему вида:

$$\begin{cases} \dfrac{\partial \delta_m}{\partial c_0} = -2\sum\limits_{i=0}^{n} (y_i - c_0\varphi_0(x_i) - \ldots - c_m\varphi_m(x_i)) \cdot \varphi_0(x_i) = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \dfrac{\partial \delta_m}{\partial c_m} = -2\sum\limits_{i=0}^{n} (y_i - c_0\varphi_0(x_i) - \ldots - c_m\varphi_m(x_i)) \cdot \varphi_m(x_i) = 0 \end{cases} \qquad (5.13)$$

Система (5.13) – система линейных уравнений относительно $c_0,\ldots, c_m$.

Чтобы систему (5.13) записать компактно, ведем определение.

Определение. Скалярным произведением функций f на g на множестве точек $x_0,\ldots,x_n$ называется выражение

$$(f, g) = \sum_{i=0}^{n} f(x_i) \cdot g(x_i) .$$

Тогда систему (5.13) можно записать в виде:

$$\begin{cases} c_0(\varphi_0,\varphi_0)+c_1(\varphi_0,\varphi_1)+\ldots+c_m(\varphi_0,\varphi_m)=(\varphi_0,y) \\ c_0(\varphi_1,\varphi_0)+c_1(\varphi_1,\varphi_1)+\ldots+c_m(\varphi_1,\varphi_m)=(\varphi_1,y) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ c_0(\varphi_m,\varphi_0)+c_1(\varphi_m,\varphi_1)+\ldots+c_m(\varphi_m,\varphi_m)=(\varphi_m,y) \end{cases}. \qquad (5.13а)$$

Системы (5.13) или (5.13а) будем называть нормальной системой уравнений.

Решив ее, мы найдем коэффициенты $c_0,\ldots,c_m$ и следовательно, найдем вид аппроксимирующего многочлена. Напомним, что это возможно, если базисные функции линейно независимы, а все узлы различны.

Осталось определить степень многочлена m. Прямому вычислению поддаются только значения среднеквадратичного отклонения δ_m, анализируя которые будем выбирать степень многочлена.

Алгоритм выбора степени многочлена m.

В случае, когда $m=n$ мы получим интерполяционный многочлен, поэтому выберем $m<<n$. Так же необходимо задать числа ε_1 и ε_2, учитывая следующее:

1) $\varepsilon_1 >0$ и $\varepsilon_2>0$ должны быть такими, чтобы δ_m находилось между ними;

2) первоначально m выбирают произвольно, но учитывая условие, что $m<<n$;

3) выбрав m, строят системы (5.13) и (5.13а), решив которые, находят $c_0,\ldots, c_m$;

4) используя найденные коэффициенты, вычисляется δ_m и проверяется, попало ли оно в промежуток между ε_1 и ε_2. Если попало, то степень многочлена выбрана правильно, иначе

а) если $\delta_m> \varepsilon_1$, то степень необходимо уменьшить хотя бы на единицу;

б) если $\delta_m <\varepsilon_2$, то степень необходимо увеличить хотя бы на единицу.

5) затем строим приближающую функцию .

Очень часто для приближения по методу наименьших квадратов используются алгебраические многочлены сте-

пени $m \le n$, т.е. $\varphi_k(x) = x^k$. Тогда нормальная система (5.13) принимает следующий вид:

$$\sum_{j=0}^{m}\left(\sum_{i=0}^{n} x_i^{j+k}\right) c_j = \sum_{i=0}^{n} y_i x_i^k \ , \ (k = 0,1,\ldots,m). \qquad (5.14)$$

Запишем систему (5.14) в развернутом виде в двух наиболее простых случаях $m = 1$ и $m = 2$.

В случае многочлена первой степени $P_1(x) = c_0 + c_1 x$, нормальная система имеет вид

$$\begin{cases} (n+1)c_0 + \left(\sum_{i=0}^{n} x_i\right) c_1 = \sum_{i=0}^{n} y_i \\ \left(\sum_{i=0}^{n} x_i\right) c_0 + \left(\sum_{i=0}^{n} x_i^2\right) c_1 = \sum_{i=0}^{n} y_i x_i. \end{cases} \qquad (5.15)$$

Для многочлена второй степени $P_2(x) = c_0 + c_1 x + c_2 x^2$, нормальная система имеет вид

$$\begin{cases} (n+1)c_0 + \left(\sum_{i=0}^{n} x_i\right) c_1 + \left(\sum_{i=0}^{n} x_i^2\right) c_2 = \sum_{i=0}^{n} y_i \\ \left(\sum_{i=0}^{n} x_i\right) c_0 + \left(\sum_{i=0}^{n} x_i^2\right) c_1 + \left(\sum_{i=0}^{n} x_i^3\right) c_2 = \sum_{i=0}^{n} y_i x_i \ . \\ \left(\sum_{i=0}^{n} x_i^2\right) c_0 + \left(\sum_{i=0}^{n} x_i^3\right) c_1 + \left(\sum_{i=0}^{n} x_i^4\right) c_2 = \sum_{i=0}^{n} y_i x_i^2 \end{cases} \qquad (5.16)$$

6 Численные методы решения задачи Коши для обыкновенных дифференциальных уравнений и систем дифференциальных уравнений

Будем рассматривать задачу Коши для системы обыкновенных дифференциальных уравнений(ОДУ).Запишем систему в векторной форме

$$\frac{d\boldsymbol{u}}{dt} = \boldsymbol{f}(t,\boldsymbol{u}), \qquad (6.1)$$

где: $\boldsymbol{u}$ – искомая вектор-функция; t– независимая переменная; $\boldsymbol{u}(t) = (u_1(t),...,u_m(t))$; $\boldsymbol{f}(t,\boldsymbol{u}) = (f_1,...,f_m)$, m– порядок системы; $u_1(t),...,u_m(t)$ – координаты; $t \geq 0$; $\boldsymbol{u}(0) = \boldsymbol{u}^0$.

Запишем систему (6.1) в развернутом виде

$$\frac{du_i}{dt} = f_i(t,u_1,...,u_m), \qquad (6.2)$$

где: $i=1,\ldots,m$; $u_i(0)=u_i^0$.

В случае $i=1$ – это будет ОДУ 1-го порядка, а при $i=2$ – система из двух уравнений первого порядка.

В случае $i=1$ решение задачи Коши предполагает нахождение интегральной кривой, проходящей через заданную точку и удовлетворяющую заданному начальному условию.

Задача состоит в том, чтобы найти искомую вектор-функцию $\boldsymbol{u}$, удовлетворяющую (6.1) и заданным начальным условиям.

Известны условия, гарантирующие существование и единственность решения (6.1) или (6.2).

Предположим, что функции $f_i(i=1, \ldots, m)$ непрерывны по всем аргументам в некоторой замкнутой области $D=\{t \leq a, u_i - u_i^0 \leq b\}$, где a,b– известные константы.

Из непрерывности функций следует их ограниченность, т.е. функции f_i сверху ограничены некоторой константой M: $|f_i| < M$ (где $M \geq 0$) всюду в области D и пусть в области

D функции f_i удовлетворяют условию Липшица по аргументам $u_1,...,u_m$. Это значит, что

$$|f_i(t,u_1',....,u_m') - f_i(t,u_1'',...,u_m'')| \le L(|u_1' - u_1''| + ... + |u_m' - u_m''|)$$

для любых двух точек $(t,u_1',....,u_m')$ и $(t,u_1'',...,u_m'')$ из области D. Тогда существует единственное решение задачи (6.1)

$$u_1 = u_1(t),...., u_m = u_m(t) \text{ ,определенное при } t \le T = min\{a, b/M\} \quad (6.3)$$

и принимающее при t=0 заданные начальные значения.

Существует два класса методов для решения задачи (6.1):

1) семейство одношаговых методов(Рунге-Кутта);

2) семейство многошаговых(m-шаговых) методов.

Сначала рассмотрим одношаговые методы. Для простоты возьмем одно уравнение

$$\frac{du}{dt} = f(t,u), \quad (6.4)$$

где: $u(0) = u_0$; t>0.

По оси t введем равномерную сетку с шагом τ>0 , т.е. рассмотрим систему точек $\omega_\tau = \{t_n = n \cdot \tau, n = 0,1,2,...\}$. Обозначим через $u(t)$ точное решение (6.4) , а через $y_n = y(t_n)$ приближенные значения функций u в заданной системе точек.

Приближенное решение является сеточной функцией, т.е. определено только в точках сетки ω_τ.

6.1 Семейство одношаговых методов решения задачи Коши

6.1.1 Метод Эйлера (частный случай метода Рунге-Кутта)

Уравнение (6.4) заменяется разностным уравнением

$$\frac{y_{n+1} - y_n}{\tau} = f(t_n, y_n),\ n=0,1,2,\ldots,.\ \ y_0 = u_0$$

В окончательной форме значения y_{n+1} можно определить по явной формуле

$$y_{n+1} = y_n + \tau \cdot f(t_n, y_n). \tag{6.5}$$

Вследствие систематического накопления ошибок метод используется редко или используется только для оценки вида интегральной кривой.

Определение 1. Метод сходится к точному решению в некоторой точке t, если $|y_n - u(t_n)| \to 0$, при $\tau \to 0$, $t_n = t$.

Метод сходится на интервале $(0,t]$, если он сходится в каждой точке этого интервала.

Определение 2. Метод имеет p-й порядок точности, если существует такое число $p>0$, для которого $|y_n - u(t_n)| = O(\tau^p)$, при $\tau \to 0$, где: τ – шаг интегрирования; O – малая величина порядка τ^p.

Так как $\frac{u_{n+1} - u_n}{\tau} = u'(t_n) + O(\tau)$, то метод Эйлера имеет первый порядок точности.

Порядок точности разностного метода совпадает с порядком аппроксимации исходного дифференциального уравнения.

6.1.2 Методы Рунге-Кутта

Метод Рунге-Кутта второго порядка точности

Отличительная особенность методов Рунге-Кутта от метода (6.5) заключается в том, что значение правой части уравнения вычисляется не только в точках сетки, но и также в середине отрезков(промежуточных точках).

Предположим, что приближенное значение y_n решения задачи в точке $t=t_n$ уже известно. Для нахождения y_{n+1} поступают следующим образом:

1) используют схему Эйлера в таком виде

$$\frac{y_{n+1/2} - y_n}{0{,}5\tau} = f(t_n, y_n) \tag{6.6}$$

и отсюда вычисляют промежуточное значение $y_{n+1/2}$;

2) воспользуемся разностным уравнением вида

$$\frac{y_{n+1}-y_n}{\tau} = f(t_n + 0{,}5\,\tau,\, y_{n+1/2}), \qquad (6.7)$$

откуда найдем значение y_{n+1}. Далее подставим значение $y_{n+1/2} = y_n + 0{,}5\,\tau\, f_n$ в уравнение (6.7). Тогда получим разностное уравнение

$$\frac{y_{n+1}-y_n}{\tau} = f(t_n + 0{,}5\cdot\tau,\, y_n + 0{,}5\cdot\tau\cdot f_n), \qquad (6.8)$$

где $f_n = f(t_n, y_n)$.

Можно показать, что метод (6.8) имеет второй порядок точности, т.е. $\left| y_n - u(t) \right| = O(\tau^2)$.

Реализация метода (6.8) в виде двух этапов (6.6),(6.7) называется методом предиктор–корректор (прогноза и коррекции) в том смысле, что на первом этапе (6.6) приближенное значение предсказывается с невысокой точностью $O(\tau)$, а на втором этапе (6.7) это предсказанное значение исправляется и результирующая погрешность имеет второй порядок $O(\tau^2)$.

Будем рассматривать явные методы. Задаем числовые коэффициенты a_i, b_{ij}, i=2,...,m; j=1,2,...,m-1 и σ_i, i=1,2,...,m. Последовательно вычисляем функции

$$k_1 = f(t_n, y_n);$$
$$k_2 = f(t_n + a_2\tau,\, y_n + b_{21}\tau\cdot k_1);$$
$$k_3 = f(t_n + a_3\tau,\, y_n + b_{31}\tau\cdot k_1 + b_{32}\tau\cdot k_2);$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$k_n = f(t_n + a_n\tau,\, y_n + b_{m1}\tau\cdot k_1 + \ldots + b_{mm-1}\tau\cdot k_{m-1}).$$

Затем из формулы $\frac{y_{n+1}-y_n}{\tau} = \sum_{i=1}^{m} \sigma_i k_i$ находим новое значение y_{n+1}=$y(t_{n+1})$. Здесь a_i, b_{ij}, σ_i–числовые параметры, которые определяются или выбираются из соображений точности вычислений. Чтобы это уравнение аппроксимировало уравнение (6.4), необходимо потребовать выполнения ус-

ловия $\sum_{i=1}^{m}\sigma_i = 1$. При m=1 получается метод Эйлера, при m=2 получаем семейство методов

$$y_{n+1} = y_n + \tau(\sigma_1 k_1 + \sigma_2 k_2), \qquad (6.9)$$

где: $k_1 = f(t_n, y_n)$; $k_2 = f(t_n + a_2\tau, y_n + b_{21}\tau k_1)$; $y_0 = u_0$.

Семейство определяет явные методы Рунге-Кутта. Подставив нужные σ_1 и σ_2, получаем окончательную формулу. Точность этих методов совпадает с точностью аппроксимирующего метода и равна $O(\tau^2)$.

Невязкой, или погрешностью аппроксимации метода (6.9) называется величина

$$\delta_n = -\frac{u_{n+1} - u_n}{\tau} + \sigma_1 f(t_n, u_n) + \sigma_2 f(t_n + a_2\tau, u_n + b_{21}\tau f(t_n, u_n)),$$

полученная заменой в (6.9) приближенного решения точным решением.

При $\sigma_1+\sigma_2$=1 получим первый порядок точности. Если же потребовать дополнительно $\sigma_2 b_{21} = \sigma_2 a_2 = 0{,}5$, то получим методы второго порядка точности вида

$\frac{y_{n+1} - y_n}{\tau} = (1-\sigma) f(t_n, y_n) + \sigma f(t_n + a\tau, y_n + a\tau f(t_n, y_n))$ при $\sigma \cdot a = 0{,}5$.

Приведем один из методов Рунге-Кутта третьего порядка точности

$$\frac{y_{n+1} - y_n}{\tau} = \frac{1}{6}(k_1 + 4k_2 + k_3),$$

где: $k_1 = f(t_n, y_n)$; $k_2 = f(t_n + \frac{\tau}{2}, y_n + \frac{\tau}{2} k_1)$;

$k_3 = f(t_n + \tau, y_n - \tau k_1 + 2\tau k_2)$.

Метод Рунге-Кутта 4-го порядка точности

$$\frac{y_{n+1} - y_n}{\tau} = \frac{1}{6}(k_1 + 2\cdot k_2 + 2\cdot k_3 + k_4),$$

где: $k_1 = f(t_n, y_n)$; $k_2 = f(t_n + \frac{\tau}{2}, y_n + \frac{\tau}{2}\cdot k_1)$;

$k_3 = f(t_n + \frac{\tau}{2}, y_n + \frac{\tau}{2}\cdot k_2)$; $k_4 = f(t_n + \tau, y_n + \tau \cdot k_3)$.

Методы Рунге-Кутта сходятся и порядок их точности совпадает с порядком аппроксимации разностным отношением.

Теорема. *Пусть правая часть уравнения* (6.4) $\boldsymbol{f(t,u)}$ *удовлетворяет условию Липшица по аргументу и с константой L, и пусть* δ_n*– невязка метода Рунге-Кутта. Тогда для погрешности метода при* $n \cdot \tau \le T$ *справедлива оценка*

$$\left| y_n - u(t_n) \right| \le T e^{\alpha T} \max \left| \delta_n \right|,$$

где: $\alpha = \sigma L m (1 + Lb\tau)^{m-1}$; $\sigma = \max_i \left| \sigma_i \right|$; $b = \max_{i,j} \left| b_{ij} \right|$.

На практике обычно пользуются правилом Рунге. Для этого сначала проводят вычисления с шагом τ , затем – $\tau/2$. Если y_n^{τ}– решение при шаге τ, а $y_{2n}^{\tau/2}$– при шаге $\tau/2$, p – порядок точности метода, то справедлива оценка.

$$\left| y_{2n}^{\tau/2} - u(t_n) \right| \le \frac{\left| y_{2n}^{\tau/2} - y_n^{\tau} \right|}{2^p - 1}$$

Тогда за оценку погрешности для метода четвертого порядка точности при шаге $\tau/2$ принимают величину

$$\max_i \left| \frac{y_n^{\tau} - y_{2n}^{\tau/2}}{15} \right|.$$

6.2 Многошаговые разностные методы решения задачи Коши для обыкновенных дифференциальных уравнений

Рассмотрим задачу Коши для обыкновенного дифференциального уравнения

$$\frac{du}{dt} = f(t,u), \qquad (6.10)$$

где $u(0) = u_0$.

Для решения задачи Коши для уравнения (6.10) при $t>0$ введем равномерную сетку с постоянным шагом τ

$$\omega_\tau = \{ t_n = n \cdot \tau,\ n = 0,1,... \}.$$

Введем понятие линейного m шагового разностного метода для решения задачи (6.10). Линейным m-шаговым разностным методом называется система разностных уравнений

$$\frac{a_0 y_n + a_1 y_{n-1} + ... + a_m y_{n-m}}{\tau} = b_0 f_n + b_1 f_{n-1} + ... + b_m f_{n-m}, \qquad (6.11)$$

где: $n=m,m+1...$; a_k, b_k – числовые коэффициенты, не зависящие от n; $k=0,1,\ldots,m$, причем $a_0 \neq 0$.

Систему (6.11) будем рассматривать как рекуррентные соотношения, выражающие новые значения $y_n = y(t_n)$ через ранее найденные значения $y_{n-1}, y_{n-2},..., y_{n-m}$, причем расчет начинают с индекса $n=m$, т.е. с уравнения

$$\frac{a_0 y_m + a_1 y_{m-1} + ... + a_m y_0}{\tau} = b_0 f_m + b_1 f_{m-1} + ... + b_m f_0.$$

Отсюда следует, что для начала расчета по формулам (6.11) надо знать m предыдущих значений функции y, причем $y_0=u_0$ определяется исходной задачей (6.10). Эти предыдущие m значений могут быть найдены одним из одношаговых методов Рунге-Кутта.

Отличие от одношаговых методов состоит в том, что по формулам (6.11) расчет ведется только в точках сетки.

Определение. Метод (6.11) называется *явным*, если коэффициент $b_0=0$. Тогда значение y_n легко выражается через $y_{n-1}, y_{n-2},..., y_{n-m}$. В противном случае метод называется *неявным*, и для нахождения y_n придется решать нелинейное уравнение вида

$$\frac{a_0}{\tau} y_n - b_0 f\,(t_n, y_n) = \sum_{k=1}^{m} \left(b_k t_{n-k} - \frac{a_k}{\tau} y_{n-k} \right). \qquad (6.12)$$

Обычно это уравнение решают методом Ньютона при начальном значении $y_n^{(0)} = y_{n-1}$. Коэффициенты уравнения (6.11) определены с точностью до множителя, тогда, чтобы устранить этот произвол, вводят условие $\sum_{k=0}^{m} b_k = 1$, с тем условием, что правая часть (6.11) аппроксимирует правую часть дифференциального уравнения (6.10).

На практике используют частный случай методов (6.11), т.н. методы Адамса, когда производная $u'(t)$ аппроксимируется разностным отношением, включающим две соседние точки t_n и t_{n-1}. Тогда $a_0 = -a_1 = 1$; a_k=0, k=2,...,m и

$$\frac{y_n - y_{n-1}}{\tau} = \sum_{k=0}^{m} b_k f_{n-k}. \qquad (6.13)$$

Это и есть методы Адамса. При b_0=0 метод будет явным, в противном случае – неявным.

6.2.1 Задача подбора числовых коэффициентов $a_к$, $b_к$

Выясним, как влияют коэффициенты a_k, b_k на погрешность аппроксимации уравнения (6.11), на устойчивость и сходимость.

Определение. *Невязкой*, или погрешностью аппроксимации методов (6.11) называется функция

$$r_n = -\sum_{k=0}^{m} \frac{a_k}{\tau} u_{n-k} + \sum_{k=0}^{m} b_k f(t_{n-k}, u_{n-k}), \qquad (6.14)$$

которая получается в результате подстановки точного решения $u(t)$ дифференциального уравнения (6.10) в разностное уравнение (6.11) .

Если разложить функции $u_{n-k} = u(t_n - k \cdot \tau)$ в ряд Тейлора в точках $t = t_n$ равномерной сетки, окончательно получим функцию

$$r_n = -(\sum_{k=0}^{m} \frac{a_k}{\tau}) u(t_n) + \sum_{l=1}^{p} (\sum_{k=0}^{m} (-k\tau)^{l-1} (a_k \frac{k}{l} + b_k)) \frac{u^{(l)}(t_n)}{(l-1)!} + O(\tau^p). \qquad (6.15)$$

Из вида функции r_n следует, что порядок аппроксимации будет равен p, если выполнены условия

$$\begin{gathered} \sum_{k=0}^{m} a_k = 0; \\ \sum_{k=0}^{m} k^{l-1} (k a_k + l b_k) = 0; \\ \sum_{k=0}^{m} b_k = 1, \end{gathered} \qquad (6.16)$$

где l=1,...,p.

Условия (6.16) представляют собой систему линейных уравнений из $p+2$ уравнений относительно неизвестных $a_0,\ldots,a_m,\ b_0,\ldots,b_m$. Их количество равно $2(m+1)$. Решив систему (6.16), получаем неизвестные числовые коэффициенты. Для неявных m-шаговых методов наивысшим достижимым порядком аппроксимации будет $p=2m$, а для явных – $p=2m-1$.

Запишем систему (6.16) для методов Адамса

$$\begin{gathered} l\cdot\sum_{k=1}^{m}k^{l-1}b_k=1; \\ b_0=1-\sum_{k=1}^{m}b_k, \end{gathered} \tag{6.17}$$

где $l=2,\ldots,p$. Отсюда видно, что наивысший порядок аппроксимации для неявного m-шагового метода Адамса $p=m+1$, для явного ($b_0=0$) – $p=m$.

6.2.2 Устойчивость и сходимость многошаговых разностных методов

Наряду с системами уравнений (6.11) будем рассматривать т.н. однородные разностные уравнения вида

$$a_0V_n+a_1V_{n-1}+\ldots+a_mV_{n-m}=0, \tag{6.18}$$

где $n=m,m+1,\ldots$.

Будем искать его решение в виде функции

$$V_n=q^n,$$

где q – число, подлежащее определению. Подставив V_n в (6.18) получаем уравнение для нахождения q

$$a_0q^m+a_1q^{m-1}+\ldots+a_{m-1}q+a_m=0. \tag{6.19}$$

Уравнение (6.19) принято называть характеристическим уравнением для разностных методов (6.11). Говорят, что разностный метод (6.11) удовлетворяет условию корней, если все корни уравнения (6.19) $q_1,\ \ldots,\ q_m$ лежат внутри или на границе единичного круга комплексной плоскости, причем на границе нет кратных корней.

Разностный метод (6.11), удовлетворяющий условию корней, называется устойчивым методом.

Теорема. *Пусть разностный метод* (6.11) *удовлетворяет условию корней и выполнено условие* $\left|\frac{\partial f(t,u)}{\partial u}\right| \le L$ *при* $0 \le t \le T$. *Тогда при* $m \cdot \tau \le t_n = n \cdot \tau \le T$, $n \ge m$ *и достаточно малых* τ *будет выполнена оценка*

$$|y_n - u(t_n)| \le M\left(\max_{0 \le j \le m-1} |y_j - u(t_j)| + \max_{0 \le k \le n-m} |r_k|\right), \qquad (6.20)$$

где: r_k*– погрешность аппроксимации;* $y_j - u(t_j)$*– погрешность в задании начальных условий*; M *–константа , зависящая от* L,T *и не зависящая от* n.

Методы Адамса удовлетворяют условию корней, т.к. для них $a_0 = -a_1 = 1$, следовательно, $q = q_1 = 1$.

6.2.3 Примеры m-шаговых разностных методов Адамса

Явные методы. При $m=1$ порядок точности $p=1$. Тогда метод описывается формулой

$$\frac{y_n - y_{n-1}}{\tau} = f_{n-1}.$$

В этом случае получаем метод Эйлера. При $m=2$ порядок точности $p=2$. Тогда метод описывается формулой

$$\frac{y_n - y_{n-1}}{\tau} = \frac{3}{2} f_{n-1} - \frac{1}{2} f_{n-2}.$$

При $m=3$ порядок точности $p=3$. Тогда метод описывается формулой

$$\frac{y_n - y_{n-1}}{\tau} = \frac{1}{12}(23 f_{n-1} - 16 f_{n-2} + 5 f_{n-3}).$$

При $m=4$ порядок точности $p=4$. Метод описывается формулой

$$\frac{y_n - y_{n-1}}{\tau} = \frac{1}{24}(55 f_{n-1} - 59 f_{n-2} + 37 f_{n-3} - 9 f_{n-4}).$$

Неявные m-шаговые методы Адамса:

$m=1$, $p=2$, $\frac{y_n - y_{n-1}}{\tau} = \frac{1}{2}(f_n + f_{n-1})$ – метод трапеций;

$m=2$, $p=3$, $\frac{y_n - y_{n-1}}{\tau} = \frac{1}{12}(5 f_n + 8 f_{n-1} - f_{n-2})$;

$m=3,\ p=4,\ \dfrac{y_n - y_{n-1}}{\tau} = \dfrac{1}{24}(9f_n + 19f_{n-1} - 5f_{n-2} + f_{n-3})$.

Неявные методы содержат искомое значение y_n нелинейным образом, поэтому для его нахождения применяют итерационные методы решения нелинейных уравнений.

6.3 Численное интегрирование жестких систем обыкновенных дифференциальных уравнений

Жесткие системы можно сравнить с плохо обусловленными системами алгебраических уравнений.

Рассмотрим систему дифференциальных уравнений(ДУ)

$$\frac{d\boldsymbol{u}}{dt} = \boldsymbol{f}(t,\boldsymbol{u}), \tag{6.21}$$

где $\boldsymbol{u}(0) = \boldsymbol{u}_0$. Для решения (6.21) рассмотрим разностные методы вида

$$\frac{1}{\tau} \cdot \sum_{k=0}^{m} a_k \cdot y_{n-k} = \sum_{k=0}^{m} b_k \cdot f(t_{n-k}, y_{n-k}), \tag{6.22}$$

где $n = m,\ m+1,\ m+2,\ldots$.

Устойчивость и сходимость методов (6.22) определяется расположением корней характеристического уравнения, т.е. $|q| \le 1$ – корни принадлежат единичному кругу.

Среди методов (6.22) выделим те, которые позволяют получить асимптотически устойчивые решения.

Пример. В качестве частного случая (6.21) рассмотрим уравнение вида

$$\frac{du}{dt} = \lambda \cdot u, \tag{6.23}$$

где: $u(0) = u_0$; $\lambda < 0$; $u(t) = u_0 e^{\lambda t}$ – решение ДУ. При $\lambda < 0$ решение есть монотонно убывающая функция при $t \to \infty$. Для этого решения можно записать при любом шаге $\tau > 0$

$$|u(t+\tau)| \le |u(t)|, \tag{6.24}$$

что означает устойчивость решения $u(t)$.

Рассмотрим для задачи (6.23) метод Эйлера

$$\frac{y_{n+1}-y_n}{\tau}=\lambda\cdot y_n,$$

где: n=0,1,2,…, $y_{n+1}=q\cdot y_n$, q – промежуточный параметр, равный $1+\tau\lambda$.

Оценка (6.24) для метода Эйлера будет выполнена тогда и только тогда, когда $|q|\le 1$. Шаг τ лежит в интервале $0 < \tau \le 2/|\lambda|$. Метод Эйлера для задачи (6.23) устойчив при выполнении этого условия.

Определение 1. Разностный метод (6.22) называется абсолютно устойчивым, если он устойчив при любом $\tau>0$.

Определение 2. Разностный метод называется условно устойчивым, если он устойчив при некоторых ограничениях на шаг τ.

Например, метод Эйлера для (6.23) условно устойчив, при $0 < \tau \le 2/|\lambda|$. Примером абсолютно устойчивого метода является неявный метод Эйлера

$$\frac{y_{n+1}-y_n}{\tau}=\lambda\cdot y_{n+1},$$

т.к. в этом случае $|q|=\left|(1-\lambda\tau)^{-1}\right|<1$, при любых $\tau>0$.

Замечание. Условная устойчивость является недостатком явных методов в связи с тем, что приходится выбирать мелкий шаг интегрирования.

Пример для задачи (6.23). Если $\lambda=-200$, тогда $\tau \le 0.01$. Если мы рассмотрим интервал (0,1], то необходимо будет 100 шагов. Неявные методы со своей стороны приводят к решению на каждом шаге нелинейного уравнения, но это уже недостаток неявного метода.

6.3.1 Понятие жесткой системы ОДУ

Замечание. Все вышерассмотренные методы легко реализуются на примере одного уравнения и легко переносятся на системы ДУ, но при решении систем возникают дополнительные трудности, связанные с разномасштабностью описанных процессов.

Рассмотрим пример системы двух уравнений:

$$\begin{cases} \dfrac{d\,u_1}{dt} + a_1 \cdot u_1 = 0 \\ \dfrac{du_2}{dt} + a_2 \cdot u_2 = 0 \end{cases},$$

где: $t > 0$; $a_1, a_2 > 0$.

Эта система однородных независимых ДУ имеет решение

$$\begin{cases} u_1(t) = u_1(0) \cdot e^{-a_1 \cdot t} \\ u_2(t) = u_2(0) \cdot e^{-a_2 \cdot t} \end{cases}.$$

Это решение монотонно убывает с ростом t. Пусть коэффициент a_2 на порядок больше a_1, т.е. $a_2 >> a_1$. В этом случае компонента u_2 затухает гораздо быстрее u_1, и тогда, начиная с некоторого момента времени t , решение задачи $u(t)$ почти полностью будет определяться поведением компоненты u_1. Однако при численном решении данной задачи шаг интегрирования τ будет определяться, как правило, компонентой u_2, не существенной с точки зрения поведения решения системы. Рассмотрим метод Эйлера для решения данной системы

$$\begin{cases} \dfrac{u_1^{(n+1)} - u_1^{(n)}}{\tau} + a_1 \cdot u_1^{(n)} = 0 \\ \dfrac{u_2^{(n+1)} - u_2^{(n)}}{\tau} + a_2 \cdot u_2^{(n)} = 0 \end{cases}.$$

Он будет устойчив, если на шаг τ наложены ограничения

$$\begin{aligned} \tau \cdot a_1 &\le 2 \\ \tau \cdot a_2 &\le 2 \end{aligned}.$$

Учитывая, что $a_2 >> a_1 > 0$, получаем окончательное ограничение на τ

$$\tau \le \frac{2}{|a_2|}.$$

Такие трудности могут возникнуть при решении любых систем ОДУ.

Рассмотрим в качестве примера систему

$$\frac{d\boldsymbol{u}}{dt} = \boldsymbol{A} \cdot \boldsymbol{u}, \qquad (6.25)$$

где $\boldsymbol{A}$– квадратная матрица $m*m$.

Если матрица $\boldsymbol{A}$ имеет большой разброс собственных чисел, то возникают проблемы с разномасштабностью описываемых системой процессов.

Допустим, что матрица $\boldsymbol{A}$ постоянна (т.е. не зависит от t). Тогда система (6.21) будет называться жесткой, если:

1) вещественные части собственных чисел $\lambda_k < 0$ для всех k, где k=1,…,m;

2) число $S = \frac{\max\limits_{1 \le k \le m} |Re\,\lambda_k|}{\min\limits_{1 \le k \le m} |Re\,\lambda_k|}$ велико (десятки и сотни), и число S называется числом жесткости системы.

Если же матрица $\boldsymbol{A}$ зависит от t, то и собственные числа зависят от t и λ_k зависят от t.

Решение жесткой системы (6.25) содержит как быстро убывающие, так и медленно убывающие составляющие. Начиная с некоторого t >0 решение системы определяется медленно убывающей составляющей. При использовании явных разностных методов быстро убывающая составляющая отрицательно влияет на устойчивость, поэтому приходится брать шаг интегрирования τ слишком мелким.

6.3.2 Некоторые сведения о других методах решения жестких систем

На практике разностные методы (6.22) для решения жестких систем используются в виде методов Гира (неявный разностный метод) и метода матричной экспоненты (метод Ракитского).

6.3.2.1 Методы Гира

Это частный случай методов (6.22), когда коэффициент $b_0 = 1$, $b_1 = b_2 = ... = b_m = 0$. Запишем числовые коэффициенты,

которые определяются из условия p-го порядка точности аппроксимации системы разностными методами

$$a_0 = -\sum_{k=1}^{m} a_k \; ; \; \sum_{k=1}^{m} k \cdot a_k = -1; \; \sum_{k=1}^{m} k^l \cdot a_k = 0, \qquad (6.26)$$

где l=2,…,p.

Решив систему линейных уравнений (6.26) с учетом предыдущих условий, получаем все нужные коэффициенты.

Трехшаговый метод Гира (частный случай методов (6.22) с учетом условий (6.26)) имеет вид

$$\frac{11}{6} y_n - 3y_{n-1} + \frac{3}{2} y_{n-2} - \frac{1}{3} y_{n-3} = \tau \cdot f(t_n, y_n). \qquad (6.27)$$

Метод имеет третий порядок точности.

При m=4, получаем четырехшаговый метод Гира

$$\frac{25 \cdot y_n - 48 \cdot y_{n-1} + 36 \cdot y_{n-2} - 16 \cdot y_{n-3} - 3 \cdot y_{n-4}}{12 \cdot \tau} = \qquad (6.28)$$
$$= f(t_n, y_n).$$

Запишем систему (6.26) в виде

$$\begin{cases} a_1 + 2 \cdot a_2 + ... + m \cdot a_m = -1 \\ a_1 + 2^2 \cdot a_2 + ... + m^2 \cdot a_m = 0 \\ \quad ..... \\ a_1 + 2^m \cdot a_2 + ... + m^m \cdot a_m = 0 \end{cases}. \qquad (6.29)$$

Решив (6.29) для каждого случая можем найти коэффициенты a_k, k=1,2,…,m.

Чисто неявные разностные методы обладают хорошими свойствами устойчивости, поэтому используются для решения жестких систем уравнений.

6.3.2.2 Метод Ракитского (матричной экспоненты) решения систем ОДУ

$$\frac{d\boldsymbol{u}}{dt} = \boldsymbol{A}\cdot\boldsymbol{u}, \qquad (6.30)$$

где: $\boldsymbol{u} = (u_1,..., u_n)$; $\boldsymbol{u}(0) = \boldsymbol{u}_0$; $\boldsymbol{A}$– матрица размерности $n*n$.

Допустим, что матрица $\boldsymbol{A}$ – постоянная, т.е. ее элементы не зависят от времени. Система (6.30) – однородная, с постоянными коэффициентами. Запишем аналитическое решение (6.30)

$$\boldsymbol{u} = e^{\boldsymbol{A}t}\cdot\boldsymbol{u}_0, \qquad (6.31)$$

где $e^{\boldsymbol{A}t}$– матричная экспонента и

$$e^{At} = E + \boldsymbol{A}\cdot t + \frac{(\boldsymbol{A}\cdot t)^2}{2!} + ... + \frac{(\boldsymbol{A}\cdot t)^n}{n!} + ... \qquad (6.32)$$

Проинтегрируем уравнение (6.30) при значениях $t=\tau$, 2τ, 3τ,....

Если точно знать матрицу $e^{\boldsymbol{A}\cdot\tau}$, то точное решение в указанных точках можно получить по формуле (6.31), т.е. решение можно записать

$$\boldsymbol{u}\,|_{t=\tau} = e^{\boldsymbol{A}\cdot\tau}\cdot\boldsymbol{u}_0;$$
$$\boldsymbol{u}\,|_{t=2\tau} = e^{\boldsymbol{A}\cdot\tau}\cdot\boldsymbol{u}\Big|_{x=\tau};$$

Таким образом, задача сводится к тому, чтобы достаточно точно знать матрицу $e^{\boldsymbol{A}\cdot\tau}$. На практике поступают следующим образом: при больших τ нельзя воспользоваться рядом Тейлора в связи с его бесконечностью, т.е. для удовлетворительной точности пришлось бы взять много членов ряда, что трудно. Поэтому поступают так: отрезок $[0,\tau]$ разбивают на k частей, чтобы длина $h=\tau/k$ удовлетворяла условию $||\boldsymbol{A}\cdot h||<0.1$. Тогда запишем по схеме Горнера

$$e^{\boldsymbol{A}\cdot h} = \boldsymbol{E} + \boldsymbol{A}\cdot h(\boldsymbol{E} + \frac{\boldsymbol{A}\cdot h}{2}(\boldsymbol{E} + \frac{\boldsymbol{A}\cdot h}{3}(\boldsymbol{E} + \frac{\boldsymbol{A}\cdot h}{4}))).$$

Каждый столбец матрицы $e^{\boldsymbol{A}\cdot h}$ – w_j вычисляют по формуле

$$w_j = (e^{\boldsymbol{A}\cdot h})^k \cdot w_j^0,$$

где w_j^0 – вектор столбец, в i-ой строке которого 1, а в остальных – нули.

Если эта матрица найдена, то решение находится по (6.31).

Для исследования разностных методов при решении жестких систем рассматривают модельное уравнение

$$\frac{du}{dt} = \lambda \cdot u, \qquad (6.33)$$

где λ– произвольное комплексное число.

Для того, чтобы уравнение (6.33) моделировало исходную систему (6.30) его нужно рассматривать при таких значениях λ, которые являются собственными числами матрицы $\boldsymbol{A}$. Многошаговые разностные методы (6.31) имеют вид

$$\sum_{k=0}^{m}(a_k - \mu \cdot b_k)\cdot y_{n-k}, \qquad (6.34)$$

где: $n=m,\ m+1\ldots;\ \mu=\tau\cdot\lambda$.

Если решение уравнения (6.34) искать в виде $y_n = q^n$, то для нахождения числа q получим характеристическое уравнение вида

$$\sum_{k=0}^{m}(a_k - \mu \cdot b_k)\cdot q^{m-k} = 0.$$

Для устойчивости метода достаточно выполнения условия корней $|q_k| \le 1$. В случае жестких систем используются более узкие определения устойчивости.

Предварительные сведения. Областью устойчивости разностных методов называется множество всех точек комплексной плоскости $\mu=\tau\cdot\lambda$, для которых разностный метод применительно к уравнению (6.33) устойчив.

Определение 1. Разностный метод называется A-устойчивым, если область его устойчивости содержит левую полуплоскость $Re\,\mu<0$.

Замечание. Решение модельного уравнения (6.33) асимптотически устойчиво при значениях $Re\mu<0$, поэтому сущность A-устойчивого метода заключается в том, что A-устойчивый разностный метод является абсолютно устойчивым, если устойчиво решение исходного дифференциального уравнения.

Так как класс A-устойчивых методов узок, то пользуются $A(\alpha)$-устойчивым методом.

Определение 2. Разностный метод (6.31) называется $A(\alpha)$-устойчивым, если область его устойчивости содержит угол меньший α, т.е. $|arg(-\mu)|<\alpha$, где $\mu=\tau\cdot\lambda$.

Исходя из этого, определяется, что при $\alpha=90°=\pi/2$ $A(\pi/2)$ устойчивость совпадает с определением A-устойчивого метода.

6.4 Краевые задачи для обыкновенных дифференциальных уравнений

Постановка краевой задачи.

Рассматриваем дифференциальное уравнение порядка n $(n\geq 2)$

$$y^{(n)} = f(x, y, y', ..., y^{(n-1)}). \tag{6.35}$$

Если сделаем замену переменных вида

$$\begin{cases} y_0' = y_1; \\ y_1' = y_2; \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ y_{n-1}' = f(x, y_1, ..., y_{n-1}); \\ y_k(x_0) = y_0^{(k)}, \end{cases} \tag{6.36}$$

где: $y_k(x_0) = y_0^{(k)}$; $k=0,1,\ldots,(n-1)$, то задача (6.35) сводится к задаче Коши для нормальной системы ОДУ порядка n.

Типовые примеры краевых задач.

Рассмотрим дифференциальное уравнение

$$F(x, y, y', ..., y^{(n)}) = 0. \tag{6.37}$$

Для уравнения (6.37) краевая задача формулируется следующим образом: найти решение $y=y(x)$, удовлетворяющее уравнению (6.37), для которой значения ее производных в заданной системе точек $x=x_i$ удовлетворяют n независимым краевым условиям, в общем виде нелинейным. Эти краевые условия связывают значения искомой функции y и ее производных до (n-1) порядка на границах заданного отрезка.

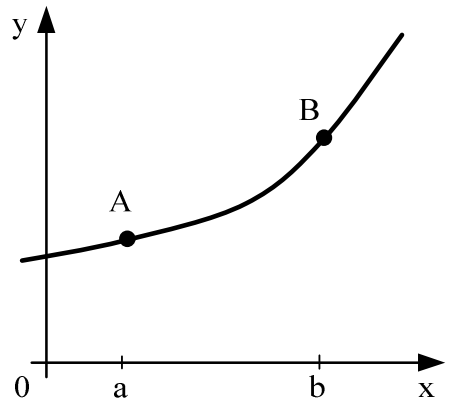


Рисунок 7 – Краевые условия для случая 1

1. Рассмотрим уравнение второго порядка $y''=f(x,y,y')$. Необходимо найти решение уравнения, удовлетворяющее заданным краевым условиям: $y(a)=$A, $y(b)=$B, т.е. необходимо найти интегральную кривую, проходящую через две заданные точки (рисунок 7).

2. Рассмотрим уравнение $y''=f(x,y,y')$ с краевыми условиями $y'(a)=A_1$, $y'(b)=B_1$.

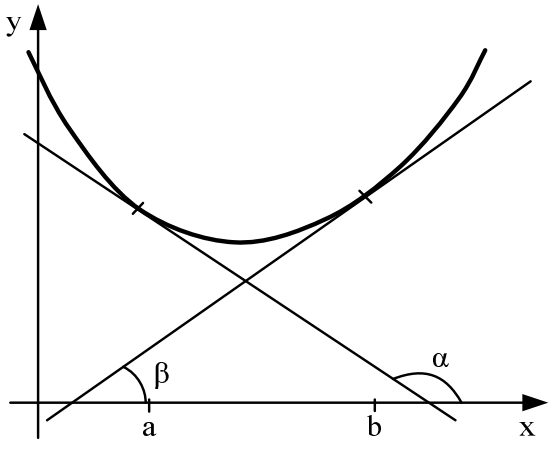


Рисунок 8 – Краевые условия для случая 2

Из графика на рисунке 8 видно, что $tg(\alpha)=A_1$, $tg(\beta)=B_1$.

Здесь интегральная кривая пересекает прямые $x=a$ и $x=b$ под заданными углами α и β соответственно.

3. Смешанная краевая задача включает оба случая.

Рассматривается то же самое уравнение $y'' = f(x, y, y')$, но с краевыми условиями $y(a)=A_1$, $y'(b)=B_1$.

Геометрическую иллюстрацию этих краевых условий легко представить, используя рисунки 7 и 8.

Замечание. Краевая задача для уравнения (6.37) в общем случае может не иметь решений, иметь единственное решение, иметь несколько решений или бесконечное множество решений.

4. Поражение заданной цели баллистическим снарядом. Дифференциальные уравнения движения снаряда с учетом сопротивления воздуха имеют вид

$$\begin{cases} \ddot{x} = -E\cos\Theta \\ \ddot{y} = -E\sin\Theta - g \end{cases},$$

где: $\ddot{x}$ – вторая производная по времени; $E=E(y,v)$ – известная функция высоты и скорости; $v=\sqrt{\dot{x}^2+\dot{y}^2}$; $g=g(y)$ – ускорение силы тяжести; Θ – угол наклона к горизонту касательной к траектории движения снаряда; $\Theta = arctg\dfrac{\dot{y}}{\dot{x}}$.

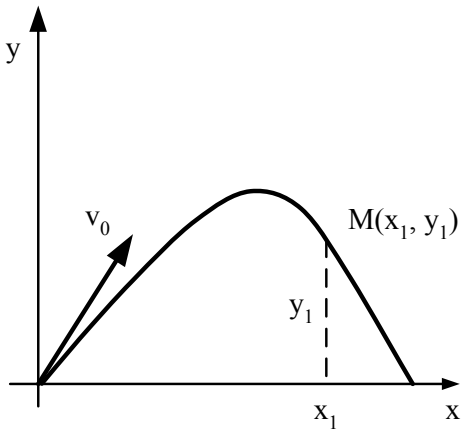


Рисунок 9 – Траектория снаряда

Предполагая, что при $t=t_0$ снаряд выпущен из точки, совпадающей с началом координат с начальной скоростью v_0 под углом Θ_0 , а в момент $t=t_1$ он поразит неподвижную мишень в точке $M(x_1, y_1)$ получаем краевые условия

$$\begin{cases} x = 0, \quad y = 0, \quad \dot{x} = v_0\cos\Theta_0, \quad \dot{y} = v_0\sin\Theta_0, \quad \text{при } t = t_0, \\ x = x_1, \quad y = y_1, \quad \text{при } t = t_1. \end{cases}$$

Здесь неизвестны значения Θ_0 и t_1. Решив данную краевую задачу, можем найти начальный угол $\Theta_0 = arctg\frac{\dot{y}_0}{\dot{x}_0}$, где: $\dot{x}_0 = \dot{x}(t_0); \dot{y}_0 = \dot{y}(t_0)$; Θ_0 – угол, при котором поражается цель в точке М.

Аналогично ставятся краевые задачи для систем дифференциальных уравнений.

6.5 Решение линейной краевой задачи

Рассмотрим важный частный случай решения краевой задачи, когда дифференциальное уравнение и краевые условия линейны.

Для этого рассмотрим уравнение

$$p_0(x)\cdot y^{(n)}(x) + p_1(x)\cdot y^{(n-1)}(x) + ... + p_n(x)\cdot y(x) = f(x), \qquad (6.38)$$

где: $p_i(x)$ и $f(x)$ известные непрерывные функции на отрезке $[a, b]$.

Предположим, что в краевые условия входят две абсциссы $x=a$, $x=b$. Это двухточечные краевые задачи. Краевые условия называются линейными, если они имеют вид

$$R_v(y) = \sum_{k=0}^{n-1}(\alpha_k^{(v)}\cdot y^{(k)}(a) + \beta_k^{(v)}\cdot y^{(k)}(b)) = \gamma, \qquad (6.39)$$

где: α, β, γ – заданные константы. Причем они одновременно не равны нулю, т.е.

$$\sum_{k=0}^{n-1}[|\alpha_k^{(\nu)}| + |\beta_k^{(\nu)}|] \neq 0, \text{ при } \nu = 1,2,\ldots,n.$$

Например, краевые условия во всех трех рассмотренных ранее задачах линейны, т.к. их можно записать в виде

$$\begin{aligned} \alpha_0\cdot y(a) + \alpha_1\cdot y'(a) &= \gamma_1 \\ \beta_0\cdot y(b) + \beta_1\cdot y'(b) &= \gamma_2 \end{aligned},$$

причем $\alpha_0 = 1, \alpha_1 = 0, \gamma_1 = A, \beta_0 = 1, \beta_1 = 0, \gamma_2 = B$ – для первой задачи.

6.6 Решение двухточечной краевой задачи для линейного уравнения второго порядка сведением к задаче Коши

Запишем линейное уравнение второго порядка в виде

$$y'' + p(x)\cdot y' + q(x)\cdot y = f(x), \qquad (6.40)$$

где: p, q, f – известные непрерывные функции на некотором отрезке $[a,b]$.

Требуется найти решение уравнения (6.40), удовлетворяющее заданным краевым условиям

$$\begin{aligned} \alpha_0\cdot y(a) + \alpha_1\cdot y'(a) &= A, \\ \beta_0\cdot y(b) + \beta_1\cdot y'(b) &= B. \end{aligned} \qquad (6.41)$$

Причем константы α и β одновременно не равны нулю

$$\begin{aligned} |\alpha_0| + |\alpha_1| &\neq 0, \\ |\beta_0| + |\beta_1| &\neq 0. \end{aligned}$$

Решение задачи (6.40), (6.41) будем искать в виде линейной комбинации

$$y = C\cdot u + V,$$

где C – константа, u – общее решение соответствующего однородного уравнения

$$u'' + p(x)\cdot u' + q(x)\cdot u = 0, \qquad (6.42)$$

а V– некоторое частное решение неоднородного уравнения

$$V'' + p(x)\cdot V' + q(x)\cdot V = f(x). \qquad (6.43)$$

Потребуем ,чтобы первое краевое условие было выполнено при любом C,

$$C(\alpha_0\cdot u(a) + \alpha_1\cdot u'(a)) + (\alpha_0\cdot V(a) + \alpha_1\cdot V'(a)) = A,$$

откуда следует, что $\alpha_0\cdot u(a) + \alpha_1\cdot u'(a) = 0,$

$$\alpha_0\cdot V(a) + \alpha_1\cdot V'(a) = A.$$

Тогда

$$\begin{aligned} u(a) &= \alpha_1 k \\ u'(a) &= -\alpha_0 k \end{aligned}, \qquad (6.44)$$

где k– некоторая константа, не равная нулю.

Значение функции V и ее производная в точке a могут быть, например,выбраны равными

$$\begin{matrix} V(a) = A/\alpha_0 \\ V'(a) = 0 \end{matrix}, \tag{6.45}$$

если коэффициент $\alpha_0 \neq 0$ и

$$\begin{matrix} V(a) = 0 \\ V'(a) = A/\alpha_1 \end{matrix}, \tag{6.46}$$

если коэффициент $\alpha_1 \neq 0$.

Из этих рассуждений следует, что функция u – есть решение задачи Коши для однородного уравнения (6.42) с начальными условиями (6.44), а функция V – есть решение задачи Коши для неоднородного уравнения (6.43) с начальными условиями (6.45) или (6.46) в зависимости от условий. Константу C надо подобрать так, чтобы выполнялись условия (6.41) (вторая строчка) в точке $x=b$

$$C(\beta_0 \cdot u(b) + \beta_1 \cdot u'(b)) + \beta_0 \cdot V(b) + \beta_1 \cdot V'(b) = B.$$

Отсюда следует, что

$$C = \frac{B - (\beta_0 \cdot V(b) + \beta_1 \cdot V'(b)}{\beta_0 \cdot u(b) + \beta_1 \cdot u'(b)},$$

где знаменатель не должен быть равен нулю, т. е.

$$\beta_0 \cdot u(b) + \beta_1 \cdot u'(b) \neq 0. \tag{6.47}$$

Если условие (6.47) выполнено, то краевая задача (6.35), (6.36) имеет единственное решение. Если же (6.47) не выполняется, то краевая задача (6.35), (6.36) либо не имеет решения, либо имеет бесконечное множество решений.

6.7 Методы численного решения двухточечной краевой задачи для линейного уравнения второго порядка

6.7.1 Метод конечных разностей

Рассмотрим линейное дифференциальное уравнение

$$y'' + p(x) \cdot y' + q(x) \cdot y = f(x) \tag{6.48}$$

с двухточечными краевыми условиями

$$\begin{cases} \alpha_0 y(a) + \alpha_1 y'(a) = A \\ \beta_0 y(b) + \beta_1 y'(b) = B \end{cases}; \qquad (6.49)$$
$$(|\alpha_0| + |\alpha_1| \neq 0, \quad |\beta_0| + |\beta_1| \neq 0),$$

где: p, q, f – известные непрерывные функции на некотором отрезке $[a,b]$.

Одним из наиболее простых методов решения этой краевой задачи является сведение ее к системе конечно-разностных уравнений.

Основной отрезок $[a,b]$ делим на n равных частей с шагом $h=(b-a)/n$, то есть рассматриваем равномерную сетку $x_i = x_0 + i \cdot h$, $i=0,1,\ldots,n$.

В каждом внутреннем узле дифференциальное уравнение (6.48) аппроксимируем, используя формулы численного дифференцирования второго порядка точности

$$y'_i = \frac{y_{i+1} - y_{i-1}}{2 \cdot h}; \qquad (6.50)$$
$$y''_i = \frac{y_{i+2} - 2 \cdot y_i + y_{i+1}}{h^2},$$

где $i=1,\ldots, n-1$.

Для граничных точек $x_0 = a$ и $x_n = b$, чтобы не выходить за границы отрезка, используем формулы численного дифференцирования первого порядка точности

$$y'_0 = \frac{y_1 - y_0}{h}, \quad y'_n = \frac{y_{n-1} - y_n}{-h}. \qquad (6.51)$$

Используя отношение (6.50) исходное дифференциальное уравнение (6.48) аппроксимируем конечно-разностными уравнениями

$$\frac{y_{i+1} - 2 \cdot y_i + y_{i-1}}{h^2} + p_i \frac{y_{i+1} - y_{i-1}}{2 \cdot h} + q_i \cdot y_i = f_i, \qquad (6.52)$$

где i=1,…,n-1. Учитывая краевые условия, получим еще два уравнения

$$\begin{cases} \alpha_0 \cdot y_0 + \alpha_1 \dfrac{y_1 - y_0}{h} = A \\ \beta_0 \cdot y_n + \beta_1 \dfrac{y_{n-1} - y_n}{-h} = B \end{cases}. \qquad (6.53)$$

Таким образом, получена линейная система $n+1$ уравнений с $n+1$ неизвестными $y_0, y_1, ..., y_n$, представляющими собой значения искомой функции $y = y(x)$ в точках $x_0, x_1, ..., x_n$. Разностная схема (6.52)-(6.53) аппроксимирует краевую задачу (6.48)- (6.49) с порядком 1 по h за счет краевых условий. Решив эту систему, получим таблицу значений искомой функции y.

6.7.2 Метод прогонки (одна из модификаций метода Гаусса)

При применении метода конечных разностей к краевым задачам для дифференциальных уравнений второго порядка получается система линейных алгебраических уравнений с трехдиагональной матрицей, т.е. каждое уравнение системы содержит три соседних неизвестных. Для решения таких систем разработан специальный метод – «метод прогонки».

Для этого систему (6.52) перепишем в виде

$$y_{i+1} + m_i \cdot y_i + n_i \cdot y_{i-1} = \tilde{f}_i \cdot h^2 \qquad (6.54)$$

для внутренних точек (i=1,…,n-1),

где: $$m_i = \frac{2 - q_i \cdot h^2}{1 + \dfrac{p_i \cdot h}{2}} \; ; \; n_i = \frac{1 - \dfrac{p_i \cdot h}{2}}{1 + \dfrac{p_i \cdot h}{2}} ; \; \tilde{f}_i = \frac{f_i}{1 + \dfrac{p_i \cdot h}{2}} .$$

На концах отрезка x_0=a и x_n=b производные заменяем разностными отношениями

$$y'_0 = \frac{y_1 - y_0}{h} \quad \text{и} \quad y'_n = \frac{y_{n-1} - y_n}{-h} .$$

Учитывая эту замену, получим еще два уравнения

$$\begin{aligned} \alpha_0 \cdot y_0 + \alpha_1 \frac{y_1 - y_0}{h} &= A; \\ \beta_0 \cdot y_n + \beta_1 \frac{y_{n-1} - y_n}{-h} &= B. \end{aligned} \tag{6.55}$$

Обратим внимание на внешний вид записи системы (6.54), (6.55). В каждом уравнении системы присутствует три ненулевых элемента. В первом и последнем – по два ненулевых коэффициента.

Разрешая уравнение (6.54) относительно y_i, получим

$$y_i = \frac{\tilde{f}_i}{m_i} h^2 - \frac{1}{m_i} y_{i+1} - \frac{n_i}{m_i} y_{i-1}. \tag{6.56}$$

Предположим, что с помощью полной системы (6.54), (6.55) из уравнения (6.56) исключена неизвестная y_{i-1}. Тогда это уравнение примет вид

$$y_i = c_i (d_i - y_{i+1}), \tag{6.57}$$

где: c_i, d_i – некоторые коэффициенты; i=1,2,…,n-1. Отсюда

$$y_{i-1} = c_{i-1}(d_{i-1} - y_i).$$

Подставляя это выражение в уравнение (6.54), получим

$$y_{i+1} + m_i y_i + n_i c_{i-1}(d_{i-1} - y_i) = \tilde{f}_i h^2,$$

а отсюда

$$y_i = \frac{(\tilde{f}_i h^2 - n_i c_{i-1} d_{i-1}) - y_{i+1}}{m_i - n_i c_{i-1}}. \tag{6.58}$$

Сравнивая (6.57) и (6.58), получим для определения c_i и d_i рекуррентные формулы

$$c_i = \frac{1}{m_i - n_i c_{i-1}}; \quad d_i = \tilde{f}_i h^2 - n_i c_{i-1} d_{i-1}; \quad i = 1,\ldots,n\text{-}1. \tag{6.59}$$

Из первого краевого условия (6.55) и из формулы (6.57) при i=0 находим

$$c_0 = \frac{\alpha_1}{\alpha_0 h - \alpha_1}; d_0 = \frac{Ah}{\alpha_1}. \tag{6.60}$$

На основании формул (6.59), (6.60) последовательно определяются коэффициенты c_i, d_i (i=1,…,n-1) до c_{n-1} и d_{n-1} включительно (прямой ход).

Обратный ход начинается с определения y_n. Для этого из второго краевого условия (6.55) и из формулы (6.57) при $i=n$-1 найдем

$$y_n = \frac{\beta_0 h + \beta_1 c_{n-1} d_{n-1}}{\beta_0 h + \beta_1 (c_{n-1} + 1)}. \qquad (6.61)$$

Далее по формуле (6.57) последовательно находим $y_{n-1}, y_{n-2}, ..., y_0$.

Заметим, что метод прогонки обладает устойчивым вычислительным алгоритмом.

7 Приближенное решение дифференциальных уравнений в частных производных

В реальных физических процессах искомая функция зависит от нескольких переменных, а это приводит к уравнениям в частных производных от искомой функции. Как и для обыкновенных дифференциальных уравнений(ОДУ), в этом случае для выбора одного конкретного решения, удовлетворяющего уравнению в частных производных, кроме начальных условий, необходимо задавать дополнительные условия (т.е. краевые условия). Чаще всего такие задачи на практике не имеют аналитического решения и приходится использовать численные методы их решения, в том числе метод сеток, метод конечных разностей и так далее. Мы будем рассматривать класс линейных уравнений в частных производных второго порядка. В общем виде в случае двух переменных эти уравнения записываются в виде

$$A(x,y)\frac{\partial^2 u}{\partial x^2} + B(x,y)\frac{\partial^2 u}{\partial x \partial y} + C(x,y)\frac{\partial^2 u}{\partial y^2} + a(x,y)\frac{\partial u}{\partial x} + b(x,y)\frac{\partial u}{\partial y} + c(x,y)u = F(x,y), \qquad (7.1)$$

где: A, B, C, a, b, c – заданные непрерывные функции двух переменных, имеющие непрерывные частные производные, u – искомая функция. Для сокращения записи введем обозначения

$$u_{xx} = \frac{\partial u}{\partial x^2};\quad u_{xy} = \frac{\partial^2 u}{\partial x \partial y};\quad u_{yy} = \frac{\partial^2 u}{\partial y^2};\ u_x = \frac{\partial u}{\partial x};\qquad u_y = \frac{\partial u}{\partial y}.$$

Будем рассматривать упрощенную форму записи (7.1) вида

$$A(x, y)u_{xx} + B(x, y)u_{xy} + C(x, y)u_{yy} + a(x, y)u_x + \\ + b(x, y)u_y + c(x, y)u = F(x, y) \qquad (7.2)$$

и рассмотрим частный случай (7.2), когда $a=b=c=F\equiv 0$, т.е.

$$A(x, y)u_{xx} + B(x, y)u_{xy} + C(x, y)u_{yy} = 0. \qquad (7.3)$$

Путем преобразований уравнение (7.3) может быть приведено к каноническому виду (к одному из трех стандартных канонических форм) эллиптическому типу, гиперболическому типу, параболическому типу. Причем тип уравнения будет определяться коэффициентами A, B, C, a именно – знаком дискриминанта

$$D=B^2-4\cdot A\cdot C.$$

Если $D<0$, то имеем уравнение эллиптического типа в точке с координатами x, y; если $D=0$, то (7.3) – параболического типа; если $D>0$, то (7.3) – гиперболического типа; если D не сохраняет постоянного знака, то (7.3) – смешанного типа.

Замечание. Если A, B, C – константы, тогда каноническое уравнение (7.3) называется полностью эллиптического, параболического, гиперболического типа.

Введем понятие оператора Лапласа для сокращенной записи канонических уравнений вида

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}.$$

Используя это определение, запишем сокращенные канонические уравнения всех трех типов

1. $\Delta u=0$. Это уравнение эллиптического типа, так называемое уравнение Лапласа. В механике это уравнение опи-

сывает стационарные тепловые поля, установившееся течение жидкости и т.д.

2. $\Delta u = -f$, где f – заданная непрерывная функция. Это уравнение Пуассона имеет эллиптический тип и описывает процесс теплопередачи с внутренним источником тепла.

3. $a^2 \Delta u = \partial u / \partial t$, где a – константа. Не во всех уравнениях в качестве переменных будут выступать стандартные переменные x, y. Может быть также переменная времени. Это уравнение диффузии описывает процесс теплопроводности и является уравнением параболического типа.

4. $\frac{\partial^2 u}{\partial t^2} = a^2 \Delta u$, a– константа. Это уравнение гиперболического типа – так называемое волновое уравнение и оно описывает процесс распространения волн.

7.1 Метод сеток для решения смешанной задачи для уравнения параболического типа (уравнения теплопроводности)

Смешанная задача означает, что следует найти искомую функцию, удовлетворяющую заданному уравнению в частных производных, краевым, а так же начальным условиям. Различить эти условия можно в том случае, если одна из независимых переменных – время, а другая – пространственная координата. При этом условия, относящиеся к начальному моменту времени, называются начальными, а условия, относящиеся к фиксированным значениям координат – краевыми.

Рассмотрим смешанную задачу для однородного уравнения теплопроводности

$$\frac{\partial u}{\partial t} = k \frac{\partial^2 u}{\partial x^2}, \; k=\text{const}>0. \qquad (7.4)$$

Задано начальное условие

$$u(x,0) = f(x) \qquad (7.5)$$

и заданы краевые условия первого рода

$$u(0,t) = \mu_1(t);$$
$$u(a,t) = \mu_2(t). \qquad (7.6)$$

Требуется найти функцию $u(x,t)$, удовлетворяющую в области D ($0<x<a$, $0<t\le T$) условиям (7.5) и (7.6).

К задаче (7.4) – (7.6) приводит задача о распространении тепла в однородном стержне длины a, на концах которого поддерживается заданный температурный режим.

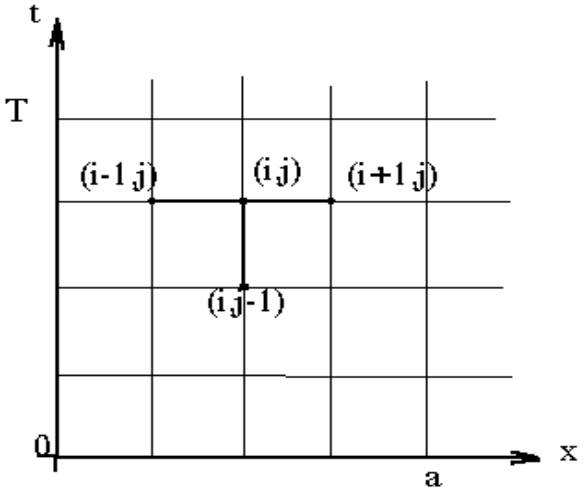


Рисунок 10 – Четырехточечный шаблон неявной схемы

При проведении замены $t \to t/k$ получим $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$, т.е. k=1. Задача решается *методом сеток*: строим в области D равномерную прямоугольную сетку с шагом h по оси x и шагом τ по оси t (см. рисунок 10).

Приближенные значения искомой функции $u(x_i,t_j)$ в точках (x_i,t_j) обозначим через u_{ij}.Тогда

$x_i = i \cdot h$; $h = a/n$; i=0,1,..., n; $t_j = j \cdot \tau$; j=0,1,..., m; $\tau = T/m$.

Заменим производные в (7.4) разностными отношениями

$$\frac{\partial u}{\partial t} = \frac{u_{i,j} - u_{i,j-1}}{\tau} + O(\tau);$$
$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + O(h^2).$$

В результате получим неявную двухслойную разностную схему с погрешностью $O(\tau+h^2)$

$$\frac{u_{i,j} - u_{i,j-1}}{\tau} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}$$

Используя подстановку $\lambda = \tau/h^2$, выразим из этой схемы $u_{i,j-1}$

$$u_{i,j-1} = (2\lambda + 1)u_{i,j} - \lambda u_{i+1,j} - \lambda u_{i-1,j}. \qquad (7.7)$$

Получаем разностную схему, которой аппроксимируем уравнение (7.4) во внутренних узлах сетки. Число уравнений меньше числа неизвестных u_{ij} . Из краевых условий получим уравнения

$$u_{0,j}=\mu_1(t_j); \quad u_{n,j}=\mu_2(t_j) ,$$

которые с (7.7) образуют неявную схему. Ее шаблон изображен на рисунке 10.

Получаем систему линейных уравнений с трехдиагональной матрицей. Решив ее любым способом (в частности, методом прогонки), получаем значения функции на определенных временных слоях. Так, на нулевом временном слое используем начальное условие $u_{i,0}=f(x_i)$, т.к. j=0. На каждом следующем слое искомая функция определяется из решения полученной системы. Неявная схема устойчива для любых значений параметра $\lambda = \tau / h^2 > 0$.

Есть и явная схема (рисунок 11), но она устойчива только при $\lambda \le 1/2$, т.е. при $\tau \le h^2/2$. Вычисления по этой схеме придется вести с малым шагом по τ, что приводит к большим затратам машинного времени.

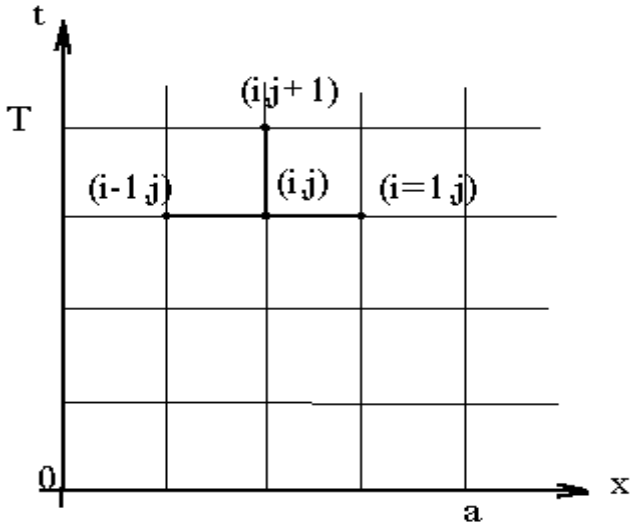


Рисунок 11 – Четырехточечный шаблон явной схемы

7.2 Решение задачи Дирихле для уравнения Лапласа методом сеток

Рассмотрим уравнение Лапласа

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 . \qquad (7.8)$$

Будем рассматривать уравнение Лапласа в прямоугольной области

$\Omega = \{(x, y), 0 \le x \le a, 0 \le y \le b\}$ с краевыми условиями

$$u(0, y) = f_1(y); \quad u(a, y) = f_2(y); \quad u(x,0) = f_3(x); \quad u(x, b) = f_4(x),$$

где f_1, f_2, f_3, f_4 – заданные функции. Заметим, что чаще всего область бывает не прямоугольной.

Введем обозначения $u_{ij}=u(x_i,y_j)$. Накладываем на прямоугольную область сетку $x_i = h \cdot i$; $i=0,1,\ldots,n$; $y_j = l \cdot j$; $j=0,1,\ldots,m$. Тогда $x_n = h \cdot n$, $y_m = l \cdot m = b$.

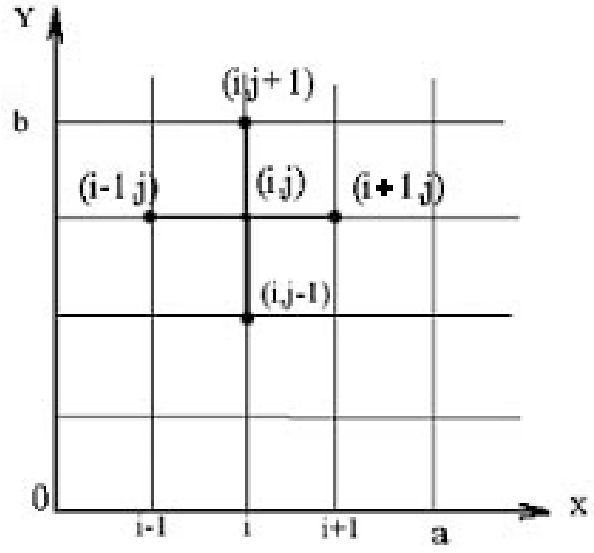


Рисунок 12 – Схема узлов «крест»

Частные производные аппроксимируем по формулам

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1} - 2 \cdot u_{i,j} + u_{i-1,j}}{h^2} + O(h^2);$$

$$\frac{\partial^2 u}{\partial y^2} = \frac{u_{i,j+1} - 2 \cdot u_{i,j} + u_{i,j-1}}{l^2} + O(l^2),$$

и заменим уравнение Лапласа конечно-разностным уравнением

$$\frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2 \cdot u_{i,j} + u_{i,j-1}}{l^2} = 0, \qquad (7.9)$$

где: $i=1,\ldots,n-1$, $j=1,\ldots,m-1$ (т.е. для внутренних узлов).

Погрешность замены дифференциального уравнения разностным составляет величину $O(h^2+l^2)$. Уравнения (7.9) и значения $u_{i,j}$ в граничных узлах образуют систему ли-

нейных алгебраических уравнений относительно приближенных значений функции $u(x,y)$ в узлах сетки. Выразим $u_{i,j}$ при $h=l$, и заменим систему

$$\begin{aligned} u_{ij} &= (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1})/4; \\ u_{i0} &= f_3(x_i); \\ u_{im} &= f_4(x_i); \\ u_{0j} &= f_1(y_j); \\ u_{nj} &= f_2(y_j). \end{aligned} \qquad (7.10)$$

Систему (7.10) линейных алгебраических уравнений можно решить любым итерационным методом (Зейделя, простых итераций и т.д.).

При построении системы использовалась схема типа «крест» (рисунок 12). Строим последовательность итераций по методу Зейделя

$$u_{i,j}^{(s+1)} = \frac{1}{4}(u_{i-1,j}^{(s+1)} + u_{i+1,j}^{(s)} + u_{i,j+1}^{(s)} + u_{i,j-1}^{(s+1)}),$$

где s– текущая итерация. Условие окончания итерационного процесса

$$\max_{i,j} \left| u_{ij}^{(s+1)} - u_{ij}^{(s)} \right| < \varepsilon \quad . \qquad (7.11)$$

Условие (7.11) ненадежно и на практике используют другой критерий

$$\max_{i,j} \left| u_{ij}^{(s+1)} - u_{ij}^{(s)} \right| \le \varepsilon(1-v),$$

где $$v = \frac{\max \left| u_{ij}^{(s+1)} - u_{ij}^{(s)} \right|}{\max \left| u_{ij}^{(s)} - u_{ij}^{(s-1)} \right|}.$$

Схема «крест»– явная устойчивая схема (малое изменение входных данных ведет к малому изменению выходных данных).

7.3 Решение смешанной задачи для уравнения гиперболического типа методом сеток

Рассмотрим уравнение колебания однородной и ограниченной струны.

Задача состоит в отыскании функции $u(x,t)$ при $t>0$, удовлетворяющей уравнению гиперболического типа

$$\frac{\partial^2 u}{\partial t^2} = c \cdot \frac{\partial^2 u}{\partial x^2}, \qquad (7.12)$$

где: $0<x< a$; $0<t\leq T$, начальным условиям

$$u(x,0) = f(x);$$

$$\frac{\partial u}{\partial t}(x,0) = g(x); \qquad (7.13)$$

$$0 \leq x \leq a$$

и краевым условиям

$$u(0,t) = \mu_1(t);$$

$$u(a,t) = \mu_2(t); \qquad (7.14)$$

$$0 \leq t \leq T.$$

Выполним замену переменных ct на t и получим уравнение

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

и в дальнейшем будем считать $c=1$.

Для построения разностной схемы решения задачи (7.12)-(7.14) построим в области $D = \{(x,t), 0 \leq x \leq a, 0 \leq t \leq T\}$ сетку $x_i = ih$; $i=0,1,\ldots,n$; $a = h \cdot n$; $t_j = j\tau$; $j=0,1,\ldots,m$; $\tau m = T$.

Аппроксимируем (7.12) в каждом внутреннем узле сетки на шаблоне типа «крест» (рисунок 12). Используем для аппроксимации частных производных разностные производные второго порядка точности относительно шага и получаем разностную аппроксимацию уравнения (7.12)

$$\frac{u_{i,j-1} - 2 \cdot u_{i,j} + u_{i,j+1}}{\tau^2} = \frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{h^2}, \qquad (7.15)$$

где u_{ij} – приближенное значение функции $u(x,t)$ в узле (x_i, t_j).

Полагая $\lambda=\tau/h$, перепишем (7.15), выразив $u_{i,j+1}$. Таким образом, получим трехслойную разностную схему

$$u_{i,j+1}=\lambda^2(u_{i+1,j}-u_{i-1,j})+2(1-\lambda^2)u_{i,j}-u_{i,j-1}, \qquad (7.16)$$

где: $i=1,\ldots,n$; $j=1,\ldots,m$. Задаем нулевые граничные условия $\mu_1(t)=0$, $\mu_2(t)=0$. Тогда в (7.16) $u_{0j}=0$, $u_{nj}=0$ для всех j.

Схема (7.16) называется трехслойной, т.к. она связывает значения искомой функции на трех временных слоях $j-1$, j, $j+1$. Схема (7.16) явная, т.е. позволяет в явном виде выразить u_{ij} через значения функции с предыдущих двух слоев.

Численное решение задачи состоит в вычислении приближенных значений u_{ij} решения $u(x,t)$ в узлах (x_i,t_j) при $i=1,\ldots,n$; $j=1,\ldots,m$. Алгоритм решения основан на том, что решение на каждом следующем слое ($j=2,3,..,n$) можно получить пересчетом решений с двух предыдущих слоев ($j=0,1,..,n-1$) по формуле (7.16). При $j=0$ решение известно из начального условия $u_{i0}=f(x_i)$. Для вычисления решения на первом слое ($j=1$) положим

$$\frac{\partial u}{\partial t}(x,0)\approx\frac{u(x,\tau)-u(x,0)}{\tau}, \qquad (7.17)$$

тогда $u_{i1}=u_{i0}-\tau g(x_i)$, $i=1,2,\ldots,n$. Для вычисления решений на следующих слоях можно использовать формулу (7.16). Решение на каждом последующем слое получается пересчетом решений с двух предыдущих слоев по формуле (7.16).

Описанная схема аппроксимирует задачу (7.12) – (7.14) с точностью $O(\tau+h^2)$. Невысокий порядок аппроксимации по τ объясняется использованием грубой аппроксимации для производной по t в формуле (7.17).

Схема будет устойчивой, если выполнено условие Куранта $\tau<h$. Это означает, что малые погрешности, возникающие при вычислении решения на первом слое, не будут неограниченно возрастать при переходе к каждому новому временному слою. Недостаток схемы в том, что сразу после выбора шага h в направлении x, появляется ограничение на величину шага τ по переменной t.

Лабораторная работа № 1
Решение систем линейных алгебраических уравнений

Точные методы

Метод Гаусса

Procedure SIMQ(*Nn:Integer;Var Aa:TMatr;*
Var Bb:TVector;Var Ks:Integer);

Входные параметры:

Nn – размерность системы;

Aa – матрица коэффициентов системы *N* ***N* типа *TMatr* = *array*[1..*n*, 1..*n*] *of real*;

Bb– вектор размерности *N* типа *TVector*= *array*[1..*n*] *of real*, содержит правые части системы.

Выходные параметры:

Bb – решение системы;

ks – признак правильности решения (код ошибки): если *ks*=0 то в массиве *Bb* содержится решение системы.

Пример. Решить систему уравнений

$$\begin{cases} x_1+x_2+x_3=6 \\ x_1-x_3=-2 \\ x_1+2x_2+x_3=8 \end{cases}$$

Схема алгоритма приведена на рисунке 13.

Текст процедуры:

```
PROCEDURE SIMQ(Nn:Integer;Var Aa:TMatr;Var Bb:TVector;Var Ks:Integer);
Label M1;
Const Eps=1e-21;
Var   Max,U,V  : Real;   I,J,K1,L : Integer;
Begin
For I:=1 To Nn Do Aa[i,Nn+1]:=Bb[i];
For I:=1 To Nn Do
 Begin
    Max:=Abs(Aa[i,i]);      K1:=I;
    For L:=I+1 To Nn Do  If (Abs(Aa[l,i])>Max) Then
   Begin
   Max:=Abs(Aa[l,i]);  K1:=L;End;
    If(Max<Eps) Then  Begin Ks:=1;    Goto M1;
    End   Else Ks:=0;
```

```
    If K1<>I Then
    For J:=I To Nn+1 Do
    Begin   U:=Aa[i,j];  Aa[i,j]:=Aa[k1,j]; Aa[k1,j]:=U;         End;
    V:=Aa[i,i];
    For J:=I To Nn+1 Do Aa[i,j]:=Aa[i,j]/V;
    For L:=i+1 To Nn Do Begin
    V:=Aa[l,i];   For J:=I+1 To Nn+1 Do Aa[l,j]:=Aa[l,j]-Aa[i,j]*V;
     End;        End;
   Bb[nn]:=Aa[Nn,Nn+1];
   For I:=Nn-1 Downto 1 Do Begin  Bb[i]:=Aa[i,nn+1];
   For J:=I+1 To Nn Do Bb[i]:=Bb[i]-Aa[i,j]*Bb[j];
   End;
  M1:End;
```

Вычисления по программе привели к следующим результатам:

x_1=0.100000E+0001 x_2=0.200000E+0001

x_3=0.300000E+0001

признак выхода 0

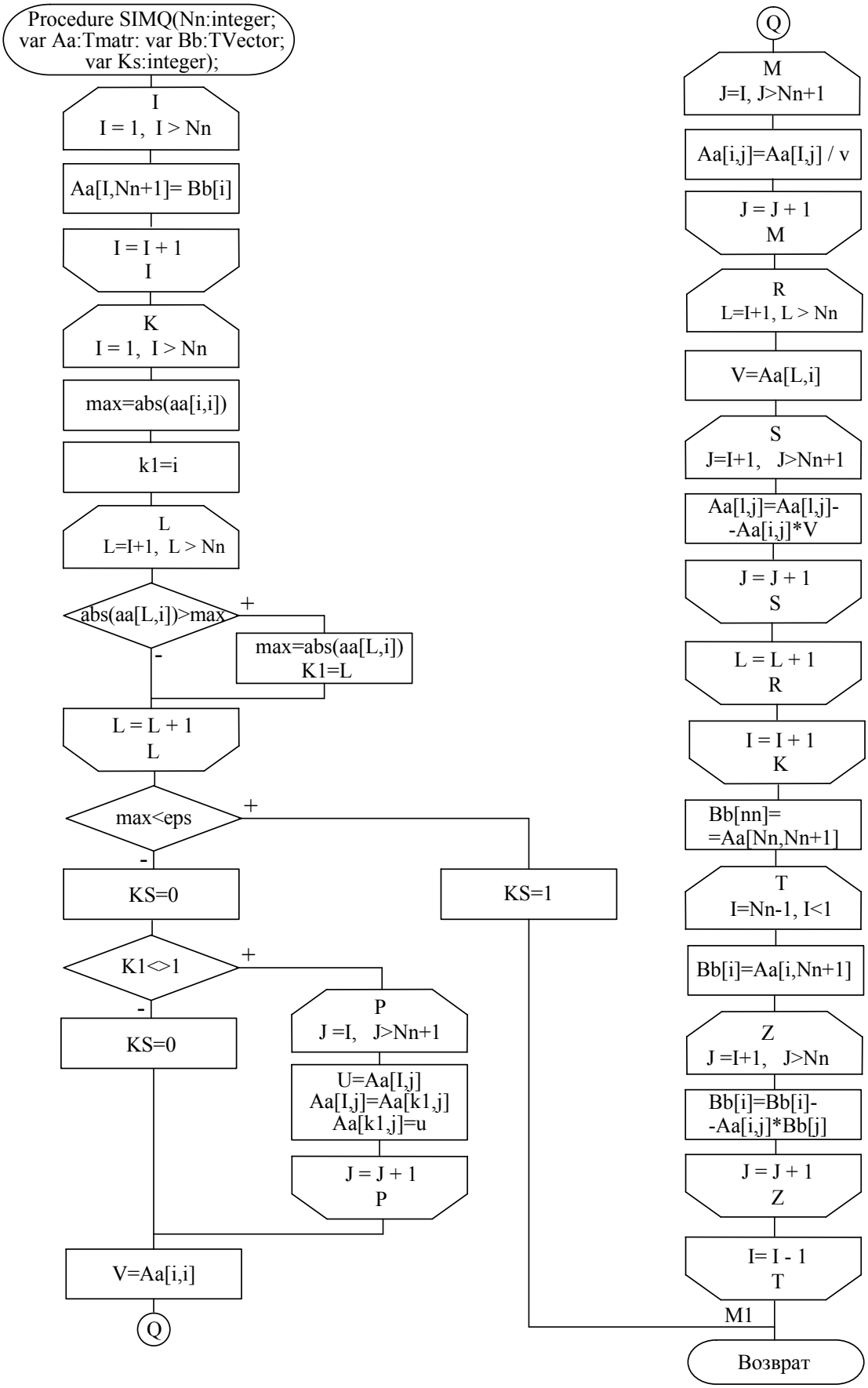


Рисунок 13 – Схема алгоритма метода Гаусса

Варианты заданий для решения систем линейных алгебраических уравнений методом Гаусса приведены в таблице 1.

Метод квадратных корней (Холецкого)

Procedure holet (*A*1*: Tmatr; var B*1*: Tvector;*
N: integer);

В х о д н ы е п а р а м е т р ы :

N – размерность системы;

*A*1– матрица коэффициентов системы *N* **N* +1, типа *Tmatr* = *array*[1..*n*, 1..*n+1*] *of real*, *N* +1 столбец используется для копирования левой части для упрощения вычислений;

*B*1– вектор свободных членов системы типа *Tvector*= *array*[1..*n*] *of real*.

В ы х о д н ы е п а р а м е т р ы :

*B*1– решение системы.

Пример. Решить систему уравнений:

$$\begin{cases} x_1+x_2+x_3=6 \\ x_1-x_3=-2 \\ x_1+2x_2+x_3=8 \end{cases}$$

Схема алгоритма приведена на рисунке 14.

Здесь произвольная матрица коэффициентов при неизвестных в системе уравнений умножением на транспонированную к ней матрицу приводится к симметричной.

Текст программы:

```
uses crt;
type
Tmatr=array[1..3,1..4]of real;
Tvector=array[1..3]of real;
const
{матрица размерности N*N , N+1 столбец используется для копирования
левой части СЛАУ для упрощения вычислений}
A:tmatr=   ((1,1,1,0),
            (1,0,-1,0),
            (1,2,1,0));
B:Tvector=       (6,-2,8);
var
k,N:integer;
{процедура для вывода матрицы}
```

```
Procedure vivod_matr(A1:Tmatr;N :integer);
var
i,j:integer;
begin
For i:=1 to N  do begin
For j:=1 to N  do write(A1[i,j]:15,' ');
writeln;
End;
writeln;
End;
{процедура для вывода вектора}
Procedure vivod_vectr(B1:Tvector; N :integer);
var
j:integer;
begin
For j:=1 to N  do writeln('x',j,'=',B1[j]:0,' ');
writeln;
End;
{метод Холецкого}
{
Входные параметры:
N – размерность системы;
A1– матрица коэффициентов системы N *N +1, N +1 столбец используется для копирования левой части СЛАУ для упрощения вычислений;
B1– вектор свободных членов системы.
Выходные параметры:
B1– решение системы.
B1-при вызове процедуры содержит вектор свободных членов СЛАУ,
Выходные параметры:
B1- содержит решение СЛАУ
}
Procedure holet(A1:Tmatr;var B1:Tvector;N :integer);
var    i,j,t:integer;
       c,L:Tmatr;
       y:Tvector;
       buf,summ:real;
Begin
For i:=1 to N  do
  For j:=1 to N +1 do
    begin
       c[i,j]:=0;
       L[i,j]:=0;
       y[i]:=0;
    End;
{Умножение матрицы на транспонированную}
For i:=1 to N  do
  For j:=1 to N  do
       begin
         summ:=0.0;
         For t:=1 to N  do
              summ:=A1[t,j]*A1[t,i]+summ;
         c[i,j]:=summ;
```

```
End;
{умножение правой части на транспонированную м-цу}
For i:=1 to N  do
  For j:=1 to N  do
        y[i]:=A1[j,i]*B1[j]+y[i];
For i:=1 to N  do For j:=1 to N  do
        begin
          A1[i,j]:=c[i,j];
          B1[i]:=y[i];
        End;
For i:=1 to N  do
  begin
    For j:=1 to i do
        begin
          summ:=0;
          For t:=1 to j-1 do
          summ:=summ+L[i,t]*L[j,t];
          if i<>j then
                L[i,j]:=(A1[i,j]-summ)/L[j,j]
          else
                L[i,i]:=sqrt(A1[i,i]-summ);
        End;
  End;
For i:=1 to N  do L[i,N +1]:=B1[i];
B1[1]:=L[1,N +1]/L[1,1];
For i:=2 to N  do
  begin
    For j:=1 to i-1 do
        L[i,N +1]:=L[i,N +1]-L[i,j]*B1[j];
    B1[i]:=L[i,N +1]/L[i,i];
  End;
For i:=1 to N  do
  begin
    For j:=i+1 to N  do
      begin
        L[i,j]:=L[j,i];
        L[j,i]:=0;
      End;
    L[i,N +1]:=B1[i];
End;
B1[N ]:=L[N ,N +1]/L[N ,N ];
For i:=N -1 downto 1 do
  begin
    For j:=i+1 to N  do
      L[i,N +1]:=L[i,N +1]-L[i,j]*B1[j];
    B1[i]:=L[i,N +1]/L[i,i];
  End;
End;

begin
N:=3;
writeln('dannaya SLAU');
```

```
{матрица задается типизированной константой}
vivod_matr(A,N);
writeln('metod Holetckogo');
holet(A,B,N);
vivod_vectr (B, N);
End.
```

Вычисления по программе привели к следующим результатам:

x_1= 0.1000000E+0001

x_2= 0.2000000E+0001

x_3= 0.3000000E+0001

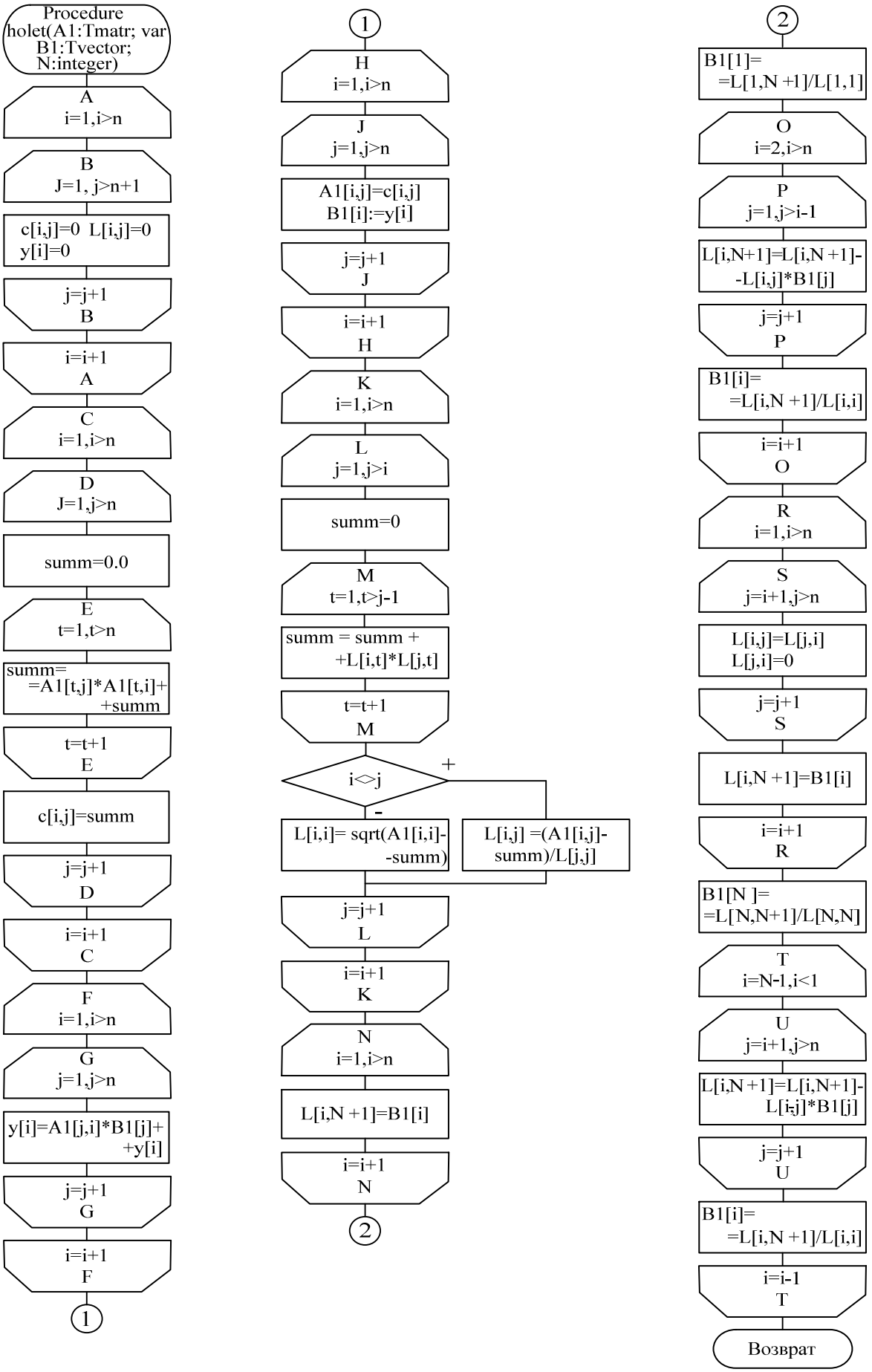


Рисунок 14 – Схема алгоритма метода Холецкого

Варианты заданий.
Решить систему линейных уравнений вида $Ax=b$

Таблица 1

Номер варианта	Матрица A коэффициентов системы			Столбец свободных членов b
1	2			3
1	1,84	2,25	2,53	-6,09
	2,32	2,60	2,82	-6,98
	1,83	2,06	2,24	-5,52
2	2,58	2,93	3,13	-6,66
	1,32	1,55	1,58	-3,58
	2,09	2,25	2,34	-5,01
3	2,18	2,44	2,49	-4,34
	2,17	2,31	2,49	-3,91
	3,15	3,22	3,17	-5,27
4	1,54	1,70	1,62	-1,97
	3,69	3,73	3,59	-3,74
	2,45	2,43	2,25	-2,26
5	1,53	1,61	1,43	-5,13
	2,35	2,31	2,07	-3,69
	3,83	3,73	3,45	-5,98
6	2,36	2,37	2,13	1,48
	2,51	2,40	2,10	1,92
	2,59	2,41	2,06	2,16
7	3,43	3,38	3,09	5,52
	4,17	4,00	3,65	6,93
	4,30	4,10	3,67	7,29
8	3,88	3,78	3,45	10,41
	3,00	2,79	2,39	8,36
	2,67	2,37	1,96	7,62
9	3,40	3,26	2,90	13,05
	2,64	2,39	1,96	10,30
	4,64	4,32	3,85	17,89
10	2,53	2,36	1,93	12,66
	3,95	4,11	3,66	21,97
	2,78	2,43	1,94	13,93
11	2,16	1,96	1,56	13,16
	3,55	3,23	2,78	21,73
	4,85	4,47	3,97	29,75

Продолжение таблицы 1

1	2			3
12	2,69	2,47	2,07	19,37
	2,73	2,39	1,92	19,43
	2.93	2,52	2,02	20,80
13	3,72	3,47	3,06	30,74
	4,47	4,10	3,63	36,80
	4,96	4,53	4,01	40,79
14	4,35	4,39	3,67	40,15
	4,04	3,65	3,17	36,82
	3,14	2,69	2,17	28,10
15	4,07	3,79	3,37	40,77
	2,84	2,44	1,95	27,68
	4,99	4,50	3,97	49,37
16	3,19	2,89	2,47	33,91
	4,43	4,02	3,53	47,21
	3,40	2,92	2,40	32,92
17	2,57	2,26	1,84	28,66
	4,47	4,03	3,57	50,27
	4,89	4,40	3,87	55,03
18	2,83	2,50	2,08	33,28
	3,00	2,55	2,07	33,59
	3,72	3,21	2,68	43,43
19	3,78	3,44	3,02	46,81
	4,33	3,88	3,39	53,43
	4,76	4,24	3,71	58,73
20	4,59	4,24	3,82	59,54
	4,83	4,36	3,88	62,33
	4,06	3,53	3,01	52,11
21	4,56	4,20	3,78	61,86
	3,21	2,73	2,25	42,98
	4,58	4,04	3,52	61,67
22	3,75	3,39	2,97	53,38
	4,18	3,70	3,22	59,28
	4,43	3,88	3,36	62,62
23	2,95	2,58	2,16	44,16
	5,11	4,62	4,14	46,68
	4,38	3,82	3,30	65,34
24	2,93	2,55	2,14	46,41
	3,47	2,98	2,50	54,78
	4,78	4,22	3,70	75,81

Продолжение таблицы 1

1	2			3
25	3,74	3,36	2,94	63,26
	4,02	3,51	3,04	67,51
	4,18	3,61	3,09	70,03
26	4,07	4,28	3,87	84,43
	5,30	4,79	4,32	95,45
	5,11	4,54	4,03	91,69
27	4,90	4,50	4,09	94,18
	3,79	3,27	2,81	71,57
	4,01	3,43	2,91	75,45
28	4,25	3,84	3,43	86,07
	3,86	3,34	2,87	77,12
	5,40	4,82	4,30	108,97
29	3,35	2,94	2,53	70,69
	5,41	4,88	4,41	115,38
	3,88	3,30	2,78	81,07
30	3,05	2,64	2,23	67,17
	4,14	3,61	3,14	91,43
	5,63	5,03	4,52	125,40

Лабораторная работа №2

Решение систем линейных алгебраических уравнений

Приближенные методы

Метод Якоби

procedure Yakoby (*var a: TMatr; var b: TVector;*
n: integer; var step: integer);

Входные параметры процедуры:

n – размерность системы;

a – матрица коэффициентов системы типа *TMatr* = *array*[1..*n*, 1..*n*+1] *of real*;

b – массив размерности *n*, содержит правые части системы.

Выходные параметры:

step – число итераций при решении системы;

b – решение системы.

Схема алгоритма приведена на рисунке 15.

Пример. Решить систему уравнений с точностью ε=0,01

$$\begin{cases} 100x_1+6x_2-2x_3=200 \\ 6x_1+200x_2-10x_3=600 \\ x_1+2x_2+100x_3=500. \end{cases}$$

Текст процедуры Yakoby:

```
procedure Yakoby(var a:TMatr; var b:TVector; n:integer;
                 var step:integer);
var x0,x:TVector;
    i,j,k:integer;
    e:real;
const eps = 0.01;
begin
    step:=0;
     for i:=1 to n do x0[i]:=b[i]/a[i,i];
     repeat
       for i:=1 to n do begin
         x[i]:=b[i]/a[i,i];
         for j:=1 to i-1 do x[i]:=x[i] - a[i,j]*x0[j]/a[i,i];
         for j:=i+1 to n do x[i]:=x[i] - a[i,j]*x0[j]/a[i,i];
       end;
       e:=0;
       for i:=1 to n do begin
```

```
        if abs(x[i] - x0[i])>e then e:=abs(x[i] - x0[i]);
        x0[i]:=x[i];
      end;
     inc(step);
    until e<=eps;
    b:=x0;
end;
```

Вычисления по программе привели к следующим результатам:

x_1=1,907
x_2=3,189
x_3=4,917
Количество итераций: 3.

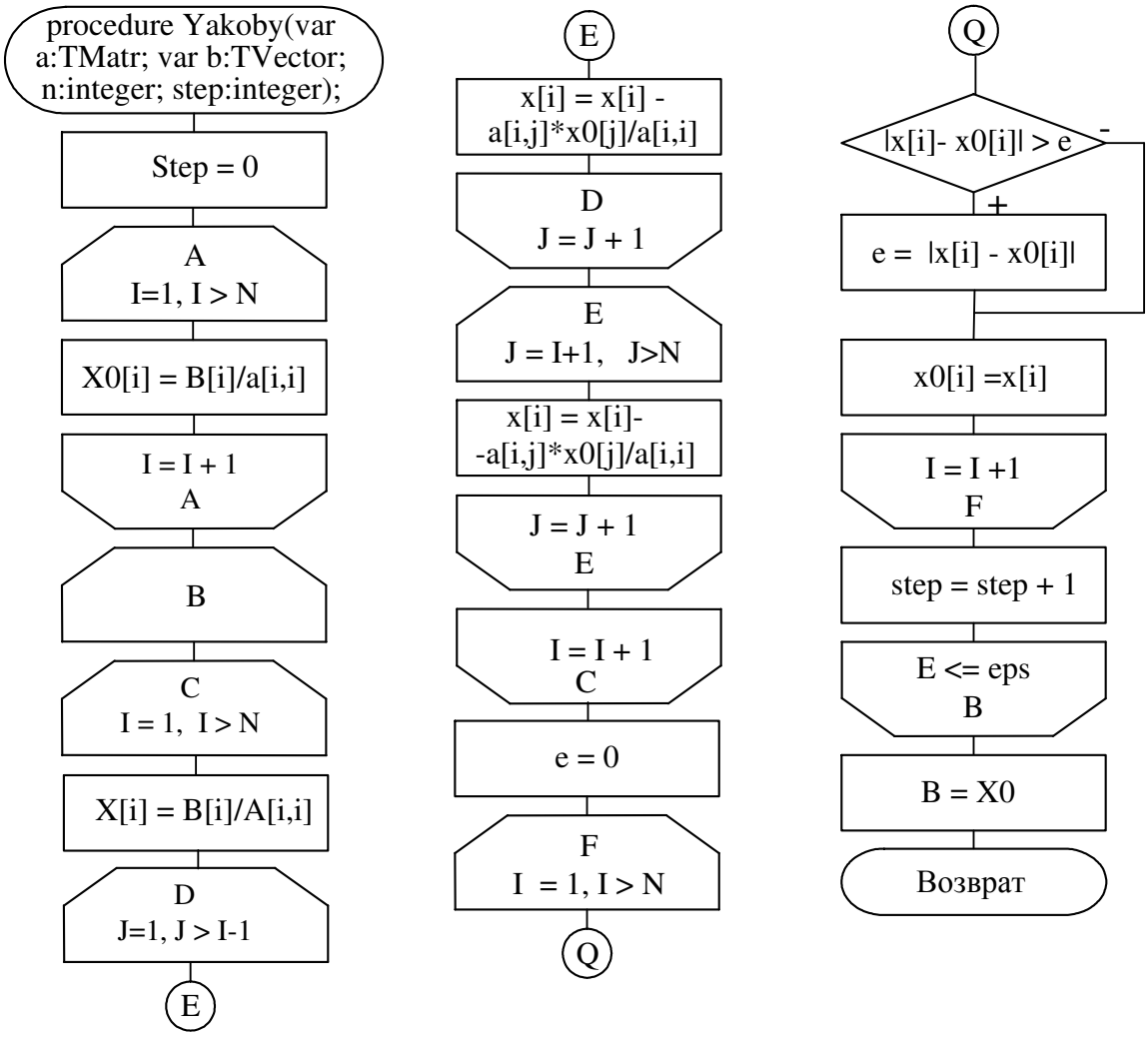


Рисунок 15 - Схема алгоритма метода Якоби

Варианты заданий для решения систем линейных алгебраических уравнений методом Якоби приведены в таблице 2.

Метод Зейделя

Procedure Zeidel(*var a: TMatr; var b: TVector; n: integer; step: integer*);

Входные параметры процедуры:

n – размерность системы;

a – массив, содержащий коэффициенты системы тип *TMatr*=*array*[1..*n*, 1..*n*+1] *of real*;

b – массив размерности n, содержит правые части системы;

Выходные параметры:

step – число итераций при решении системы;

b – решение системы.

Схема алгоритма приведена на рисунке 16.

Пример. Решить систему уравнений с точностью ε=0,01

$$\begin{cases} 100x_1+6x_2-2x_3=200 \\ 6x_1+200x_2-10x_3=600 \\ x_1+2x_2+100x_3=500. \end{cases}$$

Текст процедуры Zeidel:

```
procedure Zeidel(var a:TMatr;var b:TVector;n:integer; step:integer);
var x0,x:TVector;
   i,j,k:integer;
   e:real;
begin
step:=0;
   for i:=1 to n do x0[i]:=b[i]/a[i,i];
    repeat
      for i:=1 to n do begin
        x[i]:=b[i]/a[i,i];
        for j:=1 to i-1 do x[i]:=x[i] - a[i,j]*x[j]/a[i,i];
        for j:=i+1 to n do x[i]:=x[i] - a[i,j]*x0[j]/a[i,i];
      end;
      e:=0;
      for i:=1 to n do begin
        if abs(x[i] - x0[i])>e then e:=abs(x[i] - x0[i]);
        x0[i]:=x[i];
      end;
```

```
        inc(step);
      until e<=eps;
      b:=x0;
    end;
```

Вычисления по программе привели к следующим результатам:

x_1=1,907

x_2=3,188

x_3=4,917

Количество итераций: 3.

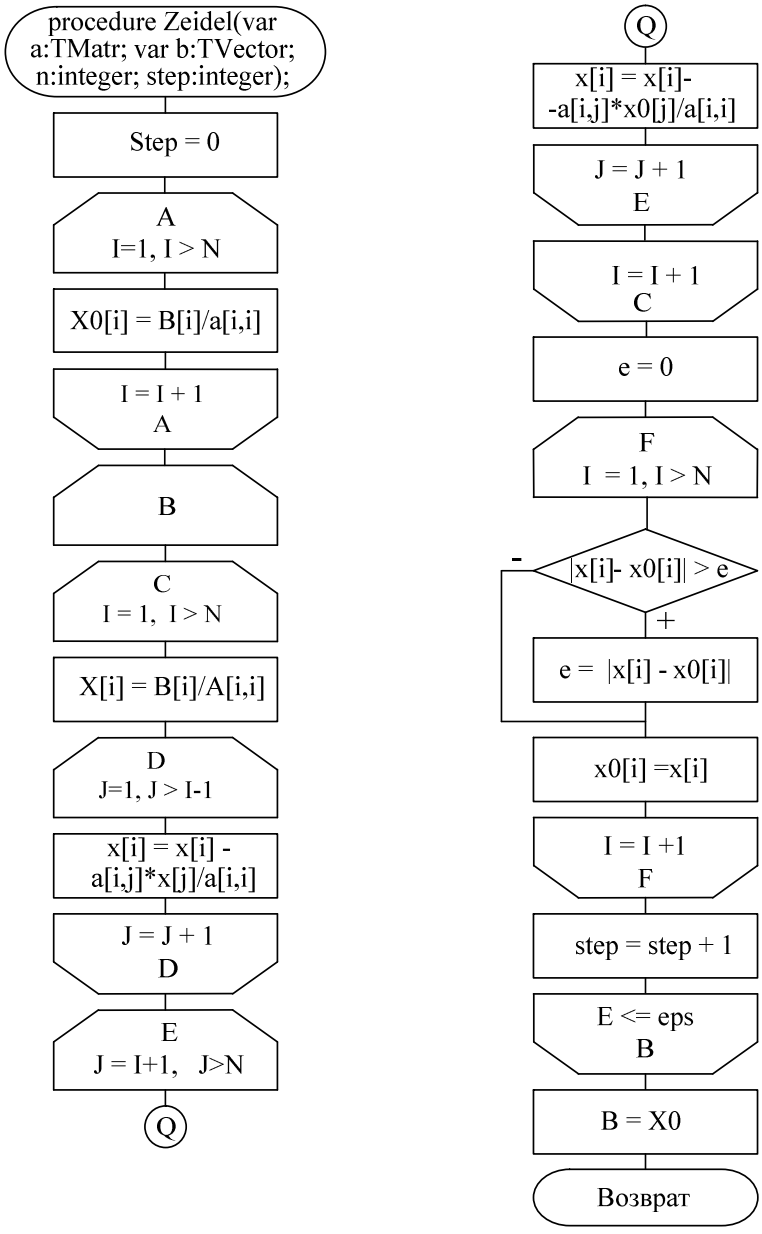


Рисунок 16– Схема алгоритма метода Зейделя

Варианты заданий для решения систем линейных алгебраических уравнений методом Зейделя приведены в таблице 2.

Метод верхней релаксации (обобщенный метод Зейделя)

Procedure ZeidelEx(*var a: TMatr; var b: TVector;*
n: integer; var step: integer; w: real);

Входные параметры процедуры:

n – размерность системы;

a – матрица коэффициентов системы типа *TMatr* = *array*[1..*n*, 1..*n*+1] *of real*;

b – массив размерности *n*, содержащий правые части системы;

w – числовой параметр, 0.2<*w*<1.8.

Выходные параметры:

step – число итераций при решении системы;

b – решение системы.

Схема алгоритма приведена на рисунке 17.

Пример. Решить систему уравнений методом релаксации с точностью ε=0,01 и параметром релаксации *w*=1,8 и *w*=0,2.

$$\begin{cases} 100x_1+6x_2-2x_3=200 \\ 6x_1+200x_2-10x_3=600 \\ x_1+2x_2+100x_3=500. \end{cases}$$

Текст процедуры ZeidelEx:

```
procedure ZeidelEx(var a:TMatr;var b:TVector;n:integer; var step:integer;
w:real);
var x0,x:TVector;
   i,j,k:integer;
   e:real;
const eps = 0.001;
begin
   step:=0;
    for i:=1 to n do x0[i]:=b[i]/a[i,i];
    repeat
      for i:=1 to n do begin
        x[i]:=w*b[i]/a[i,i]+(1-w)*x0[i];
```

```
            for j:=1 to i-1 do x[i]:=x[i] - w*a[i,j]*x[j]/a[i,i];
            for j:=i+1 to n do x[i]:=x[i] - w*a[i,j]*x0[j]/a[i,i];
          end;
          e:=0;
          for i:=1 to n do begin
            if abs(x[i] - x0[i])>e then e:=abs(x[i] - x0[i]);
            x0[i]:=x[i];
          end;
         inc(step);
        until e<=eps;
        b:=x0;
    end;
```

Вычисления по программе привели к следующим результатам:

при w=0,2

x_1=1,907

x_2=3,188

x_3=4,917

Количество итераций: 28.

при w=1,8

x_1=1,907

x_2=3,189

x_3=4,917

Количество итераций: 50.

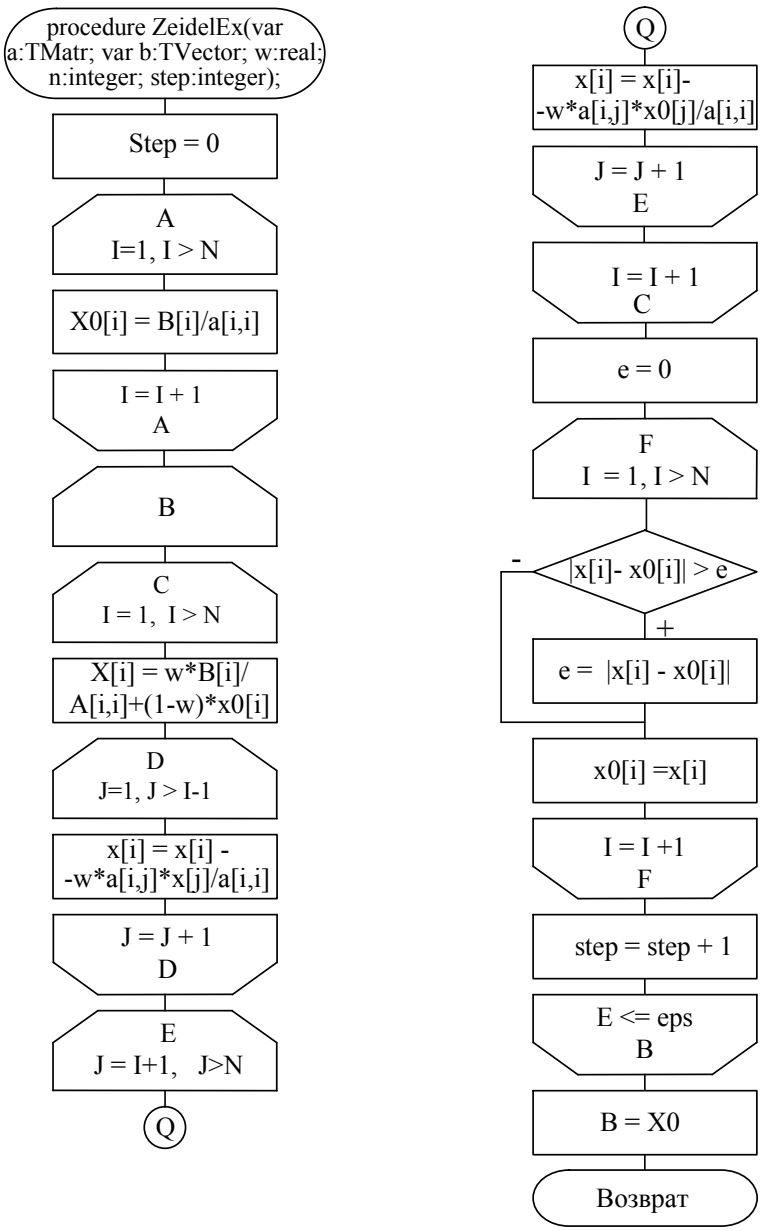


Рисунок 17 – Схема алгоритма метода релаксации

Варианты заданий для решения систем линейных алгебраических уравнений методом верхней релаксации приведены в таблице 2.

Текст программы:

```
uses crt;
const n:byte = 3;
      dw = 0.2;
type TMatr = array[1..4,1..4+1] of real;
    TVector = array[1..4] of real;
const eps = 0.001;
var  a:TMatr; b,b1:TVector;
    i,j,k:integer;
```

```
    w,buf:real;
    ch:char;
procedure Yakoby(var a:TMatr;var b:TVector; n:integer; var step:integer);
var x0,x:TVector;
   i,j,k:integer;
   e:real;
begin
   step:=0;
    for i:=1 to n do x0[i]:=b[i]/a[i,i];
    repeat
      for i:=1 to n do begin
        x[i]:=b[i]/a[i,i];
        for j:=1 to i-1 do x[i]:=x[i] - a[i,j]*x0[j]/a[i,i];
        for j:=i+1 to n do x[i]:=x[i] - a[i,j]*x0[j]/a[i,i];
      end;
      e:=0;
      for i:=1 to n do begin
        if abs(x[i] - x0[i])>e then e:=abs(x[i] - x0[i]);
        x0[i]:=x[i];
      end;
     inc(step);
    until e<=eps;
    b:=x0;
end;

procedure Zeidel(var a:TMatr;var b:TVector;n:integer; var step:integer);
var x0,x:TVector;
   i,j,k:integer;
   e:real;
begin
   step:=0;
   for i:=1 to n do x0[i]:=b[i]/a[i,i];
    repeat
      for i:=1 to n do begin
        x[i]:=b[i]/a[i,i];
        for j:=1 to i-1 do x[i]:=x[i] - a[i,j]*x[j]/a[i,i];
        for j:=i+1 to n do x[i]:=x[i] - a[i,j]*x0[j]/a[i,i];
      end;
e:=0;
      for i:=1 to n do begin
        if abs(x[i] - x0[i])>e then e:=abs(x[i] - x0[i]);
        x0[i]:=x[i];
      end;
     inc(step);
    until e<=eps;
    b:=x0;
end;

procedure ZeidelEx(var a:TMatr;var b:TVector;n:integer; var step:integer; w:real);
var x0,x:TVector;
   i,j,k:integer;
   e:real;
```

```
const eps = 0.0001;
begin
    step:=0;
     for i:=1 to n do x0[i]:=b[i]/a[i,i];
     repeat
       for i:=1 to n do begin
         x[i]:=w*b[i]/a[i,i]+(1-w)*x0[i];
         for j:=1 to i-1 do x[i]:=x[i] - w*a[i,j]*x[j]/a[i,i];
         for j:=i+1 to n do x[i]:=x[i] - w*a[i,j]*x0[j]/a[i,i];
       end;
       e:=0;
       for i:=1 to n do begin
         if abs(x[i] - x0[i])>e then e:=abs(x[i] - x0[i]);
         x0[i]:=x[i];
       end;
      inc(step);
     until e<=eps;
     b:=x0;
end;

function Proverka(a:TMatr):boolean;
var i,j:byte;
begin
    Proverka:=true;
    for i:=1 to n do
      if a[i,i] = 0 then begin
        proverka:=false; exit;
      end;
end;

procedure Out_Slau_T(a:tMatr; b:TVector);
var i,j:byte;
begin
    Writeln('Исходная система :');
    for i:=1 to n do begin
      for j:=1 to n do
        if a[i,j]<0 then write(' - ',abs(a[i,j]):0:4,'x',j)
          else write(' +', abs(a[i,j]):0:4,'x',j);
if b[i]<0 then write(' = ',' - ',abs(b[i]):0:4)
   else write(' = ',abs(b[i]):0:4);
      WriteLn;
    end;
    WriteLn;
end;

procedure ReadMatr(var a:tMatr);
var i,j:byte;
begin
    fillchar(a,sizeof(a),0);
    for i:=1 to n do begin
      Write('=>');
      for j:=1 to n do read(a[i,j]);
```

```
      Writeln;
    end;
 end;
procedure ReadVector(var b:TVector);
var i:byte;
begin
    fillchar(b,sizeof(b),0);
    Write('=>');
    for i:=1 to n do read(b[i]);
end;

begin
 ClrScr;
 a[1,1]:=100; a[1,2]:=6;     a[1,3]:=-2;    b[1]:=200;
 a[2,1]:=6;     a[2,2]:=200; a[2,3]:=-10;   b[2]:=600;
 a[3,1]:=1;     a[3,2]:=2;     a[3,3]:=100;   b[3]:=500;
Out_Slau_T(a,b);  b1:=b;

 Writeln('Решение по методу Якоби:');
 Yakoby(a,b,n,k);
 for i:=1 to n do write('x',i,' = ',b[i]:0:3,' ');
 Writeln('Число итераций = ',k);
 Writeln;
 b:=b1;

 Zeidel(a,b,n,k);
 Writeln(' Решение по методу Зейделя:');
 for i:=1 to n do write('x',i,' = ',b[i]:0:3,' ');
Writeln('Число итераций = ',k);
Writeln;

 w:=0.2;
 Writeln('Решение по обобщенному методу Зейделя:');
 while w<2.0 do begin
   b:=b1;
   ZeidelEx(a,b,n,k,w);
   for i:=1 to n do write('x',i,' = ',b[i]:0:3,'|');
   writeln('Число W = ',w:0:1,'Число k = ',k);
   w:=w+dw;
 end;

Writeln('Приступить к решению системы? [y/n]');
 Readln(ch);
 if ch<>'y' then halt(0);
 ClrScr;
 Writeln('Введите N');
 Readln(n);
Writeln;
 ReadMatr(a);
 ReadVector(b);
 b1:=b;
Out_Slau_T(a,b1);
```

```
Writeln("Решение по методу Якоби:');
 Yakoby(a,b,n,k);
 for i:=1 to n do write('x',i,' = ',b[i]:0:3,' ');
 Writeln("Число итераций = ',k);
Writeln;
 b:=b1;

 Zeidel(a,b,n,j,k);
 Writeln(" Решение по методу Зейделя:');
 for i:=1 to n do write('x',i,' = ',b[i]:0:3,' ');
 Writeln('Число итераций = ',k);
 Writeln;

 w:=0.2;
 Writeln(' Решение по обобщенному методу Зейделя:');
 while w<2.0 do begin
   b:=b1;
   ZeidelEx(a,b,n,k,w);
   for i:=1 to n do write('x',i,' = ',b[i]:0:3,'|');
   writeln("Число W = ',w:0:1,'|'Число итераций = ',k);
   w:=w+dw;
 end;
 ReadLn;Readln;
      end.
```

Вычисления по программе привели к следующим результатам:

```
BP.EXE
n
Исходная система :
 +100.0000x1 +6.0000x2 - 2.0000x3 = 200.0000
 +6.0000x1 +200.0000x2 - 10.0000x3 = 600.0000
 +1.0000x1 +2.0000x2 +100.0000x3 = 500.0000

Решение по методу Якоби:
x1= 1.907 x2= 3.189 x3= 4.917 Число итераций = 3

 Решение по методу Зейделя :
x1= 1.907 x2= 3.189 x3= 4.917 Число итераций = 3

Решение по обобщенному методу Зейделя :
x1= 1.907 | x2= 3.188 | x3= 4.917 | Число W=0.2 | Число k=28
x1= 1.907 | x2= 3.189 | x3= 4.917 | Число W=0.4 | Число k=14
x1= 1.907 | x2= 3.189 | x3= 4.917 | Число W=0.6 | Число k=9
x1= 1.907 | x2= 3.189 | x3= 4.917 | Число W=0.8 | Число k=6
x1= 1.907 | x2= 3.189 | x3= 4.917 | Число W=1.0 | Число k=4
x1= 1.907 | x2= 3.189 | x3= 4.917 | Число W=1.2 | Число k=7
x1= 1.907 | x2= 3.189 | x3= 4.917 | Число W=1.4 | Число k=11
x1= 1.907 | x2= 3.189 | x3= 4.917 | Число W=1.6 | Число k=20
x1= 1.907 | x2= 3.189 | x3= 4.917 | Число W=1.8 | Число k=50
Приступить к решению системы? [y/n]
n
```

Варианты заданий в таблице 2.

Таблица 2

Номер варианта	Матрица системы				Правая часть
1	2				3
1	.4000	.0003	.0008	.0014	.1220
	-.0029	-.5000	-.0018	-.0012	-.2532
	-.0055	-.0050	-1.4000	- .0039	- .9876
	-.0082	-.0076	-.0070	-2.3000	-2.0812
2	1.7000	.0003	.0004	.0005	.6810
	.0000	.8000	.0001	.0002	.4803
	-.0003	- .0002	-.1000	.0000	-.0802
	-.0005	-.0004	-.0003	-1.0000	-1.0007
3	3.0000	.0038	.0049	.0059	1.5136
	.0011	2.1000	.0032	.0043	1.4782
	-.0005	.0005	1.2000	.0026	1.0830
	-.0022	-.0011	-.0001	.3000	.3280
4	4.3000	.0217	.0270	.0324	2.6632
	.0100	3.4000	.0207	.0260	2.7779
	.0037	.0090	2.5000	.0197	2.5330
	- .0027	.0027	.0080	1.6000	1.9285
5	5.6000	.0268	.0331	.0393	4.0316
	.0147	4.7000	.0271	.0334	4.3135
	.0087	.0150	3.8000	.0274	4.2353
	.0028	.0090	.0153	2.9000	3.7969
6	6.9000	.0319	.0390	.0461	5.6632
	.0191	6.0000	.0333	.0405	6.1119
	.0134	.0205	5.1000	.0348	6.2000
	.0077	.0149	.0220	4.2000	5.9275
7	8.2000	.0370	.0451	.0532	7.5591
	.0234	7.3000	.0396	.0477	8.1741
	.0179	.0260	6.4000	.0422	8.4281
	.0124	.0205	.0286	5.5000	8.3210
8	9.5000	.0422	.0513	.0604	9.7191
	.0278	8.6000	.0459	.0550	10.5000
	.0224	.0315	7.7000	.0496	10.9195
	.0170	.0261	.0351	6.8000	10.9775
9	10.8000	.0475	.0576	.0676	12.1430
	.0321	9.9000	.0523	.0623	13.0897
	.0268	.0369	9.0000	.0570	13.6744
	.0215	.0316	.0416	8.1000	13.8972
10	12.1000	.0528	.0639	.0749	14.8310
	.0365	11.2000	.0586	.0697	15.9430
	.0312	.0423	10.3000	.0644	16.6926
	.0260	.0370	.0481	9.4000	17.0800

Продолжение таблицы 2

Номер варианта	Матрица системы				Правая часть
1	2				3
11	13.4000	.0581	.0702	.0822	17.7828
	.0408	12.5000	.0650	.0770	19.0599
	.0356	.0477	11.6000	.0718	19.9744
	.0304	.0425	.0546	10.7000	20.5261
12	14.7000	.0635	.0765	.0896	20.9985
	.0452	13.8000	.0714	.0844	22.4406
	.0400	.0531	12.9000	.0793	23.5195
	.0349	.0479	.0610	12.0000	24.2353
13	16.0000	.0688	.0829	.0970	24.4781
	.0496	15.1000	.0777	.0918	26.0849
	.0444	.0585	14.2000	.0867	27.3281
	.0393	.0534	.0674	13.3000	28.2078
14	17.3000	.0741	.0892	.1043	28.2215
	.0539	16.4000	.0841	.0992	29.9928
	.0488	.0639	15.5000	.0941	31.4001
	.0437	.0588	.0739	14.6000	32.4435
15	23.8000	.1010	.1212	.1414	50.8968
	.0757	22.9000	.1161	.1363	53.4873
	.0707	.0909	22.0000	.1313	55.7118
	.0656	.0858	.1060	1.1000	57.5703
16	19.9000	.0849	.1020	.1191	36.5001
	.0626	19.0000	.0969	.1140	38.5997
	.0576	.0747	18.1000	.1090	40.3345
	.0525	.0696	.0867	17.200	41.7045
17	21.2000	.0902	.1084	.1265	41.0351
	.0670	20.3000	.1033	.1215	41.2986
	.0619	.0801	19.4000	.1164	45.1968
	.0569	.0750	.0932	18.5000	46.7299
18	22.5000	.0956	.1148	.1339	45.8340
	.0714	21.6000	.1097	.1289	48.2611
	.0663	.0855	20.7000	.1238	50.3226
	.0612	.0804	.0996	19.8000	52.0184
19	23.8000	.1010	.1212	.1414	50.8968
	.0757	22.9000	.1161	.1363	53.4873
	.0707	.0909	22.0000	.1313	55.7118
	.0656	.0858	.1060	21.1000	57.5703
20	25.1000	.1063	.1276	.1488	56.2234
	.0801	24.2000	.1225	.1437	58.9772
	.0750	.0963	23.3000	.1387	61.3645
	.0700	.0912	.1124	22.4000	63.3853

Продолжение таблицы 2

Номер варианта	Матрица системы				Правая часть
1	2				3
21	26.4000	.1117	.1339	.1562	61.8139
	.0844	25.5000	.1289	.1512	64.7307
	.0794	.1017	24.6000	.1461	67.2806
	.0744	.0966	.1189	23.7000	69.4636
22	27.7000	.1171	.1403	.1636	67.6682
	.0888	26.8000	.1353	.1586	70.7478
	.0838	.1070	25.9000	.1536	73.4601
	.0788	.1020	.1253	25.0000	75.8051
23	29.0000	.1225	.1467	.1710	73.7864
	.0932	28.1000	.1417	.1660	77.0286
	.0882	.1124	27.2000	.1610	79.9030
	.0831	.1074	.1317	26.3000	82.4098
24	30.3000	.1278	.1531	.1784	80.1684
	.0975	29.4000	.1481	.1734	83.5730
	.0925	.1178	28.5000	.1684	86.6095
	.0875	.1128	.1381	27.6000	89.2778
25	31.6000	.1332	.1595	.1859	86.8143
	.1019	30.7000	.1545	.1809	90.3811
	.0969	.1232	29.8000	.1759	93.5793
	.0919	.1182	.1445	28.9000	96.4090
26	32.9000	.1386	.1659	.1933	93.7240
	.1062	32.0000	.1610	.1883	97.4528
	.1013	.1286	31.1000	.1833	100.8126
	.0963	.1236	.1510	30.2000	103.8034
27	34.2000	.1400	.1724	.2007	100.8.976
	.1106	33.3000	.1674	.1957	104.7881
	.1056	.1340	32.4000	.1907	108.3093
	.1006	.1290	.1574	31.5000	111.4610
28	35.5000	.1494	.1788	.2082	108.3351
	.1150	34.6000	.1738	.2032	112.3871
	.1100	.1394	33.7000	.1982	116.0694
	.1050	.1344	.1638	32.8000	119.3819
29	36.8000	.1547	.1852	.2156	116.0363
	.1193	35.9000	.1802	.2106	120.2497
	.1143	.1448	35.0000	.2056	124.0930
	.1094	.1398	.1702	31.1000	127.5660
30	38.1000	.1601	.1916	.2230	124.0015
	.1237	37.2000	.1866	.2180	128.3760
	.1187	.1502	36.3000	.2131	132.3800
	.1137	.1452	.1766	35.4000	136.0134

Лабораторная работа № 3

Решение плохо обусловленных систем линейных алгебраических уравнений

Метод регуляризации

Procedure regul(*Nn:Integer;a:Tmatr;b:Tvector;*
var X:Tvector; var p:integer);

Входные параметры:

nn – размерность системы;

a – матрица коэффициентов системы типа *Tmatr* = array [1..*n*, 1..*n*+1] of real;

b – массив размерности *nn*, содержащий столбец свободных членов системы.

Выходные параметры:

x – решение системы ;

p – количество итераций.

(в процедуре используются следующие процедуры:

Vozm(*nn:integer; eps:real;var a:tmatr; var b:tvector*) – изменяет вектор правой части системы на величину *eps*;

SIMQ(*Nn:Integer;Var Aa:TMatr;Var Bb:TVector;Var Ks:Integer*)– процедура решения системы методом Гаусса.)

Схема алгоритма приведена на рисунке 18.

Пример. Решить систему уравнений

$$\begin{cases} 1.03x_1 + 0.991x_2 = 2.51 \\ 0.991x_1 + 0.943x_2 = 2.41 \end{cases}$$

Текст процедуры regul:

```
type
    TMatr = array [1..2, 1..3] of real;
  TVector = array [1..2] of real;

procedure regul(Nn:Integer;a:Tmatr;b:Tvector;var X:Tvector; var p:integer);
var       a,a1,a2:tmatr;   b,b1,b2,x,x0:tvector;   alfa,s,max,eps:real;  i,j,k,l:integer;
procedure vozm(nn:integer; eps:real;var a:tmatr; var b:tvector);
var i,j:integer;
begin
for i:=1 to nn do b2[i]:=b2[i]+eps;
end;
begin
```

```
eps:=0.005;
For I:=1 To nn Do
  Begin
  For K:=1 To nn Do
    Begin
    S:=0.0;
    For J:=1 To nn Do S:=S+A[j,i]*A[j,k];
    A1[i,k]:=S;
    End;
  End;
  For I:=1 To nn Do
    Begin
    S:=0.0;
    For J:=1 To nn Do
       Begin S:=S+A[j,i]*B[j]; End;
    b1[i]:=S;
    End;
alfa:=0;  k:=0;
vozm(nn,eps,a2,b2);
repeat
  alfa:=alfa+1e-8;    inc(k);    a2:=a1;
  for i:=1 to nn do a2[i,i]:=a1[i,i]+alfa;
  for i:=1 to nn do b2[i]:=b1[i]+alfa*x0[i];
  a1:=a2;    b1:=b2;
  SIMQ(nn,a2,b2,l);
  a2:=a1;    X:=b2;    x0:=X;    b2:=b1;
  simq(nn,a2,b2,l);
  max:=abs(b2[1]-X[1]);
  for i:=2 to nn do
    if abs(b2[i]-X[i])>max then
      max:=abs(b2[i]-X[i]);
until max<eps; p=k;
end;
```

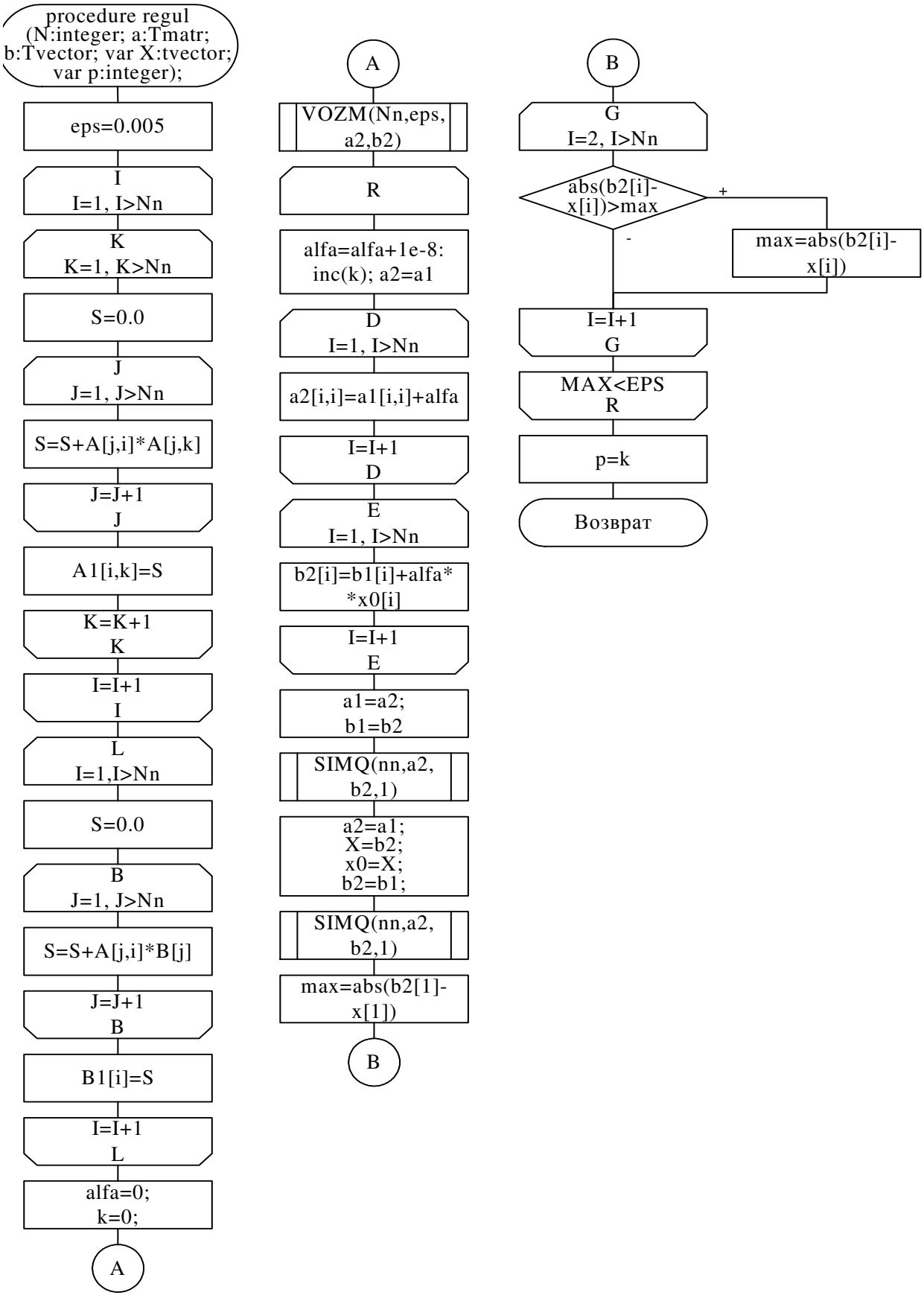


Рисунок 18 – Схема алгоритма метода регуляризации

Вычисления по программе привели к следующим результатам:

x_1=1.9810
x_2=0,4738

При возмущении правой части системы на 0,005 получаем следующий результат:

x_1=-6,8901
x_2=7,6810

Варианты заданий для решения плохо обусловленных систем методом регуляризации приведены в таблице 3.

Метод вращения (Гивенса)

Procedure Vrash (*Nn:integer; Aa:Tmatr; b: Tvector;*
var x: tvector);

Входные параметры:

nn – размерность ситемы;

a – матрица (*nn*+1 * *nn*+1), содержащая коэффициенты системы;

b – массив из *nn* действительных чисел, содержащий столбец свободных членов системы.

Выходные параметры:

x – решение системы .

Схема алгоритма приведена на рисунке 19.

Пример. Решить систему уравнений

$$\begin{cases} 1.03x_1 + 0.991x_2 = 2.51 \\ 0.991x_1 + 0.943x_2 = 2.41 \end{cases}$$

Текст процедуры:

```
Procedure Vrash(Nn:integer; Aa:Tmatr; b: Tvector; var x:tvector);
 Var   I,J,K: Integer; M,L,R : Real; F1:TEXT; Label M1,M2;
 Begin
for i:=1 to Nn do
aa[i,0]:=b[i];
M:=0.0;
```

```
For I:=1 To Nn-1 Do Begin
For K:=I+1 To Nn Do
   Begin
   If (Aa[i,i]<>0.0) or (Aa[k,i]<>0.0) Then
      begin
      M:=Sqrt(Aa[i,i]*Aa[i,i]+Aa[k,i]*Aa[k,i]);
      L:=-1.0*Aa[k,i]/M;
      M:=Aa[i,i]/M;
      end
   else begin M:=1.0;L:=0.0; end;
For J:=1 To Nn Do Begin
  R:=M*Aa[i,j]-L*Aa[k,j];
  Aa[k,j]:=L*Aa[i,j]+M*Aa[k,j];
  Aa[i,j]:=R;
  End;
  R:=M*Aa[i,0]-L*Aa[k,0];
  Aa[k,0]:=L*Aa[i,0]+M*Aa[k,0];
  Aa[i,0]:=R;
 End;     End;
 For I:=Nn Downto 1 Do Begin
M:=0.0;
For K:=0 To Nn-I-1 Do Begin M:=M+Aa[0,nn-k]*Aa[i,nn-k];     End;
Aa[0,i]:=(Aa[i,0]-M)/Aa[i,i];          End;
for i:=1 to Nn do x[i]:=Aa[0,i];
End;
```

Вычисления по программе привели к следующим результатам:

x_1=1,9813

x_2=0,4735

При возмущении правой части системы на 0,005 получаем следующий результат:

x_1=-6,8928

x_2=7,6837

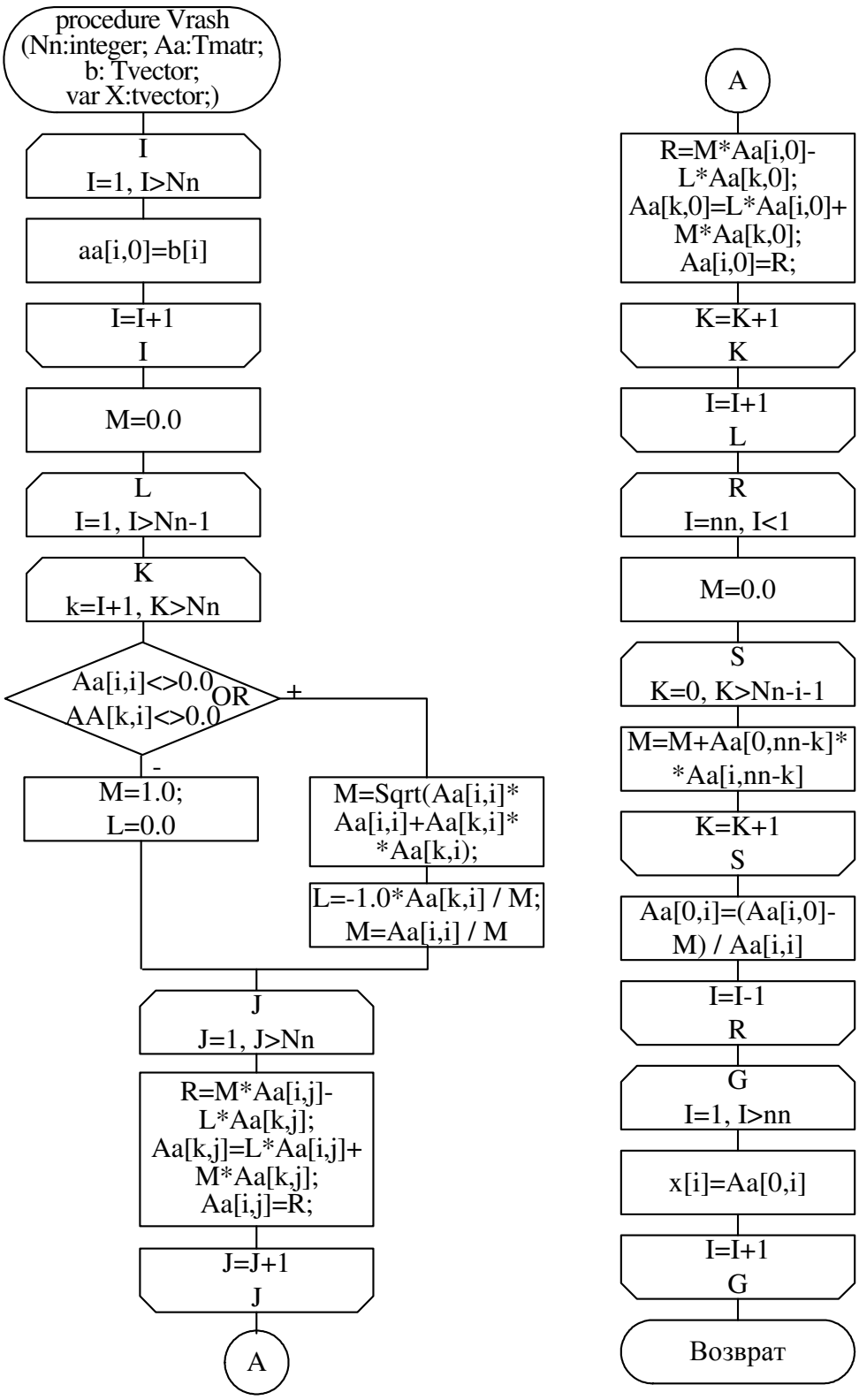


Рисунок 19 – Схема алгоритма метода Гивенса (вращения)

Варианты заданий для решения плохо обусловленных систем методом Гивенса приведены в таблице 3.

Варианты заданий

Таблица 3

№	Матрица ***A***		Вектор ***B***	№	Матрица ***A***		Вектор ***B***
1	1,03 0,991	0,991 0,940	2,51 2,41	16	1,03 0,991	0,991 0,943	2,51 2,40
2	1,04 0,991	0,992 0,941	2,52 2,42	17	1,03 0,991	0,991 0,943	2,62 2,41
3	1,05 0,991	0,992 0,942	2,53 2,43	18	1,03 0,991	0,991 0,943	2,73 2,42
4	1,06 0,991	0,994 0,943	2,54 2,44	19	1,03 0,991	0,991 0,943	2,84 2,43
5	1,07 0,991	0,995 0,944	2,55 2,45	20	1,03 0,991	0,991 0,943	2,95 2,41
6	1,08 0,991	0,996 0,944	0,502 0,482	21	1,03 0,991	0,991 0,943	2,56 2,41
7	1,09 0,991	0,997 0,945	0,502 0,482	22	1,03 0,991	0,991 0,943	2,57 2,49
8	1,03 0,991	0,998 0,946	0,502 0,482	23	1,03 0,991	0,991 0,943	2,58 2,48
9	1,03 0,991	0,999 0,947	0,502 0,482	24	1,03 0,991	0,991 0,943	2,59 2,47
10	1,03 0,991	0,996 0,948	0,502 0,482	25	1,03 0,991	0,2973 0,2829	2,50 2,46
11	1,03 0,991	0,995 0,949	2,51 2,45	26	1,03 0,991	0,2973 0,2829	2,51 2,45
12	1,03 0,991	0,994 0,951	2,52 2,44	27	1,03 0,991	0,2973 0,2829	2,51 2,44
13	1,03 0,991	0,993 0,953	2,53 2,43	28	1,03 0,991	0,2973 0,2829	2,51 2,43
14	1,03 0,991	0,992 0,952	2,54 2,42	29	1,03 0,991	0,2973 0,2829	2,51 2,42
15	1,03 0,991	0,991 0,940	2,55 2,41	30	1,03 0,991	0,2973 0,2829	2,51 2,41

Лабораторная работа № 4

Решение нелинейных уравнений и систем нелинейных уравнений

Метод простых итераций

Описание переменных, процедур и функций:

x0 – массив размерности *N*, содержит решение на предыдущей итерации;

x – массив размерности *N*, содержит решение на текущей итерации.

В программе используются следующие процедуры и функции:

function func(*x:vector;i:integer*)*: real;* - задание функции.

x –столбец значений неизвестных, *x*[1] – *x*, *x*[2] – *y*;

I – индекс нового значения неизвестной в столбце значений.

procedure vivod_vectr(*vector:vector;N*1,*N*2*:integer*)*;* – вывод вектора.

*N*1,*N*2 – используются для форматного вывода значений.

function Norma(*a:matr;N:integer*)*:real;* – вычисление нормы матрицы *a* размерности *N*N*.

function MatrJacobi(*x:vector;i,j:integer*)*:real;* - построение матрицы Якоби.

x –столбец значений неизвестных;

I,j – индексы элементов матрицы Якоби;

I – номер строки;

j – номер столбца.

procedure vivod_matr(*mat:matr;N*1,*N*2*:integer*)*;* - вывод матрицы.

Схема алгоритма приведена на рисунке 20.

Пример. Решить методом простых итераций систему:

$$\begin{cases} \cos(x+0.5)+y=0.8 \\ \sin y-2x=1.6 \end{cases}$$

В качестве нулевого приближения выберем точку x=0, y=0.

Преобразуем систему к виду:

$$\begin{cases} y = 0.8 - \cos(x + 0.5) \\ x = 0.5 * \sin y - 0.8 \end{cases}$$

Построим матрицу производных правой части системы

$$\varphi'(x, y) = \begin{pmatrix} \sin(x + 0.5) & 0 \\ 0 & 0.5\cos(y) \end{pmatrix}$$

Текст программы:

```
uses crt;
const
n=2;
eps=1e-4;
type
matr=array[1..n,1..n] of real;
vector=array[1..n]of extended;

var
a: matr;
x,f,x0:vector;
ITER,i,j: integer;
max: real;

{вычисление нормы}
Function Norma(a: matr; n: integer):real;
var    i,j: integer;
res: real;
begin
res:=0;
for i:=1 to n do
  for j:=1 to n do
     res:=res+A[i,j]*A[i,j];
Res:=sqrt(res);
Norma:=res;
end;

{задание функции}
{x –столбец значений переменных }
{I номер функции в системе }
function func(x:vector;i:integer):real;
begin
case i of
```

```
1:func:=  (sin(x[2])-1.6)/2;          {вычисляем значение первой функции  }
2:func:=0.8-cos((x[1])+0.5);              {вычисляем значение второй функции  }
end;
end;

{построение матрицы Якоби }
{x –столбец значений неизвестных }
{I,j индексы элементов  матрицы Якоби }
{I - номер строки}
{j – номер столбца}
function MatrJacobi(x:vector;i,j:integer):real;
begin
case i of
1:      case j of
          {вычисляем значение элемента матрицы Якоби индексами 1,1}
          1: MatrJacobi:=sin(x[1]+0.5);
{вычисляем значение элемента матрицы Якоби с индексами 1,2  }
        2: MatrJacobi:= 0 ;
end;
2:      case j of
{вычисляем значение элемента матрицы Якоби с индексами 2,1  }
        1: MatrJacobi:=0;
{вычисляем значение элемента матрицы Якоби с индексами 2,2}
        2: MatrJacobi:=0.5*cos(x[2]);
end;
end;
end;

{вывод матрицы}
procedure vivod_matr(mat:matr;N1,N2:integer);
var i,j: integer;
begin
for i:=1 to N do begin
        for j:=1 to N do write(mat[i,j]:n1:N2,'       ');
writeln;
end;
end;

{вывод вектора}
procedure vivod_vectr(vector:vector;N1,N2:integer);
var j: integer;
begin
for j:=1 to N do
          writeln('x',j,'= ',vector[j]:n1:N2);
end;

begin
clrscr;
x0[1]:=0;
x0[2]:=0;
iter:=0;
repeat
```

```
 for i:=1 to n do
       for j:=1 to n do
             a[i,j]:= MatrJacobi (x0,i,j);
 vivod_vectr(x0,3,13);
 {вычисление нормы матрицы A}
 writeln('норма =',Norma(a,N));
 {подсчет количества итераций}
 writeln('nomer iterazii - ',iter);
 writeln('=================');
 {нахождение нового приближения функции}
 for i:=1 to n do X[i]:=func(x0,i);
 max:=abs(X[1]-X0[1]);
 for i:=2 to n do if abs(X[I]-X0[I])>max then max:=abs(X[I]-X0[I]);
 X0:=X;
 inc(iter);
readln;
until (max<eps)or(iter>20);end.
```

Вычисления по программе привели к следующим результатам:

x_1=-0.8665

x_2=-0.1335

Norma=0.61155

Количество итераций: 9.

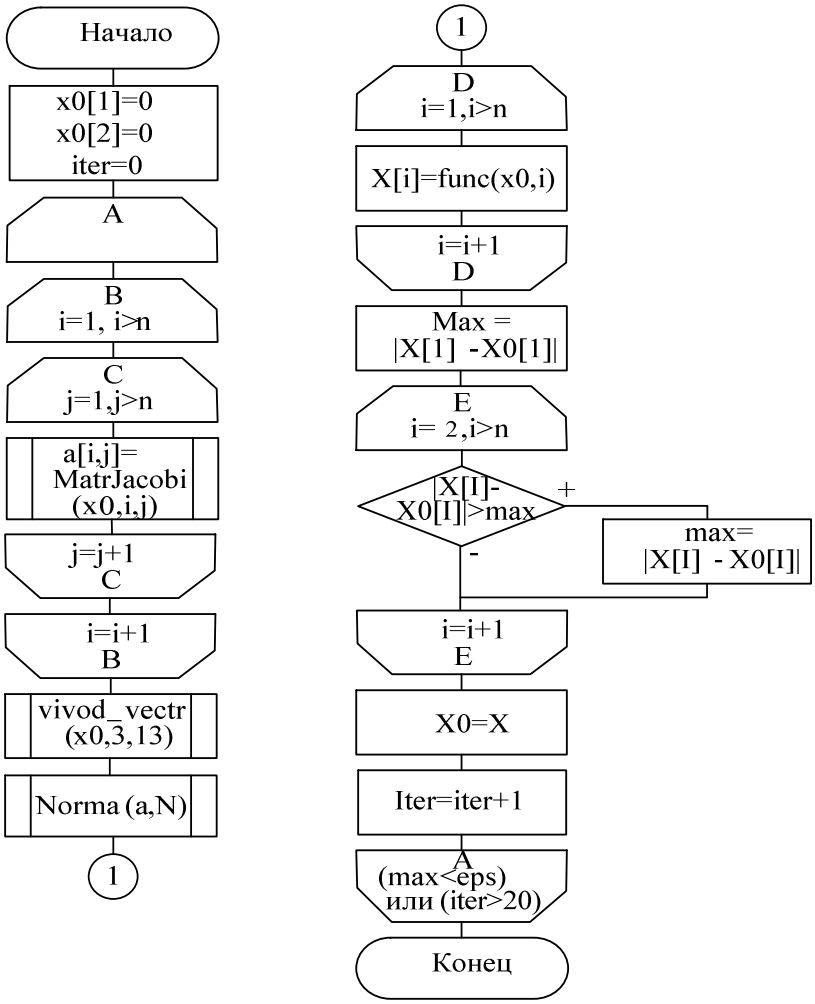


Рисунок 20 – Схема алгоритма метода простых итераций

Метод Ньютона

Входные параметры:

dx – массив размерности *N*, содержит решение на предыдущей итерации;

x – массив размерности *N*, содержит решение на текущей итерации;

A – массив размерности *N***N*+1 содержит коэффициенты системы размерности *N***N*, *N*+1 столбец используется для копирования правой части системы при решении.

В программе используются следующие процедуры и функции:

function func(x: vector; i:integer):real; - задание функции.

x – столбец значений неизвестных, *x*[1]=*x*, *x*[2]=*y*;

I – индекс нового значения неизвестной в столбце значений.

function jacobian(x:vector;i,j:integer):real; - задание элементов матрицы Якоби.

x –столбец значений неизвестных;

I,j – индексы элементов матрицы Якоби.

procedure vivod_matr(mat:matr;N1,N2:integer); – вывод матрицы.

*N*1,*N*2-используются для форматного вывода значений.

procedure vivod_vectr(vector:vector;N1,N2:integer); – вывод вектора

N1,N2-используются для форматного вывода значений.

procedure simq(Nn:integer;A:matr;var Bb:vector); – процедура решения системы линейных алгебраических уравнений методом Гаусса.

Схема алгоритма приведена на рисунке 21.

Пример. Решить методом Ньютона систему:

$$\begin{cases}\cos(x+0.5)+y=0.8\\ \sin y-2x=1.6\end{cases}$$

В качестве нулевого приближения выберем точку x=0, y=0.

Преобразуем систему к виду:

$$\begin{cases}\cos(x+0.5)+y-0.8=0\\ \sin y-2x-1.6=0\end{cases}$$

Текст программы:

```
Program Nuton_without_obr_matr
uses crt;
const
n=2;

type
matr=array[1..n,1..n] of double;
vector=array[1..n]of extended;
Tmatr=array[1..n,1..n+1]of extEnded;
```

```
const
 eps=1e-11;

var
a:matr;
x,f,dx:vector;
iter,i,j:integer;
max:real;

{задание функции}
{x –столбец значений неизвестных }
{I номер функции в системе }
function func(x:vector;i:integer):real;
begin
case i of
1:func:=cos(x[1]+0.5)+x[2]-0.8 ; {вычисляем значение первой функции  }
2:func:=sin(x[2])-2*x[1]-1.6 ; {вычисляем значение второй функцию  }
end;
end;

{задание якобиана}
{x –столбец значений неизвестных }
{I,j индексы элементов матрицы Якоби }
{I - номер строки}
{j – номер столбца}
function jacobian(x:vector;i,j:integer):real;
begin
case i of
1:    case j of
        1:jacobian:=-sin(x[1]+0.5)  ;      {вычисляем значение элемента матрицы Якоби с
индексами 1,1}
        2:jacobian:= 1 ;          {вычисляем значение элемента матрицы Якоби с индекса-
ми 1,2}
        end;
2:    case j of
        1:jacobian:=-2     ;            {вычисляем значение  элемента матрицы Якоби с ин-
дексами 2,1}
        2:jacobian:=cos(x[2]) ;      {вычисляем значение  элемента матрицы Якоби с ин-
дексами 2,2}
        end;
end;
end;

{вывод вектора}
procedure vivod_vectr(vector:vector;N1,N2:integer);
var j:integer;
begin
for j:=1 to N do
        writeln('x',j,'= ',vector[j]:n1:N2);
end;
```

```
{решение СЛАУ}
Procedure simq(Nn:integer;A:matr;var Bb:vector);
label m1;
const eps=1e-21;
var max,u,v:real;
ks,i,j,k1,l:integer;
Aa:Tmatr;
begin
For i:=1 to Nn do Aa[i,Nn+1]:=Bb[i];
For i:=1 to Nn do For j:=1 to Nn do Aa[i,j]:=A[i,j];
For i:=1 to Nn do
begin
        max:=abs(Aa[i,i]);
        k1:=i;
        For L:=i+1 to Nn do
if (abs(Aa[l,i])>max) then
                begin
                max:=abs(Aa[l,i]);
                k1:=l;
                End;

        if (max<eps)then
                begin
                ks:=1;
                goto m1;
                End
        else ks:=0;

        if k1<>i then
        For j:=1 to Nn+1 do
                begin
                U:=Aa[i,j];
                Aa[i,j]:=Aa[k1,j];
                Aa[k1,j]:=u;
                End;
        V:=Aa[i,i];
        For j:=1 to Nn+1 do Aa[i,j]:=Aa[i,j]/v;
        For l:=i+1 to Nn do
                begin
                v:=Aa[l,i];
                For j:=I+1 to Nn+1 do Aa[l,j]:=Aa[l,j]-Aa[i,j]*V;
                End;
End;
Bb[Nn]:=Aa[Nn,Nn+1];
For i:=Nn-1 downto 1 do
        begin
        Bb[i]:=Aa[i,Nn+1];
        For j:=i+1 to Nn do
Bb[i]:=Bb[i]-Aa[i,j]*Bb[j];
        End;
M1:
End;
```

```
begin
clrscr;
x[1]:=0;{x[1]=x}
x[2]:=0;{x[2]=y}
iter:=0;
repeat
  vivod_vectr(x,3,13);
  writeln('nomer iterazii - ',iter);
  writeln('================');
{подсчет количества итераций}
  inc(iter);
{вычисление значений элементов матрицы a}
  for i:=1 to n do
        for j:=1 to n do
              a[i,j]:=jacobian(x,i,j);
 {вычисление правой части СЛАУ}
  for i:=1 to n do f[i]:=-1*func(x,i);
{решение системы}
  simq(n,a,f);
  dx:=f;
  max:=abs(dx[1]);
  for i:=2 to n do if abs(dx[i])>max then max:=abs(dx[i]);
  for i:=1 to n do x[i]:=x[i]+dx[i];
until max<eps;
readln;
end.
```

Вычисления по программе привели к следующим результатам:

x_1= -0.8666
x_2= -0.1336
Количество итераций: 5.

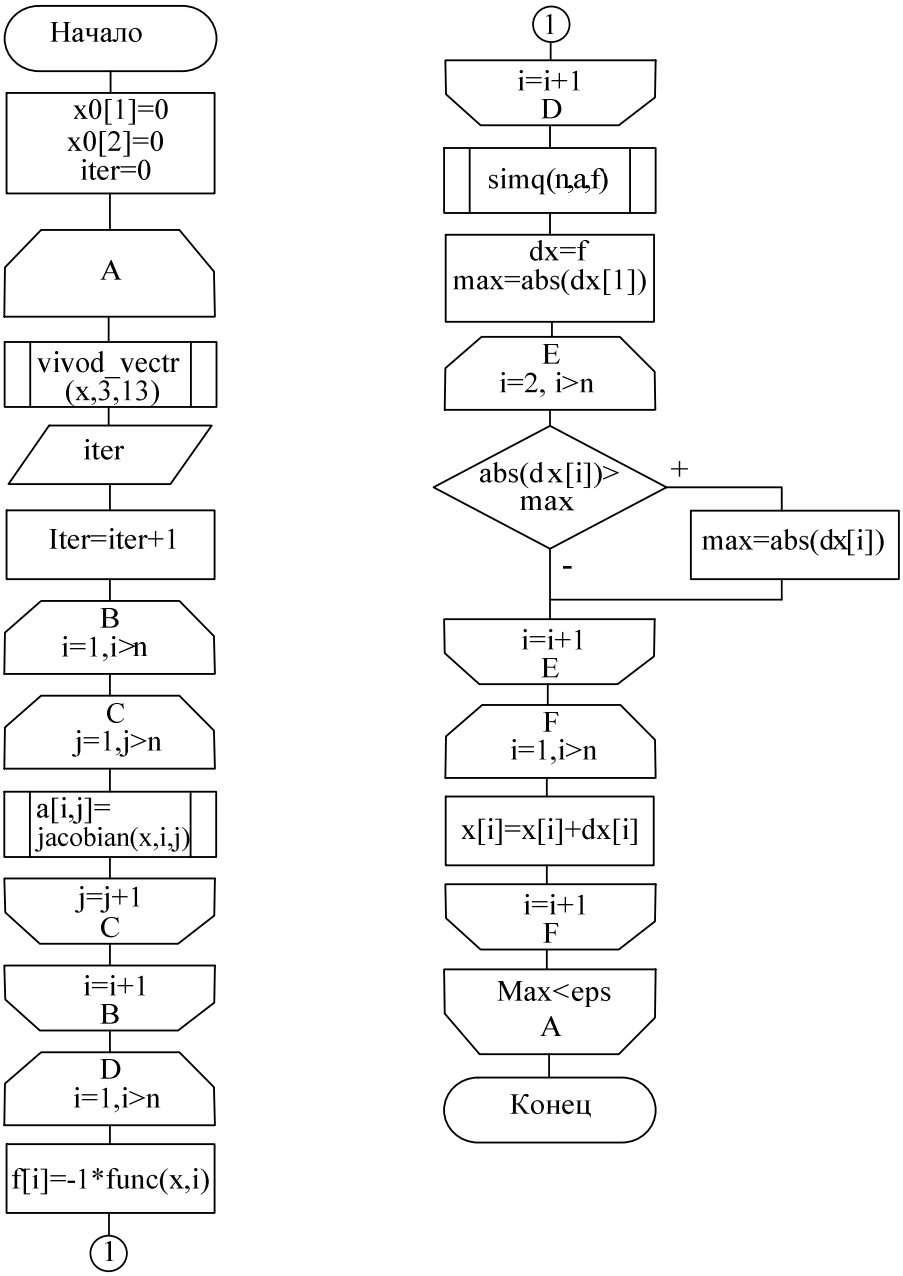


Рисунок 21 – Схема алгоритма метода Ньютона

Варианты заданий

Таблица 4

1	$\begin{cases}\sin(x+1)-y=1{,}2\\ 2x+\cos y=2.\end{cases}$	16	$\begin{cases}\sin(x+0{,}5)-y=1\\ \cos(y-2)+x=0\end{cases}$
2	$\begin{cases}\sin x+2y=2\\ \cos(y-1)+x=0{,}7\end{cases}$	17	$\begin{cases}\sin x\,(x_1)=1{,}3-y\\ x-\sin(y+1)=0{,}8\end{cases}$
3	$\begin{cases}\cos(x+0{,}5)-y=2\\ \sin y2x=1\end{cases}$	18	$\begin{cases}\sin(y+1)-x=1{,}2\\ 2y+\cos x=2.\end{cases}$
4	$\begin{cases}\sin y+2x=2\\ \cos(x-1)+y=0{,}7\end{cases}$	19	$\begin{cases}\sin(y+0{,}5)-x=1\\ \cos(x-2)+y=0\end{cases}$
5	$\begin{cases}\sin(y-1)+x=1{,}3\\ y-\sin(x+1)=0{,}8\end{cases}$	20	$\begin{cases}\cos(y+0{,}5)-x=2\\ \sin x-2y=1\end{cases}$
6	$\begin{cases}\sin(x+1)-y=1\\ 2x+\cos y=2\end{cases}$	21	$\begin{cases}\sin x+2y=1{,}6\\ \cos(y-1)+x=1\end{cases}$
7	$\begin{cases}\sin(x+0{,}5)-y=1{,}2\\ \cos(y-2)+x=0\end{cases}$	22	$\begin{cases}\cos(x-1)+y=0{,}5\\ x-\cos y=3\end{cases}$
8	$\begin{cases}\cos x+y=1{,}5\\ 2x-\sin(y-0{,}5)=1\end{cases}$	23	$\begin{cases}\cos(x+0{,}5)+y=0{,}8\\ \sin y-2x=1{,}6\end{cases}$
9	$\begin{cases}2y-\cos(x+1)=0\\ x+\sin y=-0{,}4\end{cases}$	24	$\begin{cases}\sin(x+2)-y=1{,}5\\ x+\cos(y-2)=0{,}5\end{cases}$
10	$\begin{cases}\cos(y-1)+x=0{,}5\\ y-\cos x=3\end{cases}$	25	$\begin{cases}\cos y+x=1{,}5\\ 2y-\sin(x-0{,}5)=1\end{cases}$
11	$\begin{cases}\cos(y+0{,}5)+x=0{,}8\\ \sin x-2y=1{,}6\end{cases}$	26	$\begin{cases}2x-\cos(y+1)=0\\ y+\sin x=-0{,}4\end{cases}$
12	$\begin{cases}\sin(y+2)-x=1{,}5\\ y+\cos(x-2)=0{,}5\end{cases}$	27	$\begin{cases}\cos(x-1)+y=0{,}8\\ x-\cos y=2\end{cases}$
13	$\begin{cases}\cos x+y=1{,}2\\ 2x-\sin(y-0{,}5)=2\end{cases}$	28	$\begin{cases}\cos(x+0{,}5)+y=1\\ \sin y-2x=2\end{cases}$
14	$\begin{cases}\sin(x-1)+y=1{,}5\\ x-\sin(y+1)=1\end{cases}$	29	$\begin{cases}\sin(y+1)-x=1\\ 2y+\cos x=2\end{cases}$
15	$\begin{cases}\cos(y-1)+x=0{,}8\\ y-\cos x=2\end{cases}$	30	$\begin{cases}\cos(x-1)+y=1\\ \sin y+2x=1{,}6\end{cases}$

Лабораторная работа № 5

Решение проблемы собственных значений и собственных векторов

Точные методы

Метод Леверрье

Procedure Leverre(*Matr:TMatr*);

(в процедуре используются следующие процедуры и функции: *Copy* – копирование матрицы; *MatrUmn* – перемножение матриц; *Resh* – нахождение решения характеристического многочлена методом Ньютона; *SobVect* – нахождение собственных векторов матрицы).

Входные параметры: *Matr: TMatr* – исходная матрица (формируется перед запуском).

Выходные параметры: вычисляются и выводятся на экран собственные значения и вектора.

Схема алгоритма приведена на рисунке 22.

Пример: Вычислить собственные значения и собственные вектора матрицы

$$\begin{pmatrix} 1 & -1 & -1 & 2 \\ 2 & 3 & 0 & -4 \\ 1 & 1 & -2 & -2 \\ 1 & 1 & 0 & -1 \end{pmatrix}$$

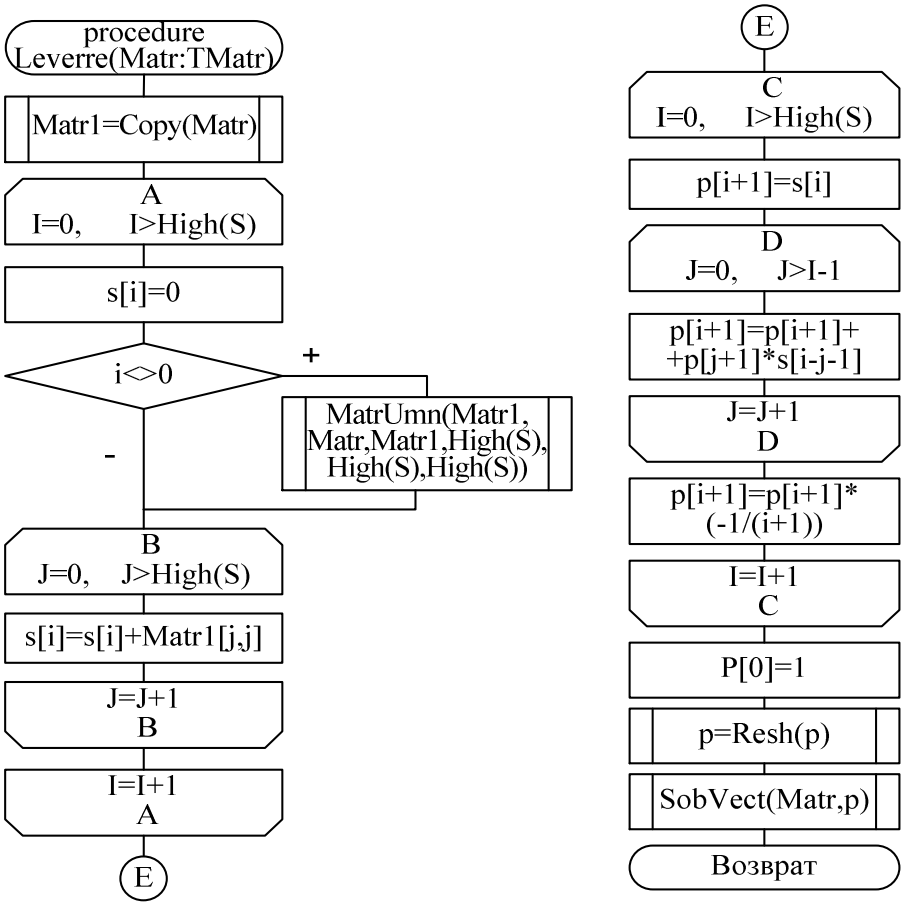
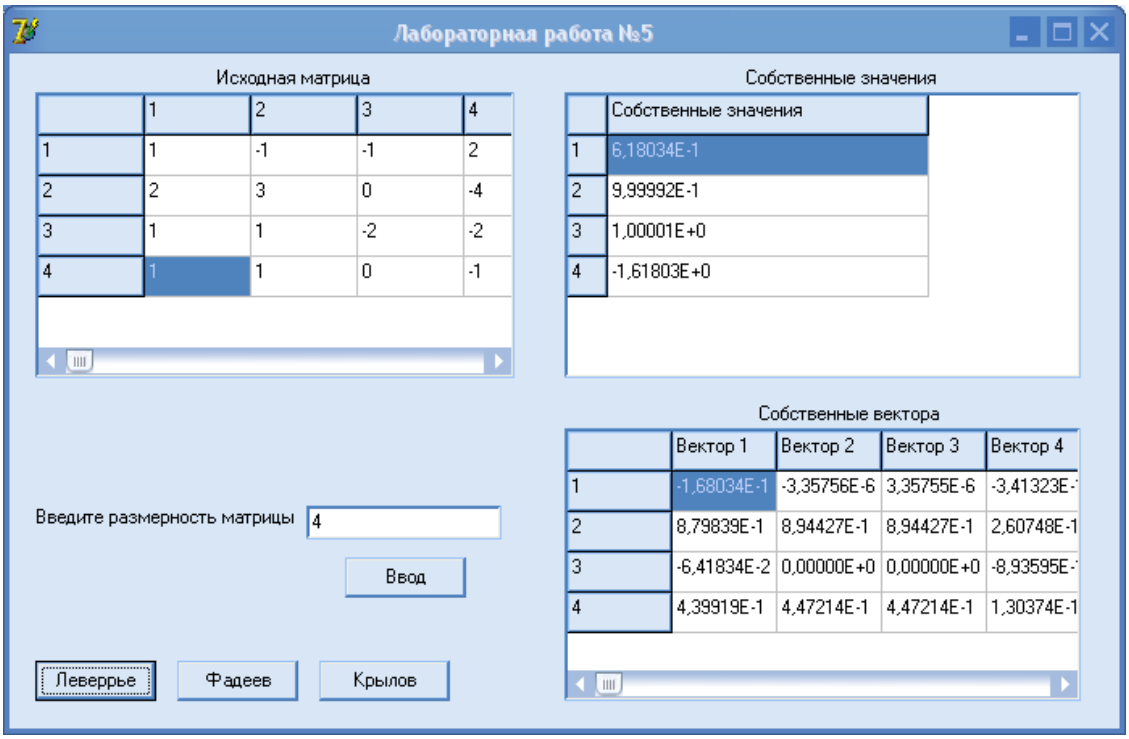


Рисунок 22 – Схема алгоритма метода Леверрье

Вычисления по программе привели к следующим результатам



Варианты заданий для нахождения собственных значений и собственных векторов матрицы приведены в таблице 5.

Метод Фадеева

Procedure Fadeev(Matr:TMatr);

(в процедуре используются следующие процедуры и функции: *Copy*1 – копирование матрицы; *MatrUmn* – перемножение матриц; *Resh* – нахождение решения характеристического многочлена методом Ньютона; *SobVect* – нахождение собственных векторов матрицы).

Входные параметры: *Matr: TMatr* – исходная матрица (формируется перед запуском).

Выходные параметры: находятся и выводятся на экран собственные значения и вектора.

Схема алгоритма приведена на рисунке 23.

Пример. Вычислить собственные значения и собственные вектора матрицы

$$\begin{pmatrix} 1 & -1 & -1 & 2 \\ 2 & 3 & 0 & -4 \\ 1 & 1 & -2 & -2 \\ 1 & 1 & 0 & -1 \end{pmatrix}$$

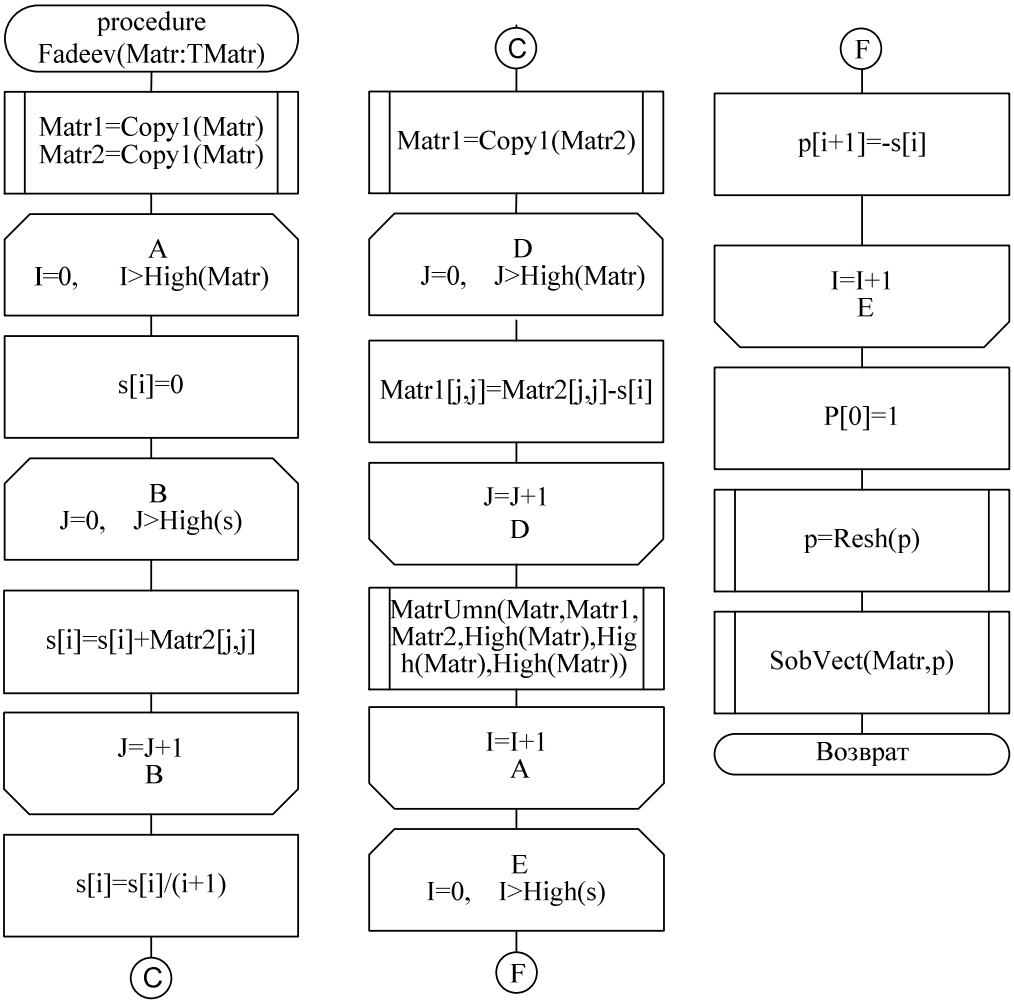
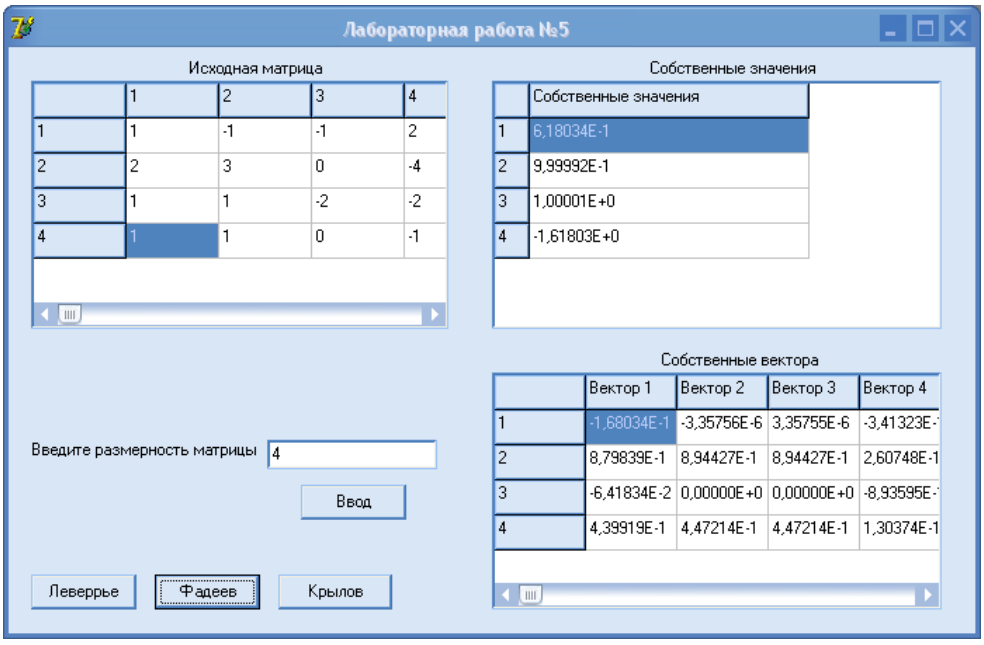


Рисунок 23 – Схема алгоритма метода Фадеева

Вычисления по программе привели к следующим результатам



Варианты заданий для нахождения собственных значений и собственных векторов матрицы приведены в таблице 5.

Метод Крылова

Procedure Krilov(Matr:TMatr);

(в процедуре используются следующие процедуры и функции: *Copy* – копирование матрицы; *MatrUmnMas* – умножение матрицы на массив; *CopyMas* – копирование массива; *CopyTrans* – копирование и транспонирование матрицы; *Det* – нахождение определителя матрицы; *SIMQ* – решение системы линийных алгебраических уравнений. (Процедура берется из первой лабораторной). *Resh* – нахождение решения характеристического многочлена методом Ньютона; *SobVect* – нахождение собственных векторов матрицы).

Входные параметры: *Matr: TMatr* – исходная матрица (формируется перед запуском).

Выходные параметры: находятся и выводятся на экран собственные значения и вектора.

Схема алгоритма приведена на рисунке 24.

Пример. Вычислить собственные значения и собственные вектора матрицы

$$\begin{pmatrix} 1 & -1 & -1 & 2 \\ 2 & 3 & 0 & -4 \\ 1 & 1 & -2 & -2 \\ 1 & 1 & 0 & -1 \end{pmatrix}$$

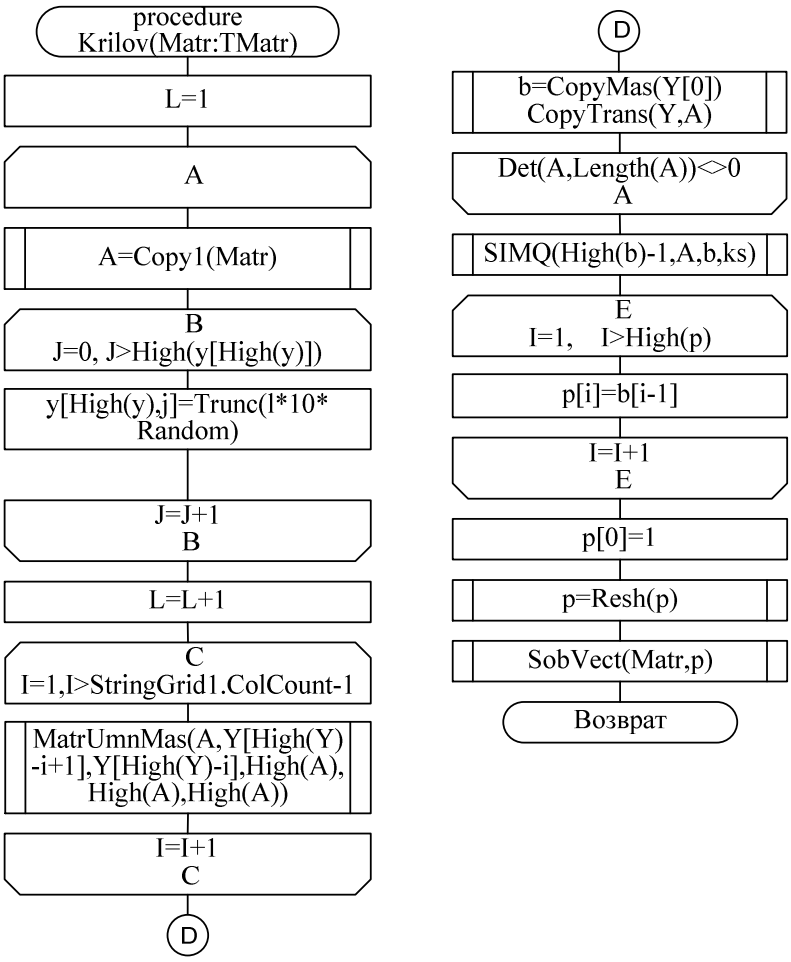
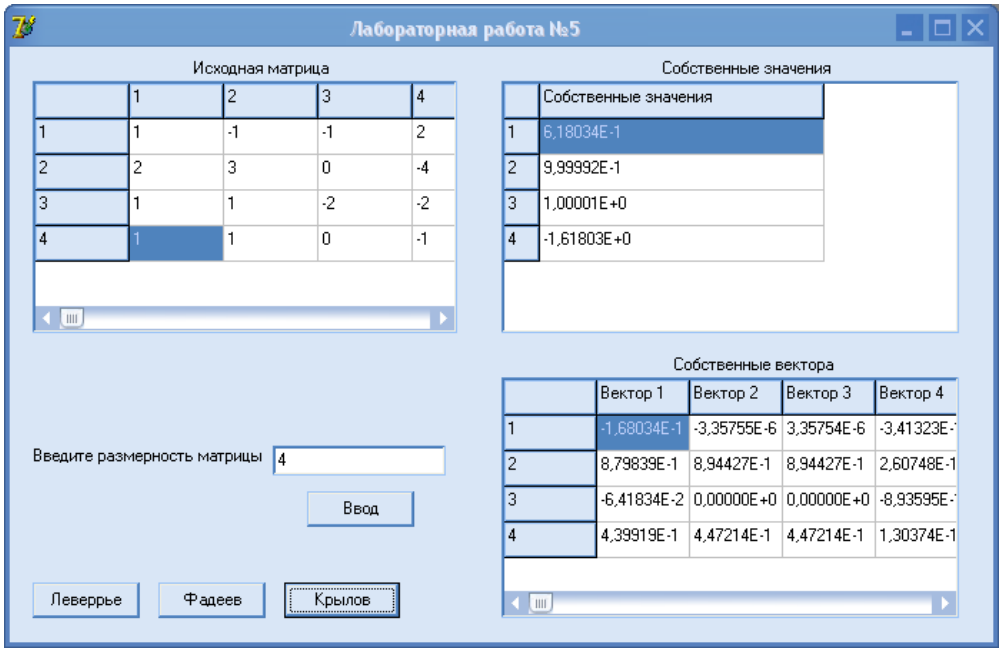


Рисунок 24 – Схема алгоритма метода Крылова

Вычисления по программе привели к следующим результатам



Варианты заданий для нахождения собственных значений и собственных векторов матрицы приведены в таблице 5.

Текст программы вычисления собственных значений и векторов методами: Леверрье, Фадеева, Крылова:

```
unit Unit1;// программа вычисления собственных значений и векторов методами: Леверрье, Фадеева, Крылова.

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    Button1: TButton;
    Edit1: TEdit;
    StringGrid2: TStringGrid;
    Button2: TButton;
    StringGrid3: TStringGrid;
    Button6: TButton;
    Label1: TLabel;
    Button3: TButton;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
type
 TMas=array of real;
 TMatr=array of TMas;
 TPer=array of Integer;
```

```
function Vkl(Per:TPer;Znach,Kol:integer):Boolean; //проверяет входит ли в массив значение Znach, используется при вычислении определителя
var
 i:integer;
begin
 result:=false;
 for i:=0 to kol-1 do
  if Per[i]=Znach then
   begin
    result:=true; exit;
   end;
end;

function Perestanovka(Per:TPer;n:Integer):boolean;//для определителя указывает знак с каким входит в сумму очередное слагаемое
var
 i,j,kol:integer;
begin
 kol:=0;
 for i:=0 to n-2 do
  for j:=i+1 to n-1 do
   if Per[i]>Per[j] then
    inc(kol);
 result:= ODD(kol);
end;

function SumMatrToPer(Matr:TMatr;Per:TPer;n:Integer):Extended;// формирует очередное слагаемое в определителе
var
 i:integer;
begin
 result:=1;
 for i:=0 to n-1 do
   result:=result*Matr[i,Per[i]];
 if  Perestanovka(Per,n) then
  result:=-1*result;
end;

function DetRec(Matr:TMatr;const n:integer;Per:TPER;n0:integer):Extended;
//рекурсивно формирует перестановки и ищет определитель
var
 i:integer;
begin
 result:=0;
 for i:=0 to n-1 do
  begin
   if Vkl(Per,i,n0) then
    continue
   else
    begin
     Per[n0]:=i;
```

```
      if n0=n-1 then
       begin
        result:=SumMatrToPer(Matr,Per,n);
       end
      else
       result:=result+DetRec(Matr,n,Per,n0+1);
     end;
   end;
end;

function Det(Matr:TMatr;n:Integer):Extended;// подготавливает массив и запускает рекурсию для нахождения определителя
var
 Per:TPer;
begin
 SetLength(Per,n);
 Per[0]:=1;
 result:=DetRec(Matr,n,Per,0);
end;

Function  SQRN(a:real;b:integer):double;//возводит в  степень B число A
begin
 if a>0 then
   sqrn:=exp(b*ln(a))
  else
   if a<>0 then
    begin
     if odd(b)=true then
      begin
       sqrn:=-exp(b*ln(abs(a)));
      end
     else
      sqrn:=exp(b*ln(abs(a)));
    end
  else
   sqrn:=0;
end;

procedure MatrUmn(a,b:TMatr;var c:TMatr;n,m,k:integer);//перемножение матриц
var
 i,j,l:integer;
 s:real;
begin
 SetLength(c,n+1,k+1);
  for i:=0 to n do
   begin
    for j:=0 to k do
     begin
      s:=0;
       for l:=0 to m do
        s:=s+a[i,l]*b[l,j];
       c[i,j]:=s;
```

```
      end;
    end;
  end;

  function Proizv(xar:TMas):TMas;// считает производную многочлена, переданного в
массиве
  var
   i:integer;
   proizv:TMas;
  begin
   SetLength(proizv,High(xar));
   for i:=0 to High(xar)-2 do
    proizv[i]:=xar[i]*(High(xar)-i);
   proizv[High(xar)-1]:=xar[High(xar)-1];
   result:=proizv;
  end;

  Function Delenie(f:TMas;koren:real):TMas;// делит многочлен на одночлен (корень),
тем самым уменьшая его степень
  var
   i:integer;
   otv:TMas;
  begin
   SetLength(otv,High(f)+1);
   otv[0]:=f[0];
   for i:=1 to High(f) do
    otv[i]:=(koren*otv[i-1])+f[i];
   result:=otv;
  end;

  function Podstanovka(xar:TMas;kor:real):real;//подставляет число в многочлен
  var
   i:integer;
   otv:real;
  begin
   otv:=0;
   for i:=0 to High(xar)-1 do
    begin
     otv:=otv+SQRN(kor,High(xar)-i)*xar[i];
    end;
   otv:=otv+xar[High(xar)];
   result:=otv;
  end;

  function Resh(xar:TMas):TMas;//находит решение многочлена
  var
   dx,xn,xn1:real;
   i,p:integer;
   f1,otv:TMas;
  begin
   SetLength(otv,High(xar));
   p:=High(xar);
```

```
 for i:=1 to p do
  begin
   xn:=0.00001;
   f1:=(Proizv(xar));
   repeat
     dx:=-(Podstanovka(xar,xn))/(Podstanovka(f1,xn));
     xn1:=dx+xn;
     xn:=xn1;
   until (abs(dx)<0.00001)or(Podstanovka(xar,xn1)=0);
   xar:=Delenie(xar,xn1);
   SetLength(xar,High(Xar));
   otv[i-1]:=xn1;
   Form1.StringGrid2.Cells[1,i]:=FloatToStrF(xn1,ffExponent,6,13);
  end;
 result:=otv;
end;

procedure TForm1.Button2Click(Sender: TObject);//подготавливает к вводу StringGrid-
ы
 var
  i:integer;
 begin
  StringGrid1.ColCount:=StrToInt(Edit1.Text)+1;
  StringGrid1.RowCount:=StrToInt(Edit1.Text)+1;
  StringGrid2.RowCount:=StrToInt(Edit1.Text)+1;
  StringGrid3.RowCount:=StrToInt(Edit1.Text)+1;
  StringGrid3.ColCount:=StrToInt(Edit1.Text)+1;
  StringGrid2.Cells[1,0]:='Собственные значения';
  for i:=1 to StrToInt(Edit1.Text) do
   begin
    StringGrid1.Cells[0,i]:=IntToStr(i);
    StringGrid3.Cells[0,i]:=IntToStr(i);
    StringGrid2.Cells[0,i]:=IntToStr(i);
    StringGrid1.Cells[i,0]:=IntToStr(i);
    StringGrid3.Cells[i,0]:='Вектор '+IntToStr(i);
   end;
 end;

 Function sum(Matr:TMatr;Mas:TMas;p:integer):real; //находит значение очередного не-
известного, считая сумму последующих элементов и деля её на элемент на главной диаго-
нали
 var
  i:integer;
  sun:real;
 begin
  sun:=0;
  for i:=p+1 to High(Matr) do
   sun:=sun+Matr[p,i]*Mas[i];
  sun:=-sun/Matr[p,p];
  result:=sun
 end;
```

```
function Perest(Var Matr:TMatr;p,i:integer):boolean;
var
 u,l:integer;
 rec:real;
begin
  result:=false;
  for u:=p+1 to High(Matr) do
   if Matr[u,i]<>0 then
    begin
    for l:=0 to High(Matr) do
     begin
      rec:=Matr[p,l];
      Matr[p,l]:=Matr[u,l];
      Matr[u,l]:=rec;
     end;
     result:=true;
     break;
     end;
end;

procedure Minim(var Matr:TMatr);//заменят все элементы в матрице меньше 0.0001 на 0
 var
  i,j:integer;
 begin
  for i:=0 to High(Matr) do
   for j:=0 to High(Matr) do
    if Abs(Matr[i,j])<0.0001 then
     Matr[i,j]:=0
end;

procedure Prov(Matr:TMatr;var b1:TMas;k,l:integer);// делается проверка, если решение до этого было выбрано любое, а теперь выясняется что оно не подходит, то оно заменяется 0
 var
  i:integer;
 begin
  for i:=l+1 to High(Matr) do
   if Matr[k,i]<>0 then
    b1[i]:=0
 end;

function Stup(var Matr:TMatr):TMas;//приводим матрицу к ступенчатому виду и находим любое частное решение
 var
  k,i,j:integer;
  b:real;
  b1:TMas;
 begin
  for i:=0 to High(Matr)-1 do
   begin
    for k:=i+1 to High(matr) do
     begin
```

```
     if abs(Matr[i,i])=0 then //break;
      if Perest(Matr,i,i)<>true then break;
     b:=-matr[k,i]/Matr[i,i];
     for j:=0 to High(Matr) do
      begin
       Matr[k,j]:=Matr[i,j]*b+Matr[k,j];
      end;
     Minim(Matr);
    end;
  end;
 SetLength(b1,High(Matr)+1);
 for i:=High(Matr) downto 0 do
  if abs(Matr[i,i])=0 then
   begin
    b1[i]:=1;
    Prov(Matr,b1,i,i);
   end
  else
   b1[i]:=sum(Matr,b1,i);
 result:=b1
end;

Function Copy1(Matr:TMatr):TMatr;//копируем матрицу
var
 i,j:integer;
 Matr1:TMatr;
begin
 SetLength(Matr1,High(Matr)+1,High(Matr)+1);
 for i:=0 to High(Matr) do
  for j:=0 to High(Matr) do
   Matr1[i,j]:=Matr[i,j];
 result:=Matr1;
end;

function CopyMas(const Mas:TMas):TMas;//копируем массив
var
 i:integer;
begin
 SetLength(result,Length(Mas)+1);
 for i:=0 to High(Mas) do
   result[i]:=Mas[i];
end;

function Clear1(Mas:TMas):TMas;//очищаем массив
var
 i:integer;
 Mas1:TMas;
begin
 SetLength(Mas1,High(Mas)+1);
 for i:=0 to High(Mas) do
  Mas1[i]:=0;
 result:=Mas1;
```

```
end;

procedure OutPut(otv1:TMas;p:integer);//нормализуем массив
var
 i:integer;
 s:real;
begin
 s:=0;
 for i:=0 to High(otv1) do
  s:=s+SQR(otv1[i]);
 s:=SQRT(s);
 for i:=0 to High(otv1) do
  begin
   otv1[i]:=otv1[i]/s;
   Form1.StringGrid3.Cells[p,i+1]:=FloatToStrF(otv1[i],ffExponent,6,13);
  end;
end;

procedure SobVect(Matr:TMatr;otv:TMas); //находим собственные вектора для собственных значений
var
 i,k:integer;
 Matr1:TMatr;
 otv1:TMas;
begin
 SetLength(otv1,High(Matr)+1);
 for k:=0 to High(otv)do
  begin
   Matr1:=Copy1(Matr);
   for i:=0 to High(otv) do
    begin
     Matr1[i,i]:=Matr[i,i]-otv[k];
    end;
   Minim(Matr1);
   otv1:=Clear1(otv1);
   otv1:=Stup(Matr1);
   OutPut(otv1,k+1);
  end;
end;

procedure Leverre(Matr:TMatr);
var
 i,j:integer;
  Matr1:TMatr;
     S,P:TMas;
begin
 SetLength(S,Form1.StringGrid1.ColCount-1);
 SetLength(P,Form1.StringGrid1.ColCount);
 Matr1:=Copy(Matr);
 For i:=0 to High(S) do
  begin
   s[i]:=0;
```

```
          if i<>0 then
           MatrUmn(Matr1,Matr,Matr1,High(S),High(S),High(S));
          for j:=0 to High(s) do
           s[i]:=s[i]+Matr1[j,j];
         end;
        for i:=0 to High(s) do
         begin
          p[i+1]:=s[i];
          for j:=0 to i-1 do
           begin
            p[i+1]:=p[i+1]+p[j+1]*s[i-j-1];
           end;
          p[i+1]:=p[i+1]*(-1/(i+1));
         end;
        p[0]:=1;
        p:=Resh(p);
        SobVect(Matr,p);
       end;

       procedure TForm1.Button1Click(Sender: TObject);//находим коэффициенты характери-
стического многочлена методом Леверрье
       var
        Matr:TMatr;
        i,j:integer;
       begin
        SetLength(Matr,StringGrid1.ColCount-1,StringGrid1.ColCount-1);
        for i:=1 to StringGrid1.ColCount-1 do
         for j:=1 to StringGrid1.RowCount-1 do
          begin
           Matr[i-1,j-1]:=StrToFloat(StringGrid1.Cells[j,i]);
          end;
        Leverre(Matr);
       end;

       procedure Fadeev(Matr:TMatr);
       var
        i,j:integer;
        Matr1,Matr2:TMatr;
        S,P:TMas;
       begin
        SetLength(S,Form1.StringGrid1.ColCount-1);
        SetLength(P,Form1.StringGrid1.ColCount);
        Matr1:=Copy1(Matr);
        Matr2:=Copy1(Matr);
        For i:=0 to High(Matr) do
         begin
          s[i]:=0;
          for j:=0 to High(s) do
           s[i]:=s[i]+Matr2[j,j];
          s[i]:=s[i]/(i+1);
          Matr1:=Copy1(Matr2);
          for j:=0 to High(Matr) do
```

```
        Matr1[j,j]:=Matr2[j,j]-s[i];
       MatrUmn(Matr,Matr1,Matr2,High(Matr),High(Matr),High(Matr));
      end;
     for i:=0 to High(s) do
      p[i+1]:=-s[i];
     p[0]:=1;
     p:=Resh(p);
     SobVect(Matr,p);
    end;

    procedure TForm1.Button6Click(Sender: TObject);//находим коэффициенты характери-
стического многочлена методом Фадеева
    var
     i,j:integer;
      Matr:TMatr;
    begin
     SetLength(Matr,StringGrid1.ColCount-1,StringGrid1.ColCount-1);
     for i:=1 to StringGrid1.ColCount-1 do
      for j:=1 to StringGrid1.RowCount-1 do
       begin
        Matr[i-1,j-1]:=StrToFloat(StringGrid1.Cells[j,i]);
       end;
     Fadeev(Matr);
    end;

    procedure MatrUmnMas(a:tmatr;b:tmas;Var c:tmas;n,m,k:byte);// находим произведение
матрицы на массив
    var
    s:real;
    i,j,l:byte;
    Begin
     for i:=0 to n do
       begin
        for j:=0 to k do
         begin
          s:=0;
           for l:=0 to m do
            s:=s+a[i,l]*b[l];
           c[i]:=s;
         end;
       end;
    end;

    procedure CopyTrans(A:TMatr;var OutPut:TMatr);//копируем и транспонируем матри-
цу, удаляя последнюю строчку
    var
     i,j:integer;
    begin
     for i:=0 to High(A)-1 do
      for j:=0 to High(a)-1 do
       OutPut[i,j]:=a[j+1,i];
    end;
```

```
procedure SIMQ(n:integer;  a:tmatr; var b:tmas;var ks:Integer);
label m1;
const eps=1e-21;
var
 max,u,v:real;
 i,j,k1,l:integer;
begin
 for i:=0 to n do a[i,n+1]:=-b[i];
 for i:=0 to n do
  begin
   max:=abs(a[i,i]); k1:=i;
   for l:=i+1 to n do if (abs(a[l,i])>max) then
    begin
     max:=abs(a[l,i]); k1:=l;
    end;
   if(max<eps)then begin ks:=1; goto m1;
   end  else ks:=0;
   if k1<>i then
   for j:=i to n+1 do
   begin u:=a[i,j]; a[i,j]:=a[k1,j]; a[k1,j]:=u; end;
   v:=a[i,i];
   for j:=i to n+1 do a[i,j]:=a[i,j]/v;
   for l:=i+1 to n do begin
   v:=a[l,i]; for j:=i+1 to n+1 do a[l,j]:=a[l,j]-a[i,j]*v;
   end;  end;
   b[n]:=a[n,n+1];
   for i:=n-1 downto 0 do begin b[i]:=a[i,n+1];
   for j:=i+1 to n do b[i]:=b[i]-a[i,j]*b[j];
   end;
  m1:end;

procedure Krilov(Matr:TMatr);
var
 A,Y:TMatr;
 b,p:TMas;
 ks,i,j,l:integer;
begin
 SetLength(p,Form1.StringGrid1.ColCount);
 SetLength(A,form1.StringGrid1.ColCount-1,Form1.StringGrid1.ColCount-1);
 SetLength(Y,Form1.StringGrid1.ColCount,Form1.StringGrid1.ColCount-1);
 SetLength(b,Form1.StringGrid1.ColCount-1);
 l:=1;
 repeat
  A:=Copy1(Matr);
 for j:=0 to High(y[High(y)]) do
  y[High(y),j]:=Trunc(l*10*Random);
  inc(l);
  for i:=1 to Form1.StringGrid1.ColCount-1 do
   MatrUmnMas(A,Y[High(Y)-i+1],Y[High(Y)-i],High(A),High(A),High(A));
  b:=CopyMas(Y[0]);
  CopyTrans(Y,A);
```

```
 until Det(A,Length(A))<>0 ;
 SetLength(A,Form1.StringGrid1.ColCount-1,Form1.StringGrid1.ColCount);
 SIMQ(High(b)-1,A,b,ks);
 for i:=1 to High(p) do
  p[i]:=b[i-1];
 p[0]:=1;
 p:=Resh(p);
 SobVect(matr,p);
end;

procedure TForm1.Button3Click(Sender: TObject);//находим коэффициенты характери-
стического многочлена методом Крылова
var
 Matr:TMatr;
 i,j:integer;
begin
 SetLength(Matr,StringGrid1.ColCount-1,StringGrid1.ColCount-1);
 for i:=1 to StringGrid1.ColCount-1 do
  for j:=1 to StringGrid1.RowCount-1 do
   begin
    Matr[i-1,j-1]:=StrToFloat(StringGrid1.Cells[j,i]);
   end;
 Krilov(Matr);
end;

end.
```

Лабораторная работа № 6

Решение проблемы собственных значений и собственных векторов

Итерационные методы

Метод QR-разложения

Procedure Qr (*Var Nn: Integer; Var A: Tmatr;*
Var R_1: Tvector; Var R_2: Tvector);

Входные параметры: *nn* – размерность системы; *a* – матрица размерности *nn* * *nn*, содержащит коэффициенты системы.

Выходные параметры: *R*_1 – массив из *nn* действительных чисел, при выходе из программы содержит собственные числа матрицы; *a* – матрица *nn** *nn* содержит собственные вектора матрицы.

Пример. Вычислить собственные значения и собственные вектора матрицы

$$\begin{pmatrix} 2 & 2 & -2 \\ 2 & 5 & -4 \\ -2 & -4 & 5 \end{pmatrix}$$

Текст процедуры:

```
PROCEDURE Qr (Var Nn:Integer;Var A:Tmatr;Var R_1:Tvector;Var R_2:Tvector);;
  Var
  I,J,K,L,M,Na,Its  : Integer;
  I1,M1,N1,L1,N     : Integer;
  M3,U1,U11,U2      : Real;
  Q,P,R,S,T,U,X,Y,Z,W : Real;
  Aa: tMatr;
  r1,r2:tvector;
  Label NexTw,NexTit,Cont_1,Cont_2,Cont_3;
  Label Onew,TwOw,Fin;
  Begin
  Aa:=A;   N:=Nn;
   for i:=0 to Nn do
   begin    r1[i]:=0;     r2[i]:=0; end;
   T:=0.0;
   NexTw:
   If (N=0) Then Goto Fin;
   Its:=0;        Na:=N-1;      M3:=1.0e-11;
   NexTit:
```

```
  For L:=N DownTo 2 Do Begin
   U:=Abs(Aa[l,l-1]); U1:=M3*(Abs(Aa[l-1,l-1])+Abs(Aa[l,l]));
   If (U<=U1) Then Goto Cont_1;
    End;
  L:=1;   Cont_1:
       X:=Aa[n,n];       If(L=N) Then Goto Onew;
       Y:=Aa[na,na];          W:=Aa[n,na]*Aa[na,n];
       If(L=Na) Then Goto Twow;          If(Its=30) Then Goto Fin;
       If((Its=10)And(Its=20)) Then
Begin
         T:=T+X;
         For I:=1 To N Do Begin
   Aa[i,i]:=Aa[i,i]-X;      End;{}
   S:=Abs(Aa[n,na])+Abs(Aa[na,n-2]);
   X:=0.75*S;      Y:=0.75*S; W:=-0.4375*S*S;
                          End;
  Its:=Its+1;
  For M:=N-2 Downto L Do Begin
                    Z:=Aa[m,m];                          R:=X-Z;
                    S:=Y-Z;         P:=(R*S-W)/Aa[m+1,m]+Aa[m,m+1];
                    Q:=Aa[m+1,m+1]-Z-R-S;     R:=Aa[m+2,m+1];
                    S:=Abs(P)+Abs(Q)+Abs(R);
                    P:=P/S;       Q:=Q/S;      R:=R/S;
                    If(M=L) Then Goto Cont_2;
                    U11:=Abs(Aa[m,m-1])*(Abs(Q)+Abs(R));
               U2:=M3*Abs(P)*(Abs(Aa[m-1,m-1])+Abs(Z)+Abs(Aa[m+1,m+1]));
                    If(U11<=U2) Then Goto Cont_2;
                   End;
  Cont_2:          For I:=M+2 To N Do Begin   Aa[i,i-2]:=0.0;        End;
  For I:=M+3 To N Do Begin       Aa[i,i-3]:=0.0;     End;
  For K:=M To Na Do Begin
  If (K<>M) Then Begin;
  P:=Aa[k,k-1];     Q:=Aa[k+1,k-1];    If(K<>Na) Then R:=Aa[k+2,k-1]
   Else     R:=0.0;
               X:=Abs(P)+Abs(Q)+Abs(R);
               If(X=0.0) Then Goto Cont_3;
                P:=P/X;    Q:=Q/X;   R:=R/X;
  End;
               S:=Sqrt(P*P+Q*Q+R*R);
               If(P<0.0) Then S:=-S;
               If(K<>M) Then Aa[k,k-1]:=-S*X
               Else If(L<>M) Then Aa[k,k-1]:=-Aa[k,k-1];
               P:=P+S; X:=P/S;  Y:=Q/S; Z:=R/S;Q:=Q/P;  R:=R/P;
               For J:=K To N Do Begin
                          P:=Aa[k,j]+Q*Aa[k+1,j];
                          If (K<>Na) Then Begin
                          P:=P+R*Aa[k+2,j];  Aa[k+2,j]:=Aa[k+2,j]-P*Z;    End;
                          Aa[k+1,j]:=Aa[k+1,j]-P*Y;   Aa[k,j]:=Aa[k,j]-P*X;
                         End;
    If((K+3)<N) Then J:=K+3
    Else                  J:=N;
               For I:=L To J Do Begin
```

```
                              P:=X*Aa[i,k]+Y*Aa[i,k+1];   If(K<>Na) Then Begin;
                              P:=P+Z*Aa[i,k+2];     Aa[i,k+2]:=Aa[i,k+2]-P*R;
End;
                               Aa[i,k+1]:=Aa[i,k+1]-P*Q;              Aa[i,k]:=Aa[i,k]-P;
                              End;
      Cont_3:End;   Goto NexTit;   Onew:
         R1[n]:=X+T;  {R1-ўГй}   R2[n]:=0.0;    {Cnt[n]:=its;}   N:=Na;
         Goto Nextw;   Twow:
         P:=(Y-X)/2.0;    Q:=P*P+W; Y:=Sqrt(Abs(Q));
         {Cnt[n]:=-its;  Cnt[na]:=its;}   X:=X+T;
         If(Q>0.0) Then Begin
         If(P<0.0) Then Y:=-1.0*Y;   Y:=P+Y;   R1[na]:=X+Y;
         R1[n]:=X-W/Y;   R2[na]:=0.0;   R2[n]:=0.0;   End
         Else Begin R1[na]:=X+P;   R1[n]:=X+P;   R2[na]:=Y;   R2[n]:=-1.0*Y;End;
         N:=N-2;        Goto Nextw;  Fin:  r_1:=r1;  r_2:=r2;  End;
```

Вычисления по программе привели к следующим результатам:

Собственные числа		Собственные вектора	
.10000E+01	-.94281E+00	.23570E+00	-.23570E+00
.10000E+02	-.33333E+00	-.66667E+00	.66667E+00
.10000E+01	.00000E+00	-.70711E+00	-.70711E+00

Варианты заданий для нахождения собственных значений и собственных векторов матрицы приведены в таблице 5.

Метод итераций

Procedure I t(*a:tmatr;y*0*:tvec;eps:real;n:integer*);

Входные параметры:

n: integer –размерность матрицы *a* и длина вектора *y*0;

a: tmatr – исходная матрица;

*y*0: tvec – исходный вектор;

eps: real – точность вычислений.

(процедура использует 2 функции:

function umnm (*aa*1,*aa*2*:tmatr; n*5*:integer;*

var a:tmatr)*:integer;* – перемножение матриц;

function umnv(*a5:tmatr;b5:tvec;n5:integer;*

var b:tvec)*:integer;* – умножение матрицы на вектор.)

Схема алгоритма метода приведена на рисунке 25.

Пример. Вычислить собственные значения матрицы

$$\begin{pmatrix} 2 & 2 & -2 \\ 2 & 5 & -4 \\ -2 & -4 & 5 \end{pmatrix}$$

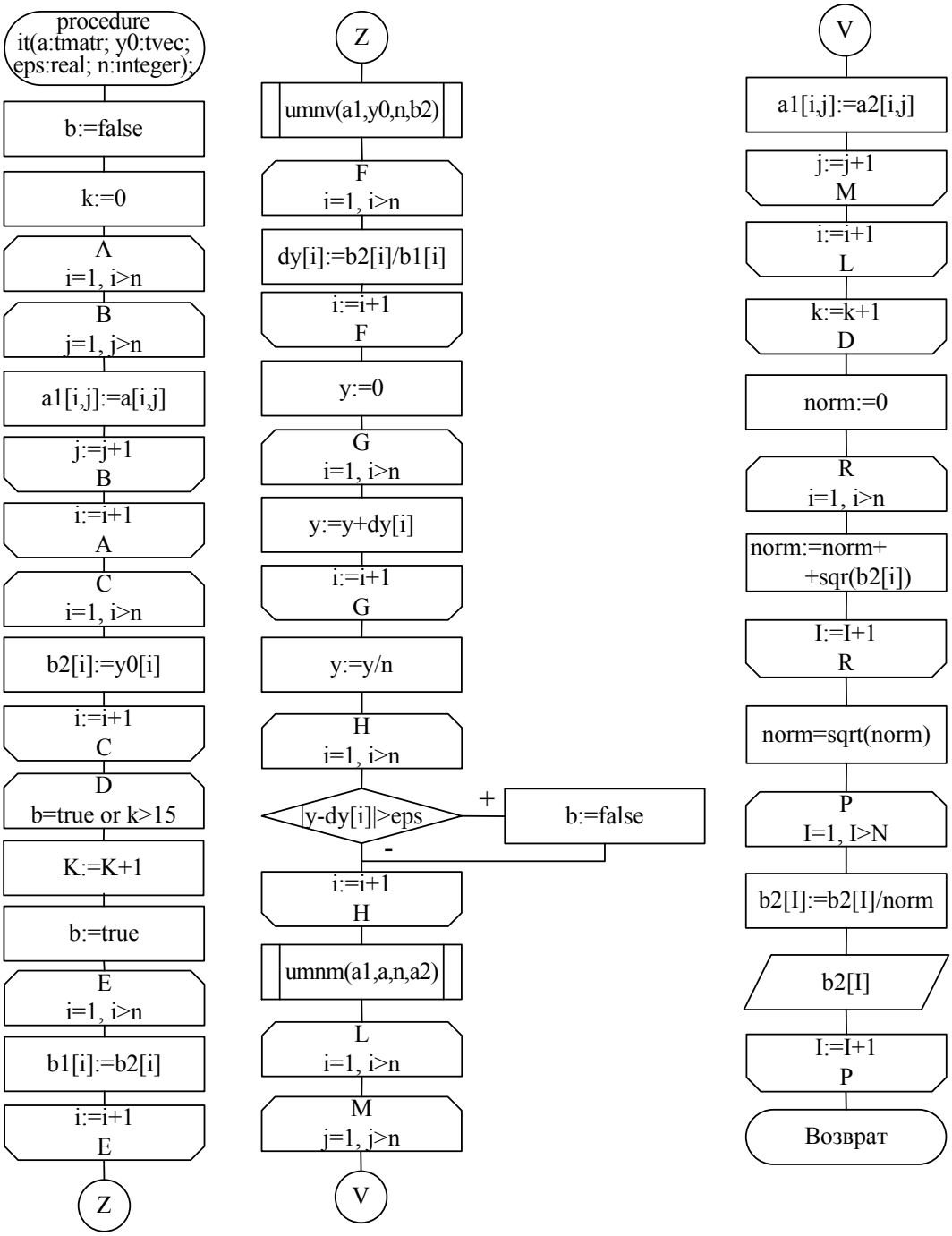


Рисунок 25 – Схема алгоритма метода итераций

Текст программы:

```
program iter;
uses crt;
type
tmatr = array[1..10,1..10] of real;
tvec = array[1..10] of real;
procedure vvod1(n:integer; var a:tmatr);
var
i5,j5:integer;
begin
writeln('Wwedite ',sqr(n),' elementow');
for i5:=1 to n do
for j5:=1 to n do
read(a[i5,j5]);
end;

procedure vvod2(n:integer; var b:tvec);
var
i5:integer;
begin
writeln('Wwedite ',n,' elementow');
for i5:=1 to n do
read(b[i5]);
end;

//перемножение матриц
function umnm(aa1,aa2:tmatr; n5:integer;var a:tmatr):integer;
var
i,j,i1,j1:integer;
r1:real;
begin
for i1:=1 to n5 do
for j1:=1 to n5 do begin
r1:=0;
for i:=1 to n5 do
r1:=r1+aa1[i1,i]*aa2[i,j1];
a[i1,j1]:=r1;
end;
end;
//умножение матрицы на вектор
function umnv(a5:tmatr;b5:tvec;n5:integer; var b:tvec):integer;
var
i,i1:integer;
r1:real;
begin
for i1:=1 to n5 do begin
r1:=0;
for i:=1 to n5 do begin
r1:=r1+a5[i1,i]*b5[i];
end;
b[i1]:=r1;
```

```
end;
end;

procedure it(a:tmatr;y0:tvec;eps:real;n:integer);
var
i,j:integer;
dy:array[1..10] of real;
b1,b2:tvec;
a1,a2:tmatr;
y,norm:real;
b:boolean;
k:integer;
begin
b:=false;
k:=0;
for i:=1 to n do
for j:=1 to n do
a1[i,j]:=a[i,j];
for i:=1 to n do b2[i]:=y0[i];
while (not b)and(k<14) do begin
k:=k+1;
b:=true;
for i:=1 to n do b1[i]:=b2[i];
//умножаем матрицу на вектор
umnv(a1,y0,n,b2);
for i:=1 to n do begin dy[i]:=b2[i]/b1[i]; end;
y:=0;
for i:=1 to n do begin writeln(i,' : ',dy[i]:4:4); end;
for i:=1 to n do begin y:=y+dy[i]; end;
y:=y/n; writeln('na shage ',k,'  chislo ',y);
for i:=1 to n do if abs(y-dy[i])>eps then b:=false;
//умножаем матрицу на матрицу
umnm(a1,a,n,a2);
for i:=1 to n do
for j:=1 to n do
a1[i,j]:=a2[i,j];
for i:=1 to n do begin
for j:=1 to n do
write(a1[i,j]:4:4,'  ');
writeln;
end;
readln;
end;
norm:=0;
//нормируем вектор собственных значений
for i:=1 to n do begin norm:=norm+sqr(b2[i]); end;
norm:=sqrt(norm);
for i:=1 to n do begin b2[i]:=b2[i]/norm; writeln(b2[i]:4:7); end;
//в данном сегменте значения выводятся на экран
readln;
end;
```

```
var
a7:tmatr;
n1:integer;
y1:tvec;
k:integer;
begin
n1:=3;
while 1=1 do begin
clrscr;
writeln('Wibirite:',#13#10,'1- wwod matrici',#13#10,'2-wwod wectora',#13#10,'3-
wichislit',#13#10,'4-wihod');
read(k);
if k=1 then vvod1(n1,a7);
if k=2 then vvod2(n1,y1);
if k=3 then it(a7,y1,0.01,n1);
if k=4 then halt;
end;
end.
```

Вычисления по программе привели к следующим результатам:

Собственные значения
.10000E+01
.10000E+02
.10000E+01

Варианты заданий

Таблица 5

Вариант	Матрица ***A***			
1	2			
1	.15446E + 00	-.43024E + 00	.55900E - 01	.24055E + 00
	.33012E + 00	-.94878E + 00	.12327E + 00	.51413E + 00
	-.54037E + 00	.15346E + 01	.19939E + 00	-.84158E + 00
	.58495E + 00	-.16812E + 01	-.21843E + 00	.91101E + 00
2	.10194E + 01	.13661E + 01	-.51685E + 01	-.22275E + 01
	.13770E + 00	.19687E + 00	-.74484E + 00	-.30089E + 00
	.14613E + 00	.19598E + 00	-.74144E + 00	-.31930E + 00
	-.12171E - 01	-.17401E - 01	.65832E - 01	.26593E - 01
3	.37712E + 01	.48587E + 00	.19836E + 00	-.53757E + 00
	-.43613E + 01	-.59594E + 00	-.24330E + 00	.62168E + 00
	.52997E + 01	.69610E + 00	.28419E + 00	-.75546E + 00
	.40591E + 01	.55464E + 00	.22644E + 00	-.57861E + 00

Продолжение таблицы 5

Вари-ант	Матрица ***A***			
1	2			
4	.41585E + 00 .13936E + 01 -.87625E + 00 -.48377E + 01	-.35891E + 00 .12212E + 01 -.73761E + 00 -.42394E + 01	-.63151E + 00 -.21488E + 01 .12978E + 01 .74592E + 01	.48148E + 00 .16136E + 01 -.10145E + 01 -.56012E + 01
5	.11107E + 01 .33082E + 00 -.17127E + 00 .29074E + 00	.10590E + 02 .30755E + 01 -.17300E + 01 .27030E + 01	.79845E + 01 .23188E + 01 -.13043E + 01 .20379E + 01	-.37547E + 01 -.11183E + 01 .57899E + 00 -.98286E + 00
6	.94217E + 01 -.82311E + 01 .10527E + 02 .28367E + 01	-.20262E + 01 .171414 + 01 .22359E + 01 -.59074E + 00	.20679E + 01 -.17494E + 01 -.22819E + 01 .60288 E + 00	-.27418E + 01 .23953E + 01 .30634E + 01 -.82550E + 00
7	.22224E + 00 .17226E + 01 .10474E + 02 .54542E + 01	.20919E + 00 .12234E + 01 .74958E + 01 .38736E + 01	.26602E + 00 .15558E + 01 .95322E + 01 .49260E + 01	.19367E + 00 .15012E + 01 .91273E + 01 .47530E + 01
8	.42016E + 00 .19439E + 00 - 61729E + 01 -.20038E + 00	-16937E + 02 -.76571E + 01 .28952E + 01 .78932E + 01	.10087E + 02 .45605E + 01 -.17253E+ 01 -.47011E + 01	-.28570E + 01 -.13218E + 01 .41974E + 00 .13625E + 01
9	.15788E + 02 -.10221E + 02 .46778E + 01 -.11261E + 02	-.31013E + 01 .20256E + 01 -.88782E + 00 .22317E + 01	-.71755E + 01 .46865E + 01 -.20541E + 01 .51635E - 01	-.71410E + 01 .46230E + 01 -.21158E + 01 .50935E + 01
10	.41750E - 01 .93292E - 01 -.19135E + 02 .89416E + 01	.72743E - 01 -.69765E - 01 .14194E + 02 -.66866E + 01	-.20820E - 01 .19967E - 01 .40626E + 01 .19138E + 01	.27069E - 01 .60487E - 01 -.12406E + 02 .57974E + 01
11	.70954E - 03 -.99360E - 04 .37446E - 03 -.35194E - 04	-.35012E + 01 .22264E + 01 .25177E + 01 -.78859E + 00	.23236E + 02 .14775E + 02 .16709E + 02 -.52335E + 01	-.16032E + 00 .22450E - 01 -.84609E - 01 .79521E - 02
12	.21121E + 02 -.97494E + 01 .54455E + 01 .92933E + 01	.15882E + 02 -.74096E + 01 .39960E + 01 .70629E + 01	-.32325E + 00 .15081E + 00 -.81334E - 01 -.14376E + 00	-.13430E + 02 .61993E + 01 -.34626E + 01 -.59092E + 01
13	.58993E + 00 -.43515E + 01 .67688E + 01 -.31085E + 02	.36004E - 01 -.24076E + 00 .39005E + 00 -.17198E + 01	-.32946E + 00 .22030E + 01 -.35692E + 01 .15737E + 02	.27315E + 00 -.20148E + 01 .31341E + 01 -.14393E + 02

Продолжение таблицы 5

Вари-ант	Матрица A			
1	2			
14	.46329E + 00 .37616E + 00 -.42388E + 00 .28498E + 00	.23325E + 02 .19986E + 02 -.21495E + 02 .15141E + 02	.56599E + 01 .48497E + 01 -.52159E + 01 .36742E + 01	.33563E + 01 .27251E + 01 -.30708E + 01 .20646E + 01
15	.23797E + 02 -.72308E + 01 -.74144E + 01 -.96619E + 00	-.96479E + 01 .28581E + 01 .30008E + 01 .38190E + 00	.25078E + 02 -.74291E + 01 -.78002E + 01 -.99269E + 00	-.20370E + 02 .61895E + 01 .63466E + 01 .82706E + 00
16	.33057E + 01 -11653E + 02 .27268E + 02 .32367E + 02	.10035E + 01 -.31905E + 01 .76123E + 01 .88617E + 01	.54364E + 00 -17284E + 01 .41118E + 01 .48006E + 01	.99381E + 00 -.35034E + 01 .81975E + 01 .97306E + 01
17	.16823E + 01 .19161E + 01 .73100E - 01 -.28993E + 01	-.12521E + 02 -14239E + 02 -.90481E + 00 .21545E + 02	.17330E + 02 .19707E + 02 .12523E + 01 -.29821E + 02	.58778E +01 .66946E + 01 .25541E + 00 -.10130E + 02
18	.23026E + 02 -.34268E + 01 .25324E + 01 -.24584E + 01	-.28865E + 02 .44373E + 01 -.30771E + 01 .31833E + 01	-.27154E + 02 .41743E + 01 -.28947E + 01 .29947E + 01	-.26187E + 02 .38973E + 01 -.28802E + 01 .27959E + 01
19	.93047E + 01 -.20747E + 02 -.50813E + 02 .23840E + 01	-.96771E + 00 .25205E + 01 .60245E + 01 -.28963E + 00	.79337E + 00 -.20664E + 01 -.49391E + 01 .23745E + 00	.14105E + 01 -.31451E + 01 -.77027E + 01 .36139E + 00
20	.29497E + 01 .46102E + 01 .63919E + 01 .53881E + 01	-.87224E + 01 -.13269E + 02 -.18646E + 02 -.15501E + 02	-.13989E + 02 -.21272E+ 02 -.29905E + 02 -.24861E + 02	.65990E + 01 .10314E + 02 .14300E + 02 .12054E + 02
21	.18899E + 02 -15438E + 00 .11694E + 00 .15299E + 00	.47944E + 02 -.43842E + 00 -.45833E - 01 .43447E + 00	-.21890E + 02 .20017E + 00 .20926E - 01 -.19837E + 00	-.28868E + 02 .23582E + 00 -.17863E + 00 -.23370E + 00
22	.18736E + 02 -.29757E + 02 .31857E + 02 -.45869E + 02	-.80167E - 01 .57914E - 01 -.12683E + 00 .89269E - 01	-.25998E + 00 .18781E + 00 -.41130E + 00 .28950E + 00	.16584E + 00 -.26340E + 00 .28199E + 00 -.40601E + 00
23	.36164E + 01 .74821E + 01 -.17309E + 02 .38860E + 01	.10248E + 02 .22156E + 02 -.50828E + 02 .11507E + 02	-.17581E + 01 -.38041E + 01 .87201E + 01 -.19742E + 01	.57433E + 01 .11883E + 02 -.27490E + 02 .61716E + 01

Продолжение таблицы 5

1	2			
24	.12738E +02	.26061E + 01	.63995E + 02	-.27195E + 02
	.16202E + 01	.35161E + 00	.86341E + 01	-.34589E + 01
	.15111E + 01	.30380E + 00	.74599E + 01	-.32261E + 01
	.81142E + 00	.17609E + 00	.43244E + 01	-.17323E + 01
25	.30846E + 02	-.40806E + 01	-.36208E + 00	-.41187E + 01
	-.36344E + 02	.49263E + 01	.43712E + 00	.48529E + 01
	.17157E + 02	-.20743E + 01	-.18406E + 00	-.22910E + 01
	.53507E + 02	-.72527E + 01	-.64354E + 00	-.71445E + 01
26	.27346E + 01	-.11110E + 01	.50222E + 01	.32234E + 01
	.91361E + 01	-.35849E + 01	.16206E + 02	. .10769E + 02
	.12961E + 02	-.51725E + 01	.23383E + 02	.15278E + 02
	-.28402E + 02	.11145E+02	-.50380E + 02	-.33479E + 02
27	.63868E + 01	-.67524E + 02	-.24412E + 02	-.20908E + 02
	.18697E + 01	-.18932E + 02	-.68445E + 01	-.61208E + 01
	-.17156E + 01	.18209E + 02	.65832E + 01	.56162E + 01
	.75164E + 00	-.76109E + 01	-.27516E + 01	-.24606E + 01
28	.43510E + 02	.59136E + 01	-.11467E + 02	-.12245E + 02
	-.38692E + 02	-.50907E + 01	.98719E + 01	.10889E + 02
	-.48615E + 02	-.65295E + 01	.12662E + 02	.13682E + 02
	-.17455E + 02	-.22966E + 01	.44536E + 01	.49125E + 01
29	.10921E + 01	-.12118E + 01	-.83855E + 00	.96881E + 00
	.74301E + 01	-.72230E + 01	-.49981E + 01	.65916E + 01
	.26781E + 01	-.07865E + 01	-.49087E + 02	.76564E + 01
	.42614E + 02	-.41427E + 02	-.28666E + 02	.37805E +02
30	.17965E + 01	.49866E + 02	-.55218E + 02	-.11507E + 02
	.78083E + 00	.22034E + 02	-.24399E + 02	-.50015E + 01
	.26239E + 00	.60831E + 01	-.67359E + 01	-.16807E + 01
	-.82565E + 00	-.23299E + 02	.25799E + 02	.52886E + 01

Лабораторная работа № 7
Приближение функций

Интерполяционный полином Лагранжа

Function Lagr(*n:integer; x,y:Vector; q:real*): *real;*

Входные параметры: n – число узлов интерполяции; x – массив размерности n, содержащий значения узлов интерполяции; y – массив размерности n, содержащий значения функции в узлах интерполяции; q – точка, в которой вычисляем значение функции.

Выходные параметры: значение функции, вычисленное с помощью интерполяционного полинома Лагранжа в точке q.

Схема алгоритма приведена на рисунке 26.

Пример. Интерполировать с помощью полинома Лагранжа табличную функцию в заданных точках.

x_i	y_i	x_i	y_i
1	2,05	6	1,88
2	1,94	7	1,71
3	1,92	8	1,60
4	1,87	9	1,56
5	1,77	10	1,40

Текст функции:

```
function Lagr(n:integer; x,y:Vector; q:real):real;
 var i,j:integer;
    l,s:real;
BEGIN
 L:=0;
 for i:=1 to n do
  begin
   s:=1;
   for j:=1 to N do
    if j<>i then s:=s*(q-x[j])/(x[i]-x[j]);
   L:=L+y[i]*s;
  end;
 Lagr:=L; END;
```

Вычисления по программе привели к следующим результатам:

x=1.00	y=2.050
x=1.50	y=2.386
x=2.00	y=1.940
x=2.50	y=1.833
x=3.00	y=1.920
x=3.50	y=1.948
x=4.00	y=1.870
x=4.50	y=1.781
x=5.00	y=1.770
x=5.50	y=1.831
x=6.00	y=1.880
x=6.50	y=1.838
x=7.00	y=1.710
x=7.50	y=1.596
x=8.00	y=1.600
x=8.50	y=1.675
x=9.00	y=1.560
x=9.50	y=1.095
x=10.00	y=1.400

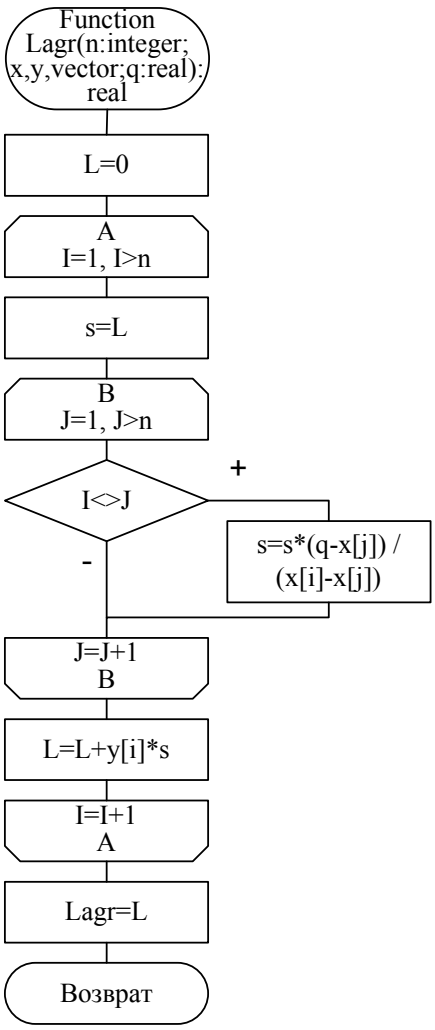


Рисунок 26 – Схема алгоритма приближения функций полиномом Лагранжа

Варианты заданий для решения задачи приближения функций приведены в таблице 6.

Интерполирование функций с помощью кубического сплайна

Procedure SPLINE(*n*1*:Byte; x*1,*y*1,*a*1,*b*1,*c*1,d1,*u*1,*v*1*:Mass;*
*var z*1*:Mass; xx*1*:Real; var s:real ;ind*1*:Byte);*

Входные параметры: *n*1 – число узлов интерполяции; *x*1 – массив размерности *n*1, содержащий значения узлов интерполяции; *y*1 – массив размерности *n*1, содержащий значения функции в узлах интерполяции; *a*1, *b*1, c1, *d*1 – рабочие массивы размерности *n*1; *u*1, *v*1 – рабочие массивы размерности *n*1+1; *xx*1 – точка, в которой вычис-

ляем значение сплайна; *ind*1 – указатель режима работы: *ind*1=0 – программа вычисляет коэффициенты сплайна и значение сплайна в точке *xx*1, *ind*1=1 – коэффициенты сплайна считаются известными и программа вычисляет только значение сплайна в точке *xx*1.

Выходные параметры: *s*1 – значение сплайна в точке *xx*1; *z*1 – массив размерности *n*1, содержащий коэффициенты сплайна.

Процедура SPLINE содержит обращение к подпрограмме PROGON решения системы линейных уравнений с трехдиагональной матрицей.

Procedure PROGON(*a*2,*b*2,*c*2,*d*2,*u*2,*v*2 *: Mass;*
*var x*2 *: Mass; n*2 *: Integer*);

Процедура *PROGON* решает линейную систему с трехдиагональной матрицей вида:

$A[i]*x[i-1]-b[i]*x[i]+c[i]*x[i+1]=d[i]$

методом прогонки.

Входные параметры: *n*2 – размерность системы; *a*2,*b*2,*c*2,*d*2 – массивы размерности *n*2, элементы расположенные на диагоналях матрицы и правые части системы; *u*2,*v*2 – рабочие массивы размерности *n*2+1.

Выходные параметры: *x*2 – массив размерности *n*2, содержащий решение системы.

Схема алгоритма приведена на рисунке 27.

Пример. Интерполировать табличную функцию с помощью кубического сплайна в заданных точках.

x_i	y_i	x_i	y_i
1	2,05	6	1,88
2	1,94	7	1,71
3	1,92	8	1,60
4	1,87	9	1,56

5	1, 77	10	1, 40

Текст программы:

```
Program VM_SPLINE;
const
    n = 10;
type
   Mass = array [1..20] of Real;
const
  x : Mass = (1,2,3,4,5,6,7,8,9,10,0,0,0,0,0,0,0,0,0,0);
  y : Mass = (2.05,1.94,1.92,1.87,1.77,1.88,1.71,1.60,1.56,1.40,0,0,0,0,0,0,0,0,0,0);
var
   a,b,c,d,u,v,z : Mass;
   ind : Byte;
   xx, s : Real;
   i : byte;
{=====================================================================}
{* Процедура PROGON решает линейную систему с трехдиагональной      *}
{* матрицей вида:                                                    *}
{* a2[i]*x2[i-1]-b2[i]*x2[i]+c2[i]*x2[i+1]=d2[i]                     *}
{* методом прогонки                                                  *}
{* n2          - размерность системы                                 *}
{* a2,b2,c2,d2 - массивы размерности n2, элементы расположенные на  *}
{*  диагоналях матрицы и правые части системы                        *}
{* u2,v2       - рабочие массивы размерности n2+1                    *}
{*                                                                   *}
{* Результат:                                                        *}
{* x2          - массив размерности n2, содержащий решение системы  *}
{=====================================================================}
Procedure PROGON(a2,b2,c2,d2,u2,v2 : Mass; var X2 : Mass; n2 : Integer);
var
   i, j, i1, im, nm, nn : byte;
   zz : Real;

begin
  a2[1] := 0;
  c2[n2] := 0;
  u2[1] := 0;
  v2[1] := 0;
  for i := 1 to n2 do
    begin
      i1 := i + 1;
      zz := 1/(b2[i]-a2[i]*v2[i]);
      v2[i1] := c2[i]*zz;
      u2[i1] := (a2[i]*u2[i] - d2[i])*zz;
    end;
  nm := n2-1;
  nn := n +1;
  x2[n2] := u2[nn];
  for j := 1 to nm do
     begin
       i := nn - j;
       im := i-1;
```

```
      x2[im] := v2[i]*x2[i]+u2[i];
    end;
end;
{=====================================================================}
{* Программа SPLINE строит кубический сплайн для интерполирования   *}
{* таблично заданной функции                                         *}
{* n1 - количество узлов интерполяции                                *}
{* x1 - массив размерности n1, содержит узлы интерполяции            *}
{* y1 - массив размерности n1, содержит значение функции в узлах     *}
{* a1,b1,c1,d1 - рабочие массивы размерности n1                      *}
{* u1,v1 - рабочие массивы размерности n1+1                          *}
{* z1 - массив размерности n1, содержит коэффициенты сплайна         *}
{* ind1 - указатель режима работы:                                   *}
{*        ind1=0 коэффициенты вычисляются                            *}
{*        ind1=1 коэффициенты считаются известными                   *}
{* xx1 - точка, в которой вычисляется функция                        *}
{*                                                                   *}
{* Содержит обращение к подпрограмме PROGON                          *}
{*                                                                   *}
{* Результат:                                                        *}
{* s1 - приближенное значение функции в точке xx1                    *}
{=====================================================================}
Procedure SPLINE(n1:Byte;x1,y1,a1,b1,c1,d1,u1,v1:Mass;var z1:Mass; xx1:Real;var s:real;ind1:Byte);
var
  i, j, nm : Integer;
  hj, hj1, am, al : Real;
  t,t1,t2,t12,s1 : Real;
  Label M2,M4;
begin
  nm := n - 1;
  if ind <> 0 then goto M2;
  a1[1] := 0;
  b1[1] := -2;
  c1[1] := 1;
  d1[1] := 3*(y1[2] - y1[1])/(x1[2]-x1[1]);
  For j := 2 to nm do
    Begin
      hj := x1[j+1] - x1[j];
      hj1 := x1[j] - x1[j-1];
      am := hj1/(hj1 + hj);
      al := 1 - am;
      a1[j] := al;
      b1[j] := -2;
      c1[j] := am;
      d1[j] := 3*(am*(y1[j+1]-y1[j])/hj+al*(y1[j]-y1[j-1])/hj1);
    end;
  a1[1] := 1;
  b1[1] := -2;
  c1[1] := 0;
  d1[1] := 3*(y1[n1] - y1[n1-1])/(x1[n1]-x1[n1-1]);

  PROGON(a1,b1,c1,d1,u1,v1,z1,n1);
  M2:
  For j := 2 to n do
    if x1[j] > xx then break;
  t := (xx1-x1[j-1])/(x1[j]-x1[j-1]);
  t1 := (1 - t);
```

```
   hj := x1[j] - x1[j-1];
   t2 := t*t;
   t12 := t1*t1;
   s1 := y1[j-1]*t12*(1 + 2 * t);
   s1 := y1[j]*t2*(3 - 2*t)+s1;
   s1 := z1[j-1]*hj*t*t12 + s1;
   s := s1-z1[j]*hj*t2*t1;
end;
Begin
  xx := 1;
  ind := 1;
  WriteLn('Итерполирование функций с помощью кубического сплайна:');
  For i := 1 to 19 do
    Begin
      Write('x=',xx:0:3,' ');
      SPLINE(n,x,y,a,b,c,d,u,v,z,xx,s,ind);
      WriteLn('y=',s:0:3);
      xx := xx + 0.5;
    end;
  ReadLn;
End.
```

Вычисления по программе привели к следующим результатам:

$x = 1.00 \quad y = 2.050$
$x = 1.50 \quad y = 1.995$
$x = 2.00 \quad y = 1.940$
$x = 2.50 \quad y = 1.930$
$x = 3.00 \quad y = 1.920$
$x = 3.50 \quad y = 1.895$
$x = 4.00 \quad y = 1.870$
$x = 4.50 \quad y = 1.820$
$x = 5.00 \quad y = 1.770$
$x = 5.50 \quad y = 1.825$
$x = 6.00 \quad y = 1.880$
$x = 6.50 \quad y = 1.795$
$x = 7.00 \quad y = 1.710$
$x = 7.50 \quad y = 1.655$
$x = 8.00 \quad y = 1.600$
$x = 8.50 \quad y = 1.580$
$x = 9.00 \quad y = 1.560$
$x = 9.50 \quad y = 1.480$
$x = 10.00 \quad y = 1.400$

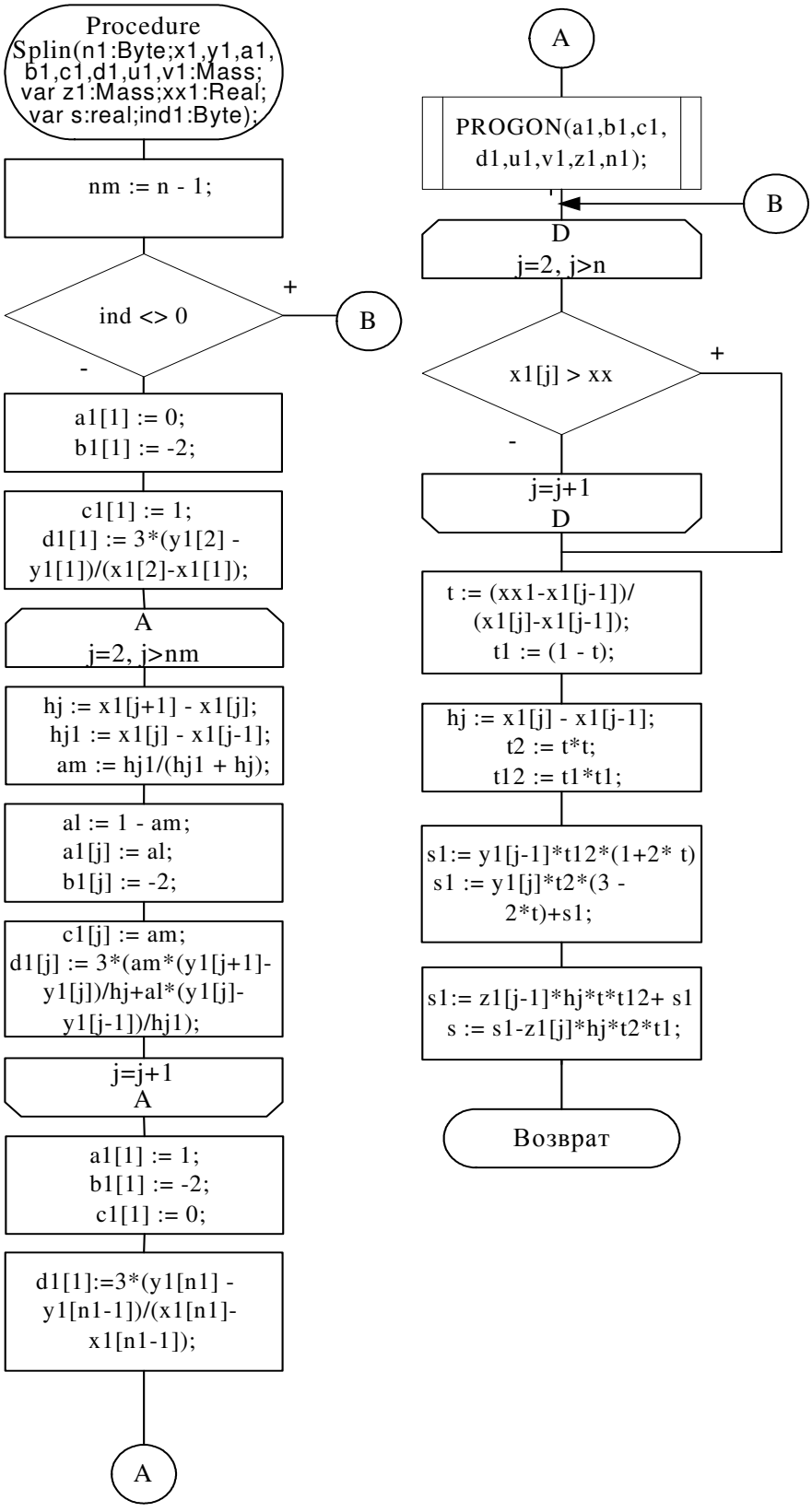


Рисунок 27 – Схема алгоритма приближения функций с помощью сплайнов

Варианты заданий для решения задачи приближения функций приведены в таблице 6.

Приближение полиномами Ньютона

Function New (*n:integer; x,y:Vector; q:real*)*:real;*

Входные параметры: *n* – число узлов интерполяции; *x*–массив размерности *n*, содержащий значения узлов интерполяции; *y* – массив размерности *n*, содержащий значения функции в узлах интерполяции; *q*– точка, в которой вычисляем значение функции.

Выходные параметры: приближенное значение функции в точке *q*, вычисленное с помощью интерполяционных формул Ньютона.

Схема алгоритма приведена на рисунке 28.

Пример. Интерполировать табличную функцию с помощью формул Ньютона в заданных точках.

x_i	y_i	x_i	y_i
1	2,05	6	1,88
2	1,94	7	1,71
3	1,92	8	1,60
4	1,87	9	1,56
5	1,77	10	1,40

Текст функции:

```
function New(n:integer;a,b:Vector;x:Real):real;
var i,j,k,f:integer; s,p:real; M:array [1..20,1..20] of real;
BEGIN
  for i:=2 to n do m[1,i-1]:=b[i]-b[i-1];
   for i:=2 to n-1 do
    begin
     f:=1;
```

```
      for k:=i downto 2 do f:=f*k;
      for j:=2 to n-i+1 do m[i,j-1]:=(m[i-1,j]-m[i-1,j-1])/f;
     end;
   if (a[n]-x)-(x-a[1])<0 then
     begin
      S:=b[n];
      for i:=1 to N-1 do
       begin
        P:=1;
        for j:=1 to i do P:=P*(x-a[n+1-j]);
        s:=s+P*m[i,n-i];
       end;
      New:=s;
     end
    else
     begin
      S:=b[1];
      for i:=1 to N-1 do
       begin
        P:=1;
        for j:=1 to i do P:=P*(x-a[j]);
        s:=s+P*m[i,1];
       end;
      New:=s
     end;
     END;
```

Вычисления по программе привели к следующим результатам:

$x = 1.00 \qquad y = 2.050$
$x = 1.50 \qquad y = 1.980$
$x = 2.00 \qquad y = 1.940$
$x = 2.50 \qquad y = 1.923$
$x = 3.00 \qquad y = 1.920$
$x = 3.50 \qquad y = 1.925$
$x = 4.00 \qquad y = 1.930$
$x = 4.50 \qquad y = 1.930$
$x = 5.00 \qquad y = 1.918$
$x = 5.50 \qquad y = 1.891$
$x = 6.00 \qquad y = 1.683$
$x = 6.50 \qquad y = 1.638$
$x = 7.00 \qquad y = 1.615$
$x = 7.50 \qquad y = 1.605$
$x = 8.00 \qquad y = 1.600$

$x = 8.50 \qquad y = 1.589$
$x = 9.00 \qquad y = 1.560$
$x = 9.50 \qquad y = 1.502$
$x = 10.00 \qquad y = 1.400$

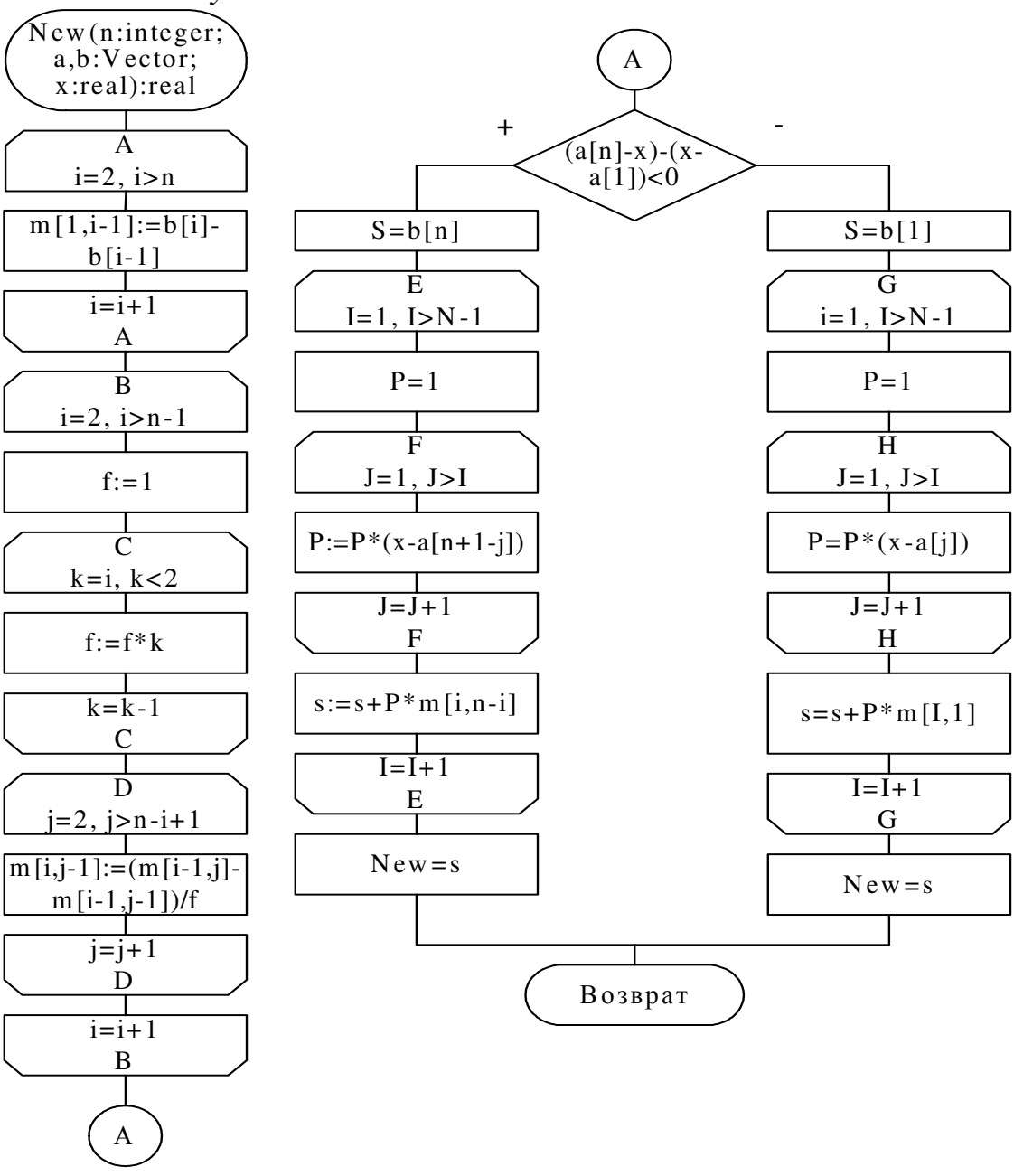


Рисунок 28 – Схема алгоритма приближения функций полиномами Ньютона

Варианты заданий для решения задачи приближения функций приведены в таблице 6.

Аппроксимация функций методом наименьших квадратов многочленом второй степени.

Function Kvadr(*n:integer; x,y:Vector; q:real; var c0,c1,c2:real*):*real*; (используется процедура: *SIMQ* для решения системы уравнений. Процедура описана в первой лабораторной работе).

Входные параметры: n – число узлов; x – масив размерности n, содержащий значения узлов; y – массив размерности n, содержащий значения функции в узлах; q – точка, в которой вычисляем значение функции.

Выходные параметры: приближенное значение функции в точке q присваивается имени функции; $c0,c1,c2$ – переменные содержат коэффициенты аппроксимирующего многочлена второй степени

$$P_2(x)=c0+c1* x +c2* x^2.$$

Схема алгоритма приведена на рисунке 29.

Пример. Аппроксимировать табличную функцию многочленом второй степени методом наименьших квадратов.

x_i	y_i	x_i	y_i
1	2,05	6	1,88
2	1,94	7	1,71
3	1,92	8	1,60
4	1,87	9	1,56
5	1,77	10	1,40

Текст функции:

```
function Kvadr(n:integer;x,y:Vector;q:real;var c0,c1,c2: real;):real;
 var i,j:integer;
    a:Matr;
    b,c:Vector;
    s:real;
 BEGIN
    a[1,1]:=N;
```

```
    s:=0;
    for  i:=1 to N do
        s:=s+x[i];
    a[1,2]:=s;
    a[2,1]:=s;
    s:=0;
    for  i:=1 to N do
        s:=s+x[i]*x[i];
    a[1,3]:=s;
    a[2,2]:=s;
    a[3,1]:=s;
s:=0;
    for  i:=1 to N do
        s:=s+x[i]*x[i]*x[i];
    a[2,3]:=s;
    a[3,2]:=s;
    s:=0;
    for  i:=1 to N do
        s:=s+x[i]*x[i]*x[i]*X[i];
    a[3,3]:=s;
    s:=0;
    for i:=1to N do s:=s+y[i];
    b[1]:=s;
    s:=0;
    for i:=1 to N do s:=s+y[i]*x[i];
    b[2]:=s;
    s:=0;
    for i:=1 to N do s:=s+y[i]*x[i]*x[i];
    b[3]:=s;
    j:=N;
    N:=3;
    SimQ(n,a,b,k);
    c0:=b[1];
    c1:=b[2];
    c2:=b[3];
    Kvadr:=b[1]+b[2]*q+b[3]*q*q;
END;
```

Вычисления по программе привели к следующим результатам:

$c0$= 2.0226666666E+00
$c1$=-1.5181818164E-02
$c2$=-4.3939393955E-03
x =1.00 y =2.003
x =1.50 y =1.990
x =2.00 y =1.975

x =2.50 y =1.957
x =3.00 y=1.938
x =3.50 y =1.916
x =4.00 y=1.892
x =4.50 y =1.865
x =5.00 y =1.837
x =5.50 y =1.806
x =6.00 y =1.773
x =6.50 y =1.738
x =7.00 y =1.701
x =7.50 y =1.662
x =8.00 y =1.620
x =8.50 y =1.576
x =9.00 y =1.530
x =9.50 y =1.482
x =10.00 y =1.431

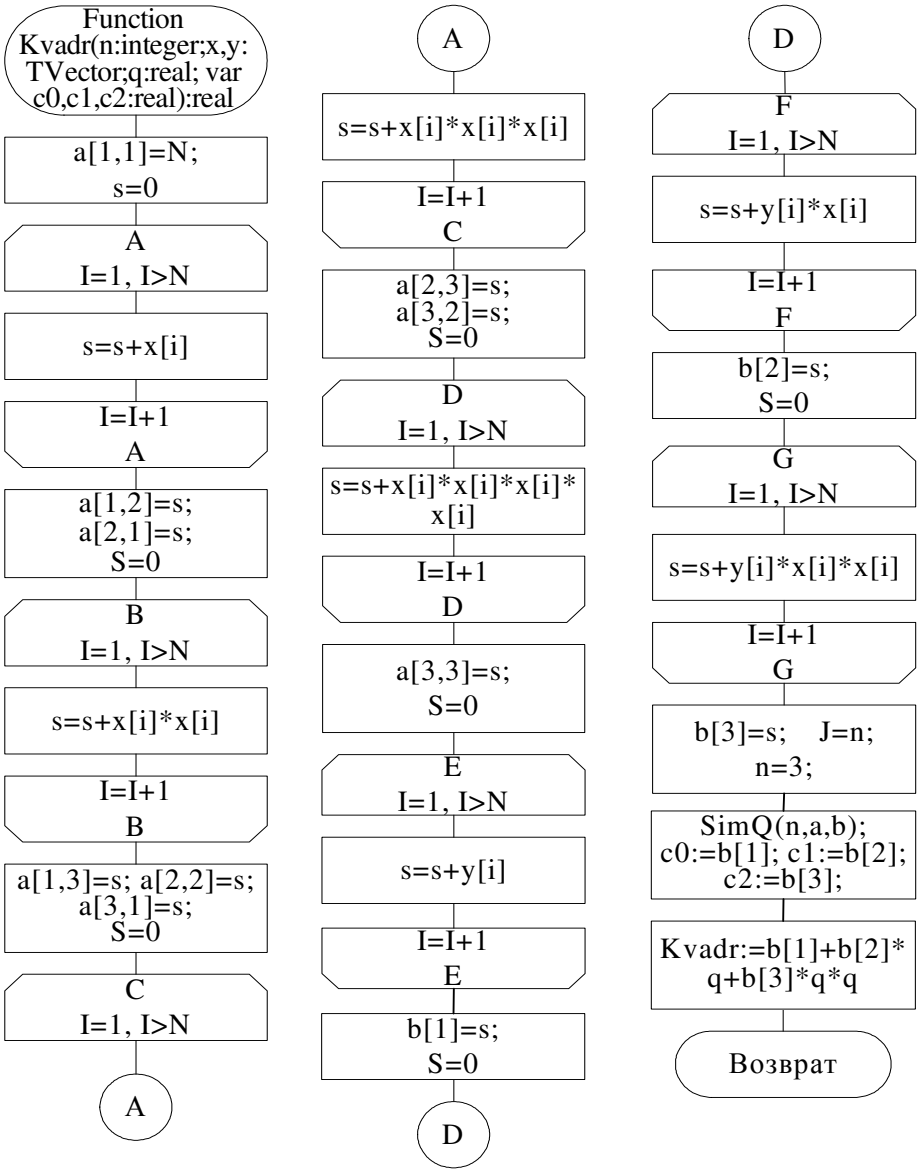


Рисунок 29 – Схема алгоритма аппроксимации функций методом наименьших квадратов многочленом второй степени

Варианты заданий. Значения $x_i = i \times 0{,}1$; $i = 1,2 \ldots 20$ одинаковые для всех вариантов

Таблица 6

	Значения $y_i = y(x_i)$					
	Вариант 1	Вариант 2	Вариант 3	Вариант 4	Вариант 5	Вариант 6
1	2	3	4	5	6	7
1	2,05	2,09	2,02	1,99	2,23	2,07
2	1,94	2,05	1,98	2,03	2,29	2,17
3	1,92	2,19	1,67	2,20	2,27	2,21
4	1,87	2,18	1,65	2,39	2,62	2,31
5	1,77	2,17	1,57	2,19	2,72	2,10
6	1,88	2,27	1,42	2,61	2,82	2,09

7	1,71	2,58	1,37	2,35	3,13	2,12
8	1,60	2,73	1,07	2,60	3,49	1,63
9	1,56	2,82	0,85	2,55	3,82	1,78
10	1,40	3,04	0,48	2,49	3,95	1,52
11	1,50	3,03	0,35	2,50	4,22	1,16
12	1,26	3,45	-0,30	2,52	4,48	1,07
13	0,99	3,62	-0,61	2,44	5,06	0,85
14	0,97	3,85	-1,20	2,35	5,50	0,56
15	0,91	4,19	-1,39	2,26	5,68	0,10
16	0,71	4,45	-1,76	2,19	6,19	-0,25
17	0,43	4,89	-2,28	2,24	6,42	-0,65
18	0,54	5,06	-2,81	2,34	7,04	-1,06
19	0,19	5,63	-3,57	1,96	7,57	-1,66
20	0,01	5,91	-4,06	2,19	8,10	-2,01

	Значения $y_i = y(x_i)$					
	Вариант 7	Вариант 8	Вариант 9	Вариант 10	Вариант 11	Вариант12
1	2,18	-0,10	-0,16	2,09	2,15	0,10
2	2,43	-0,21	0,01	2,31	2,41	-0,01
3	2,40	0,01	0,10	2,72	2,58	-0,19
4	2,43	0,05	0,16	2,77	2,84	-0,11
5	2,65	-0,13	0,05	2,78	3,28	-0,31
6	2,75	-0,23	0,35	2,97	3,46	-0,78
7	2,67	-0,21	0,19	3,00	4,02	-0,64
8	2,66	-0,43	0,50	3,51	4,11	-0,85
9	2,63	-0,57	0,74	3,43	4,61	-1,18
10	2,75	-0,44	1,03	3,58	5,03	-1,39
11	2,41	-0,44	1,06	3,58	5,34	-1,79
12	2,24	-0,83	1,49	3,54	5,86	-2,02
13	2,12	-0,78	1,79	3,82	6,33	-2,48
14	1,74	-0,81	2,03	3,90	6,81	-2,93

Продолжение таблицы 6

1	2	3	4	5	6	7
15	1,57	-1,06	2,22	3,77	7,21	-3,26
16	1,17	-1,41	2,50	3,81	7,67	-3,91
17	0,96	-1,40	2,88	4,00	8,23	-4,41
18	0,63	-1,70	3,21	3,97	8,68	-4,91
19	0,25	-1,96	3,63	4,08	9,35	-5,30
20	-0,01	-1,91	3,90	4,08	9,93	-6,00
	Значения $y_i = y(x_i)$					
	Вариант 13	Вариант 14	Вариант 15	Вариант 16	Вариант 17	Вариант 18
1	0,17	0,80	0,04	0,08	-0,02	0,14
2	0,07	0,29	0,47	0,14	0,44	0,23
3	0,17	0,52	0,78	0,37	0,51	0,44
4	0,05	0,77	1,01	0,36	0,67	0,54
5	0,12	0,93	1,19	0,44	0,69	0,72
6	0,00	1,20	1,60	0,48	1,04	3,76
7	0,01	1,20	1,93	0,27	1,1	0,37
8	-0,05	1,35	2,22	0,39	1,3	0,64
9	-0,21	1,39	2,50	0,50	1,7	357
10	-0,50	1,48	3,01	0,48	2,0	3,44
11	-0,50	1,52	3,22	0,69	2,1	3,41
12	-0,86	1,71	3,71	0,50	2,4	0,30
13	-1,24	1,72	4,23	0,31	2,90	-0,0
14	-1,47	1,87	4,78	0,37	3,50	-0,03
15	-1,79	1,86	5,27	0,43	3,99	-0,47
16	-2,25	1,89	5,75	0,33	4,06	-0,68
17	-2,55	2,04	6,16	0,31	4,54	-0,93
18	-3,18	1,73	6,76	0,09	4,99	-1,28
19	-3,60	2,04	7,30	0,08	5,36	-1,53
20	-3,93	2,03	8,00	0,03	5,99	-1,93
	Значения $y_i = y(x_i)$					
	Вариант 19	Вариант 20	Вариант 21	Вариант 22	Вариант 23	Вариант 24
1	-1,86	- 1,65	-1,89	-1,84	-1,92	-1,90
2	-1,95	- 2,00	-2,07	-1,98	-1,60	-1,80
3	-2,12	- 1,87	-2,30	-1,72	-1,57	-1,82
4	-2,06	- 1,89	-2,26	- 1,58	-1,41	-1,86
5	-2,15	- 1,75	-2,34	- 1,69	-1,36	-1,83
6	-2,00	- 1,59	-2,66	- 1,59	-0,97	-2,00
7	-2,12	-1,44	-2,88	-1,58	-0,59	-2,01
8	-2,31	-1 ,61	-2,85	-1,64	-0,71	-2,05
9	-2,29	- 1,00	-3,16	- 1,55	-0,15	-2,46
10	-2,57	- 1,17	-3,49	- 1,35	0,01	-2,68

Продолжение таблицы 6

1	2	3	4	5	6	7
11	-2,56	-0,87	-3,88	- 1,33	0,22	-2,85
12	-2,86	-0,47	-4,22	- 1,47	0,63	-2,98
13	-2,85	-0,33	-4,45	- 1,50	1,07	3,30
14	-3,03	-0,00	-4,99	- 2,65	1,42	-3,40
15	-3,25	0,34	-5,36	- 1,65	1,68	-3,90
16	-3,08	0,49	-5,71	- 1,87	2,49	-4,37
17	-3,29	0,81	-6,51	- 1,61	2,57	-4,65
18	-3,67	1,37	-6,76	- 1,86	3,09	-5,00
19	-3,70	1,72	-7,35	- 1,84	3,40	-5,42
20	-3,85	2,03	-8,02	- 1,91	4,00	-6,13
	Значения $y_i = y\ (x_i)$					
	Вариант 25	Вариант26	Вариант27	Вариант 28	Вариант 29	Вариант 30
1	-1,80	-1,65	-1,88	- ,01	-4,13	-3,97
2	-1,66	-1,64	-1,69	- ,06	-4,11	-4,07
3	-1,36	-1,41	-1,52	- ,88	-3,87	-4,04
4	-1,41	-0,91	-1,55	-3,98	-3,74	-4,30
5	-1,13	-0,63	-1,16	-4,36	-3,85	-4,27
6	-0,82	-0,34	-1,27	-4,18	-3,71	-4,54
7	-0,74	-0,12	-1,23	-4,16	-3,53	-4,79
8	-076	0,25	-1,36	-4,51	-3,56	-5,07
9	-0,64	0,64	-1,26	-4,53	-3,19	-5,30
10	-0,46	0,96	-1,47	-4,38	-3.04	-5,51
11	-0,30	1,50	-1,72	-4,76	-2,83	-5,83
12	-0,27	1,77	-1,76	-4,66	-2,54	-6,06
13	-0,22	2,24	-2,00	-4,82	-2,41	-6,40
14	-0,11	2,93	-2,03	-4,77	-1,97	-6,83
15	-0,02	3,17	-2,35	-5,12	-1,78	-7,54
16	0,11	3,77	-2,46	-5,23	-1,53	-7,68
17	0,11	4,42	-2,88	-5,40	-1,04	-8,36
18	-0,02	4,79	-3,27	-5,84	-0,86	-8,91
19	0,03	5,50	-3,68	-5,86	-0,48	-9,39
20	0,01	6,01	-3,98	-6,01	0,09	-9,98

Лабораторная работа №8
Решение задачи Коши
Одношаговые методы

Метод Эйлера

Для всех методов, которые будут описаны далее, определим следующие типы:

```
type
 TF=Function (x,y1,y2:real):real; // функциональный тип;
 TFMas=array [1..2] of TF; // массив функций типа TF;
 TFunZnach=array [1..2,1..150] of real; //значения 2-х функций;
 TMas=array [1..2] of real;
```

Описание массива *TFMas* для примера на стр. 199:

```
Function  SQRN(a,b:real):double;
begin
 if a<>0 then    sqrn:=exp(b*ln(a))
  else    sqrn:=0;
end;

function f1 (x,y1,y2:real):real;
begin
 Result:=y2;
end;

function f2(x,y1,y2:real):real;
begin
 Result:=SQRN(2.71828182846,-x*y1);
end;

var
Fmas:TFMas;
//..
Fmas[1]:=f1;
Fmas[2]:=f2;
//..
```

Перед вычислением нужно:

1. Задать массив *TFMas*;
2. Присвоить элементам массива *TFMas* конкретные функции;
3. Задать массив *TFunZnach*;
4. Присвоить первому столбцу массива *TFunZnach* значения;
5. Задать переменную *x* и присвоить ей значение.

Procedure eiler (*a, b: real; n, kolfun*: *integer; var x: real*; *f: TFMas; y_*1: *TFunZnach*);

Входные параметры: *a,b* – отрезок интегрирования; *n* – количество точек приближения; *Kolfun* – порядок системы; *x* – начальное значение x_0; *f* – массив функций, содержащий правые части системы; *y_*1 – массив, содержащий начальные значения *y*.

Выходные параметры: *y_*1 – массив, содержащий приближенное решение системы.

Схема алгоритма показана на рисунке 30.

Пример. Решить на отрезке [0,3] с шагом 0,1 задачу Коши

$$\begin{cases} dy_1 / dx = y_2 \\ dy_2 / dx = e^{-xy_1} \end{cases} \qquad \begin{cases} y_1(0) = 0 \\ y_2(0) = 0 \end{cases}$$

Текст процедуры:

```
procedure eiler (a,b:real;n,kolfun:integer;var x:real;f:TFMas; y_1:TFunZnach);
 var
 i,k:integer;
 t:real;
 begin
  Form1.StringGrid1.Cells[0,1]:='Y1';
  Form1.StringGrid1.Cells[0,2]:='Y2';
  t:=(b-a)/n;
   x:=x+t;
  for i:=2 to n do   begin
   for k:=1 to kolfun do y_1[k,i]:=y_1[k,i-1]+t*f[k](x,y_1[1,i-1],y_1[2,i-1]);
// Выводим значения
   Form1.StringGrid1.Cells[i,1]:=FloatToStrF(y_1[1,i],ffExponent,4,6);
   Form1.StringGrid1.Cells[i,0]:='X='+FloatToStr(x);
   Form1.StringGrid1.Cells[i,2]:=FloatToStrF(y_1[2,i],ffExponent,4,6);
```

```
      x:=x+t;
  end;
//Выводим значения
  Form1.StringGrid1.Cells[1,1]:=FloatToStrF(y_1[1,1],ffExponent,4,6);
 Form1.StringGrid1.Cells[1,0]:='X='+FloatToStr(a);
 Form1.StringGrid1.Cells[1,2]:=FloatToStrF(y_1[2,1],ffExponent,4,6);
 end;
```

Вычисления по программе привели к следующим результатам:

$x = 0$	$y_1 = 0{,}000E+0$	$y_2 = 6{,}269E-2$
$x = 0{,}1$	$y_1 = 0{,}000E+0$	$y_2 = 1{,}000E-1$
$x = 0{,}2$	$y_1 = 1{,}050E-2$	$y_2 = 2{,}000E-1$
$x = 0{,}3$	$y_1 = 3{,}150E-2$	$y_2 = 2{,}996E-1$
$x = 0{,}4$	$y_1 = 6{,}296E-2$	$y_2 = 3{,}981E-1$
$x = 0{,}5$	$y_1 = 1{,}048E-1$	$y_2 = 4{,}946E-1$
$x = 0{,}6$	$y_1 = 1{,}567E-1$	$y_2 = 5{,}877E-1$
$x = 0{,}7$	$y_1 = 2{,}184E-1$	$y_2 = 6{,}761E-1$
$x = 0{,}8$	$y_1 = 2{,}894E-1$	$y_2 = 7{,}583E-1$
$x = 0{,}9$	$y_1 = 3{,}690E-1$	$y_2 = 8{,}333E-1$
$x = 1$	$y_1 = 4{,}532E-1$	$y_2 = 8{,}957E-1$
$x = 1{,}1$	$y_1 = 5{,}474E-1$	$y_2 = 9{,}559E-1$
$x = 1{,}2$	$y_1 = 6{,}480E-1$	$y_2 = 1{,}008E+0$
$x = 1{,}3$	$y_1 = 7{,}539E-1$	$y_2 = 1{,}051E+0$
$x = 1{,}4$	$y_1 = 8{,}645E-1$	$y_2 = 1{,}086E+0$
$x = 1{,}5$	$y_1 = 9{,}787E-1$	$y_2 = 1{,}114E+0$
$x = 1{,}6$	$y_1 = 1{,}096E+0$	$y_2 = 1{,}136E+0$
$x = 1{,}7$	$y_1 = 1{,}215E+0$	$y_2 = 1{,}152E+0$
$x = 1{,}8$	$y_1 = 1{,}336E+0$	$y_2 = 1{,}164E+0$
$x = 1{,}9$	$y_1 = 1{,}459E+0$	$y_2 = 1{,}172E+0$
$x = 2$	$y_1 = 1{,}582E+0$	$y_2 = 1{,}178E+0$
$x = 2{,}1$	$y_1 = 1{,}706E+0$	$y_2 = 1{,}182E+0$
$x = 2{,}2$	$y_1 = 1{,}830E+0$	$y_2 = 1{,}184E+0$
$x = 2{,}3$	$y_1 = 1{,}955E+0$	$y_2 = 1{,}186E+0$
$x = 2{,}4$	$y_1 = 2{,}080E+0$	$y_2 = 1{,}187E+0$
$x = 2{,}5$	$y_1 = 2{,}204E+0$	$y_2 = 1{,}188E+0$
$x = 2{,}6$	$y_1 = 2{,}329E+0$	$y_2 = 1{,}188E+0$
$x = 2{,}7$	$y_1 = 2{,}454E+0$	$y_2 = 1{,}188E+0$
$x = 2{,}8$	$y_1 = 2{,}579E+0$	$y_2 = 1{,}188E+0$
$x = 2{,}9$	$y_1 = 2{,}704E+0$	$y_2 = 1{,}188E+0$
$x = 3$	$y_1 = 2{,}812E+0$	$y_2 = 1{,}188E+0$

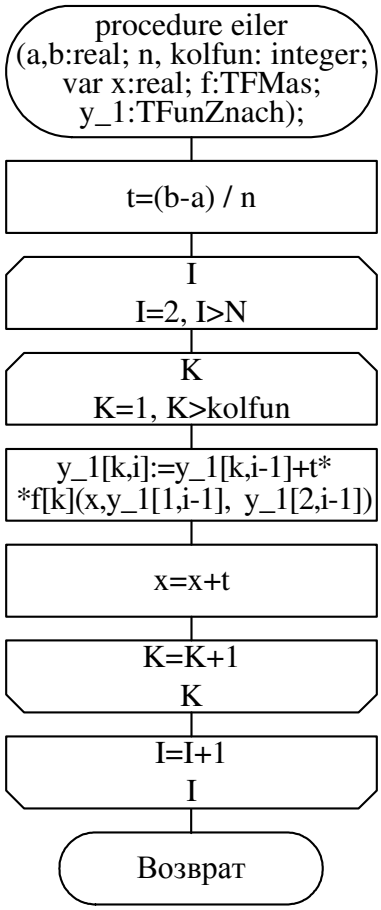


Рисунок 30 – Схема алгоритма метода Эйлера

Варианты заданий для решения задачи Коши методом Эйлера приведены в таблице 7.

Метод Эйлера-Коши

Procedure prognoz (*a,b: real; n, kolfun: integer; var x: real; f: TFMas; var y_1: TFunZnach*);

Входные параметры: *a,b* – отрезок интегрирования; *n* – количество точек приближения; *Kolfun* – порядок системы; *x* – начальное значение x_0; *f* – массив функций, содержащий правые части системы; *y_1* – массив, содержащий начальное значение *y*.

Выходные параметры: *y_1* – массив, содержащий приближенное решение системы.

Схема алгоритма показана на рисунке 31.

Пример. Решить на отрезке [0,3] с шагом 0,1 задачу Коши

$$\begin{cases} dy_1 / dx = y_2 \\ dy_2 / dx = e^{-xy_1} \end{cases} \quad \begin{cases} y_1(0) = 0 \\ y_2(0) = 0 \end{cases}$$

Текст процедуры:

```
procedure prognoz (a,b:real;n,kolfun:integer;var x:real;f:TFMas; var y_1:TFunZnach);
 var
 i,k:integer;
 t:real;
 begin
  t:=(b-a)/n;
   x:=x+t;
  for i:=2 to n do   begin
   for k:=1 to kolfun do
   y_1[k,i]:=y_1[k,i-1]+t*f[k](x+0.5*t,y_1[1,i-1]+0.5*t*y_1[1,i-1],y_1[2,i-
1]+0.5*t*y_1[2,i-1]);
   // Выводим значения
Form1.StringGrid1.Cells[i,1]:=FloatToStrF(y_1[1,i],ffExponent,4,6);
   Form1.StringGrid1.Cells[i,0]:='X='+FloatToStr(x);
   Form1.StringGrid1.Cells[i,2]:=FloatToStrF(y_1[2,i],ffExponent,4,6);
      x:=x+t;
  end;
  // Выводим значения
  Form1.StringGrid1.Cells[1,1]:=FloatToStrF(y_1[1,1],ffExponent,4,6);
 Form1.StringGrid1.Cells[1,0]:='X='+FloatToStr(a);
 Form1.StringGrid1.Cells[1,2]:=FloatToStrF(y_1[2,1],ffExponent,4,6);
 end;
```

Вычисления по программе привели к следующим результатам:

$x = 0$	$y_1 = 0{,}000E+0$	$y_2 = 6{,}269E-2$
$x = 0{,}1$	$y_1 = 0{,}000E+0$	$y_2 = 1{,}000E-1$
$x = 0{,}2$	$y_1 = 1{,}050E-2$	$y_2 = 2{,}000E-1$
$x = 0{,}3$	$y_1 = 3{,}150E-2$	$y_2 = 2{,}996E-1$
$x = 0{,}4$	$y_1 = 6{,}296E-2$	$y_2 = 3{,}981E-1$
$x = 0{,}5$	$y_1 = 1{,}048E-1$	$y_2 = 4{,}946E-1$
$x = 0{,}6$	$y_1 = 1{,}567E-1$	$y_2 = 5{,}877E-1$
$x = 0{,}7$	$y_1 = 2{,}184E-1$	$y_2 = 6{,}761E-1$
$x = 0{,}8$	$y_1 = 2{,}894E-1$	$y_2 = 7{,}583E-1$
$x = 0{,}9$	$y_1 = 3{,}690E-1$	$y_2 = 8{,}333E-1$
$x = 1$	$y_1 = 4{,}565E-1$	$y_2 = 8{,}998E-1$
$x = 1{,}1$	$y_1 = 5{,}510E-1$	$y_2 = 9{,}575E-1$
$x = 1{,}2$	$y_1 = 6{,}515E-1$	$y_2 = 1{,}006E+0$

$x = 1{,}3$	$y_1 = 7{,}572\text{E-}1$	$y_2 = 1{,}046\text{E+}0$
$x = 1{,}4$	$y_1 = 8{,}670\text{E-}1$	$y_2 = 1{,}077\text{E+}0$
$x = 1{,}5$	$y_1 = 9{,}801\text{E-}1$	$y_2 = 1{,}102\text{E+}0$
$x = 1{,}6$	$y_1 = 1{,}096\text{E+}0$	$y_2 = 1{,}120\text{E+}0$
$x = 1{,}7$	$y_1 = 1{,}213\text{E+}0$	$y_2 = 1{,}133\text{E+}0$
$x = 1{,}8$	$y_1 = 1{,}332\text{E+}0$	$y_2 = 1{,}143\text{E+}0$
$x = 1{,}9$	$y_1 = 1{,}452\text{E+}0$	$y_2 = 1{,}149\text{E+}0$
$x = 2$	$y_1 = 1{,}573\text{E+}0$	$y_2 = 1{,}154\text{E+}0$
$x = 2{,}1$	$y_1 = 1{,}694\text{E+}0$	$y_2 = 1{,}157\text{E+}0$
$x = 2{,}2$	$y_1 = 1{,}816\text{E+}0$	$y_2 = 1{,}158\text{E+}0$
$x = 2{,}3$	$y_1 = 1{,}937\text{E+}0$	$y_2 = 1{,}160\text{E+}0$
$x = 2{,}4$	$y_1 = 2{,}059\text{E+}0$	$y_2 = 1{,}160\text{E+}0$
$x = 2{,}5$	$y_1 = 2{,}181\text{E+}0$	$y_2 = 1{,}161\text{E+}0$
$x = 2{,}6$	$y_1 = 2{,}303\text{E+}0$	$y_2 = 1{,}161\text{E+}0$
$x = 2{,}7$	$y_1 = 2{,}425\text{E+}0$	$y_2 = 1{,}161\text{E+}0$
$x = 2{,}8$	$y_1 = 2{,}546\text{E+}0$	$y_2 = 1{,}161\text{E+}0$
$x = 2{,}9$	$y_1 = 2{,}668\text{E+}0$	$y_2 = 1{,}161\text{E+}0$
$x = 3$	$y_1 = 2{,}714\text{E+}0$	$y_2 = 1{,}161\text{E+}0$

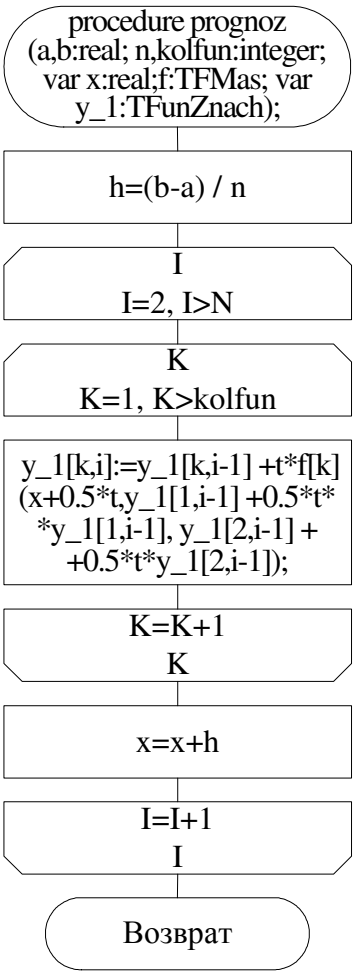


Рисунок 31 – Схема алгоритма метода Эйлера- Коши

Варианты заданий для решения задачи Коши методом Эйлера-Коши приведены в таблице 7.

Метод Рунге-Кутта 4-го порядка

Procedure Runge_Kut (*a,b: real; n: integer; var x: real;*
var y_1: TFunZnach; FMas: TFMas);

Входные параметры: *a,b* – отрезок интегрирования; *n* – количество точек приближения; *Kolfun* – порядок системы; *x* – начальное значение x_0; *f* – массив функций, содержащий правые части системы; *y*_1 – массив, содержащий начальное значение *y*.

Выходные параметры: *y*_1 – массив, содержащий приближенное решение системы.

Схема алгоритма показана на рисунке 32.

Пример. Решить на отрезке [0,3] с шагом 0,1 задачу Коши

$$\begin{cases} dy_1 / dx = y_2 \\ dy_2 / dx = e^{-xy_1} \end{cases} \qquad \begin{cases} y_1(0) = 0 \\ y_2(0) = 0 \end{cases}$$

Текст процедуры:

```
Procedure Runge_Kut(a,b:real;n:integer;var x:real;var y_1:TFunZnach;FMas:TFMas);
var
 i,j:integer;
 k:array [1..4] of real;
 h:real;
begin
  h:=(b-a)/n;
 for i:=2 to n do
  begin
   for j:=1 to 2 do
    begin
     k[1]:=FMas[j](x,y_1[1,i-1],y_1[2,i-1]);
     k[2]:=FMas[j](x+h/2,y_1[1,i-1]+h/2*k[1],y_1[2,i-1]+h/2*k[1]);
     k[3]:=FMas[j](x+h/2,y_1[1,i-1]+h/2*k[2],y_1[2,i-1]+h/2*k[2]);
     k[4]:=FMas[j](x+h,y_1[1,i-1]+h*k[3],y_1[2,i-1]+h*k[3]);
     y_1[j,i]:=y_1[j,i-1]+h/6*(k[1]+2*k[2]+2*k[3]+k[4]);
    end;
   x:=x+h;
  end;
 end;
```

Вычисления по программе привели к следующим результатам:

$x = 0$	$y_1 = 0{,}000\text{E+0}$	$y_2 = 6{,}266\text{E-2}$
$x = 0{,}1$	$y_1 = 0{,}000\text{E+0}$	$y_2 = 9{,}967\text{E-2}$
$x = 0{,}2$	$y_1 = 1{,}048\text{E-2}$	$y_2 = 1{,}988\text{E-1}$
$x = 0{,}3$	$y_1 = 3{,}139\text{E-2}$	$y_2 = 2{,}973\text{E-1}$
$x = 0{,}4$	$y_1 = 6{,}266\text{E-2}$	$y_2 = 3{,}944\text{E-1}$
$x = 0{,}5$	$y_1 = 1{,}041\text{E-2}$	$y_2 = 4{,}895\text{E-1}$
$x = 0{,}6$	$y_1 = 1{,}556\text{E-1}$	$y_2 = 5{,}815\text{E-1}$
$x = 0{,}7$	$y_1 = 2{,}168\text{E-1}$	$y_2 = 6{,}693\text{E-1}$
$x = 0{,}8$	$y_1 = 2{,}872\text{E-1}$	$y_2 = 7{,}516\text{E-1}$
$x = 0{,}9$	$y_1 = 3{,}662\text{E-1}$	$y_2 = 8{,}274\text{E-1}$
$x = 1$	$y_1 = 4{,}532\text{E-1}$	$y_2 = 8{,}957\text{E-1}$
$x = 1{,}1$	$y_1 = 5{,}474\text{E-1}$	$y_2 = 9{,}559\text{E-1}$

$x = 1{,}2$	$y_1 = 6{,}480\text{E-}1$	$y_2 = 1{,}008\text{E+}0$
$x = 1{,}3$	$y_1 = 7{,}539\text{E-}1$	$y_2 = 1{,}051\text{E+}0$
$x = 1{,}4$	$y_1 = 8{,}645\text{E-}1$	$y_2 = 1{,}086\text{E+}0$
$x = 1{,}5$	$y_1 = 9{,}787\text{E-}1$	$y_2 = 1{,}114\text{E+}0$
$x = 1{,}6$	$y_1 = 1{,}096\text{E+}0$	$y_2 = 1{,}136\text{E+}0$
$x = 1{,}7$	$y_1 = 1{,}215\text{E+}0$	$y_2 = 1{,}152\text{E+}0$
$x = 1{,}8$	$y_1 = 1{,}336\text{E+}0$	$y_2 = 1{,}164\text{E+}0$
$x = 1{,}9$	$y_1 = 1{,}459\text{E+}0$	$y_2 = 1{,}172\text{E+}0$
$x = 2$	$y_1 = 1{,}582\text{E+}0$	$y_2 = 1{,}178\text{E+}0$
$x = 2{,}1$	$y_1 = 1{,}706\text{E+}0$	$y_2 = 1{,}182\text{E+}0$
$x = 2{,}2$	$y_1 = 1{,}830\text{E+}0$	$y_2 = 1{,}184\text{E+}0$
$x = 2{,}3$	$y_1 = 1{,}955\text{E+}0$	$y_2 = 1{,}186\text{E+}0$
$x = 2{,}4$	$y_1 = 2{,}080\text{E+}0$	$y_2 = 1{,}187\text{E+}0$
$x = 2{,}5$	$y_1 = 2{,}204\text{E+}0$	$y_2 = 1{,}188\text{E+}0$
$x = 2{,}6$	$y_1 = 2{,}329\text{E+}0$	$y_2 = 1{,}188\text{E+}0$
$x = 2{,}7$	$y_1 = 2{,}454\text{E+}0$	$y_2 = 1{,}188\text{E+}0$
$x = 2{,}8$	$y_1 = 2{,}579\text{E+}0$	$y_2 = 1{,}188\text{E+}0$
$x = 2{,}9$	$y_1 = 2{,}704\text{E+}0$	$y_2 = 1{,}188\text{E+}0$
$x = 3$	$y_1 = 2{,}795\text{E+}0$	$y_2 = 1{,}188\text{E+}0$

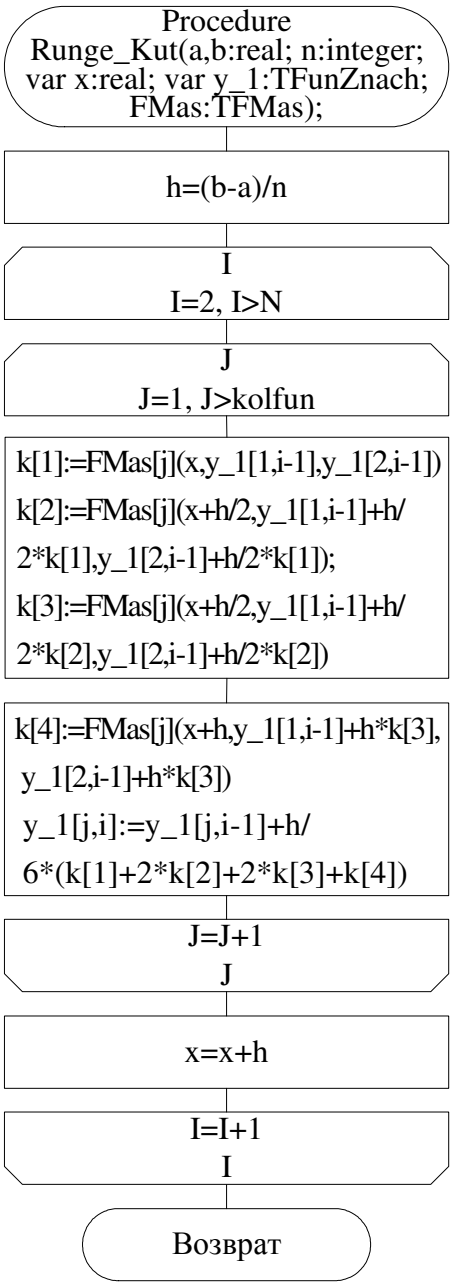


Рисунок 32 – Схема алгоритма метода Рунге-Кутта

Варианты заданий для решения задачи Коши методом Рунге-Кутта приведены в таблице 7.

Лабораторная работа №9
Решение задачи Коши
Многошаговые методы

Метод Адамса (явный)

Procedure Adams (*a,b,h: real; NewValues:TMas;*
FMas: TFMas); (использует процедуру *Runge_Kut*);
Вычисляет все значения функций на отрезке *ab*.

Входные параметры: *a* – начало отрезка; *b* – конец отрезка; *h* – шаг приращения аргумента *x*; *NewValues* – массив (размерность 2), содержащий начальные приближенные значения функций (необходимо создать и заполнить значениями из варианта до вызова процедуры); *FMas* – массив (размерность 2), содержащий указатели на функции правых частей системы (элементам массива присваиваются функции *f*1, *f*2, которые вычисляют значения правых частей системы, до вызова процедуры. Функции определяются для каждого варианта отдельно).

Выходные параметры: процедура выводит результат на экран во время работы.

(Процедура использует следующие процедуры и функции:

Procedure Runge_Kut (*a,h:real; var x:real;*
var y_1:TFunZnach; FMas:TFMas); – вычисляет начальные значения *Y* методом Рунге-Кутта для метода Адамса.

Все параметры формирует процедура *Adams*.

Входные параметры: *a* – начальная точка отрезка; *h* – шаг приращения *X*; *x* – начальное значение *X*; *y*_1 – массив (размерностью 2×4) содержащий приближенные значения функций в точке *A*; *FMas* – массив содержащий указатели на функции правых частей (передается в процедуру *Adams*).

Выходные параметры: *x* – значение *X*, на котором остановилось вычисление; *y*_1 – массив, содержащий первые 4 значения *Y*.

Функция *SQRN*(*a,b:real*):*double*; – возводит число *A* в степень *B*.)

Схема алгоритма приведена на рисунке 33.

Пример. Решить задачу Коши на отрезке [0,3] с шагом 0,1.

$$\begin{cases} dy_1 / dx = y_2 \\ dy_2 / dx = e^{-xy_1} \end{cases} \qquad \begin{cases} y_1(0) = 0 \\ y_2(0) = 0 \end{cases}$$

$F_1(x,y_1,y_2)$	$F_2(x,y_1,y_2)$	$y_1(a)$	$y_2(a)$	a	b
y_2	e^{-xy_1}	0	0	0	3

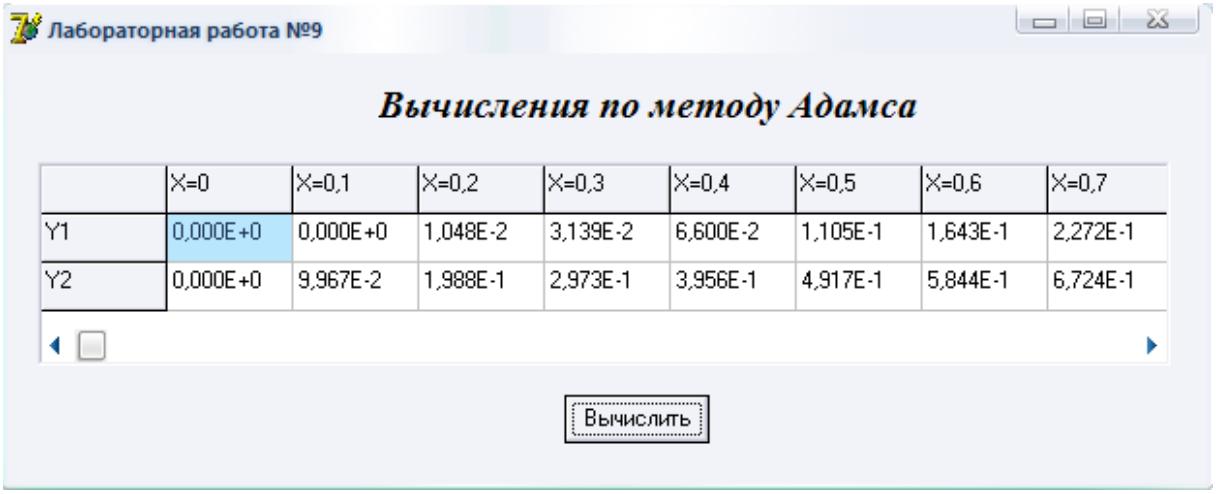
Текст программы:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    StringGrid1: TStringGrid;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.dfm}

type
 TF=Function (x,y1,y2:real):real;
 TFMas=array [1..2] of TF;
 TFunZnach=array [1..2,1..4] of real;
 TMas=array [1..2] of real;

Function  SQRN(a,b:real):double;
begin
 if a<>0 then
```

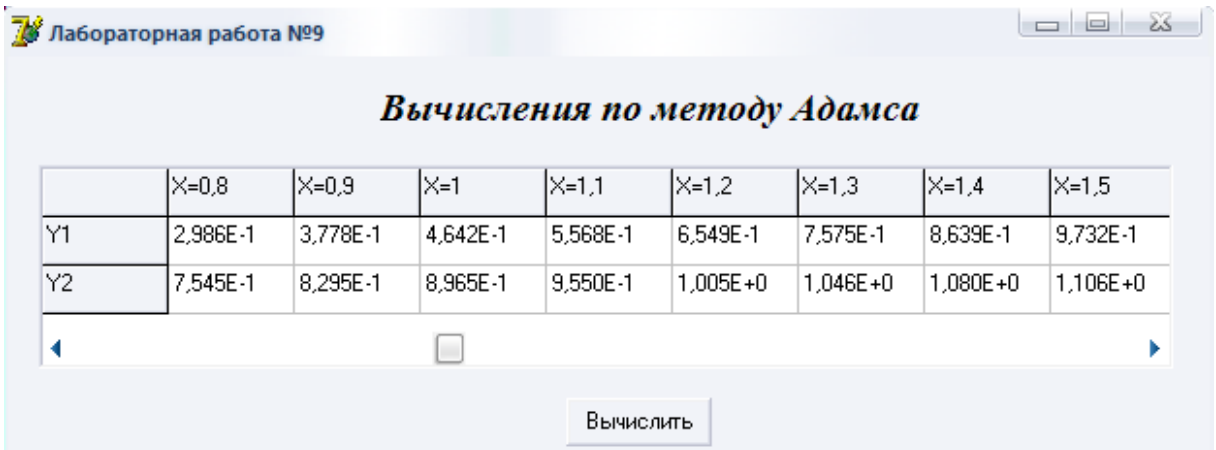
```
   sqrn:=exp(b*ln(a))
  else
   sqrn:=0;
end;

function f1 (x,y1,y2:real):real;
begin
 Result:=y2;
end;

function f2(x,y1,y2:real):real;
begin
 Result:=SQRN(2.71828182846,-x*y1);
end;

Procedure Runge_Kut(a,h:real;var x:real;var y_1:TFunZnach;FMas:TFMas);
var
 i,j:integer;
 k:array [1..4] of real;
begin
 for i:=2 to 4 do
  begin
   for j:=1 to 2 do
    begin
     k[1]:=FMas[j](x,y_1[1,i-1],y_1[2,i-1]);
     k[2]:=FMas[j](x+h/2,y_1[1,i-1]+h/2*k[1],y_1[2,i-1]+h/2*k[1]);
     k[3]:=FMas[j](x+h/2,y_1[1,i-1]+h/2*k[2],y_1[2,i-1]+h/2*k[2]);
     k[4]:=FMas[j](x+h,y_1[1,i-1]+h*k[3],y_1[2,i-1]+h*k[3]);
     y_1[j,i]:=y_1[j,i-1]+h/6*(k[1]+2*k[2]+2*k[3]+k[4]);
    end;
   x:=x+h;
   Form1.StringGrid1.Cells[i,1]:=FloatToStrF(y_1[1,i],ffExponent,4,6);
   Form1.StringGrid1.Cells[i,0]:='X='+FloatToStr(x);
   Form1.StringGrid1.Cells[i,2]:=FloatToStrF(y_1[2,i],ffExponent,4,6);
  end;
 Form1.StringGrid1.Cells[1,1]:=FloatToStrF(y_1[1,1],ffExponent,4,6);
 Form1.StringGrid1.Cells[1,0]:='X='+FloatToStr(a);
 Form1.StringGrid1.Cells[1,2]:=FloatToStrF(y_1[2,1],ffExponent,4,6);
end;

procedure Adams(a,b,h:real;NewValues:TMas;FMas:TFMas);
var
 i,n,j:integer;
 y_1:TFunZnach;
 x:real;
begin
 Form1.StringGrid1.Cells[0,1]:='Y1';
 Form1.StringGrid1.Cells[0,2]:='Y2';
 x:=a;
 n:=Round((b-a)/h);
 y_1[1,1]:=NewValues[1];
 y_1[2,1]:=NewValues[2];
```

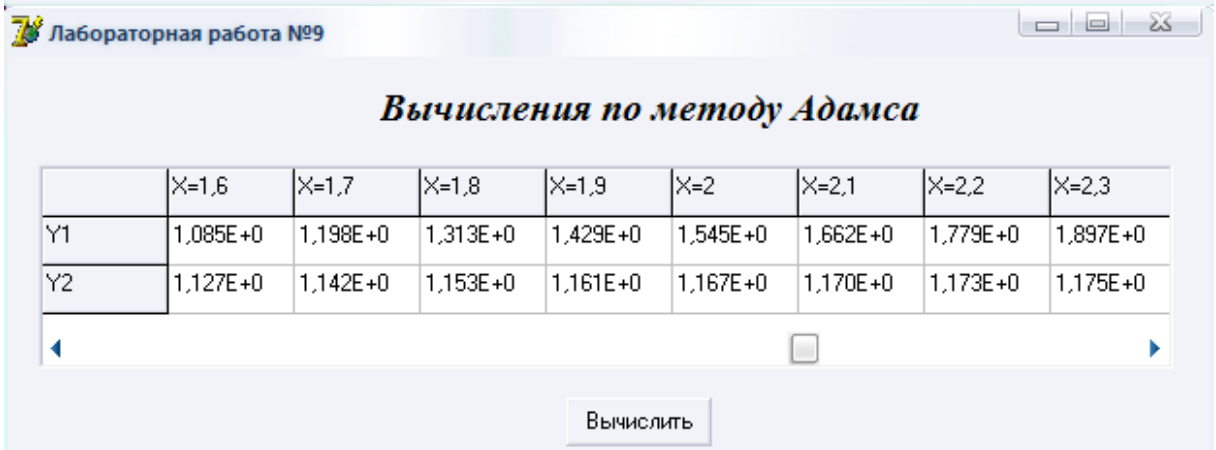
```
 Runge_Kut(a,h,x,y_1,FMas);
 Form1.StringGrid1.ColCount:=n+1;
 for i:=5 to n do
  begin
   for j:=1 to 2 do
    NewValues[j]:=y_1[j,4]+h/24*(55*FMas[j](x,y_1[1,4],y_1[2,4])-59*FMas[j](x-
h,y_1[1,3],y_1[2,3])+37*FMas[j](x-2*h,y_1[1,2],y_1[2,2])-9*FMas[j](x-
3*h,y_1[1,1],y_1[2,1]));
   for j:=2 to 4 do
    begin
     y_1[1,j-1]:=y_1[1,j];
     y_1[2,j-1]:=y_1[2,j];
    end;
   y_1[1,4]:=NewValues[1];
   y_1[2,4]:=NewValues[2];
   x:=x+h;
   Form1.StringGrid1.Cells[i,0]:='X='+FloatToStr(x);
   Form1.StringGrid1.Cells[i,1]:=FloatToStrF(NewValues[1],ffExponent,4,6);
   Form1.StringGrid1.Cells[i,2]:=FloatToStrF(NewValues[2],ffExponent,4,6);
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
 NewValues:TMas;
 FMas:TFMas;
begin
 NewValues[1]:=0;
 NewValues[2]:=0;
 FMas[1]:=f1;
 FMas[2]:=f2;
 Adams(0,3,0.1,NewValues,FMas);
end;
end.
```

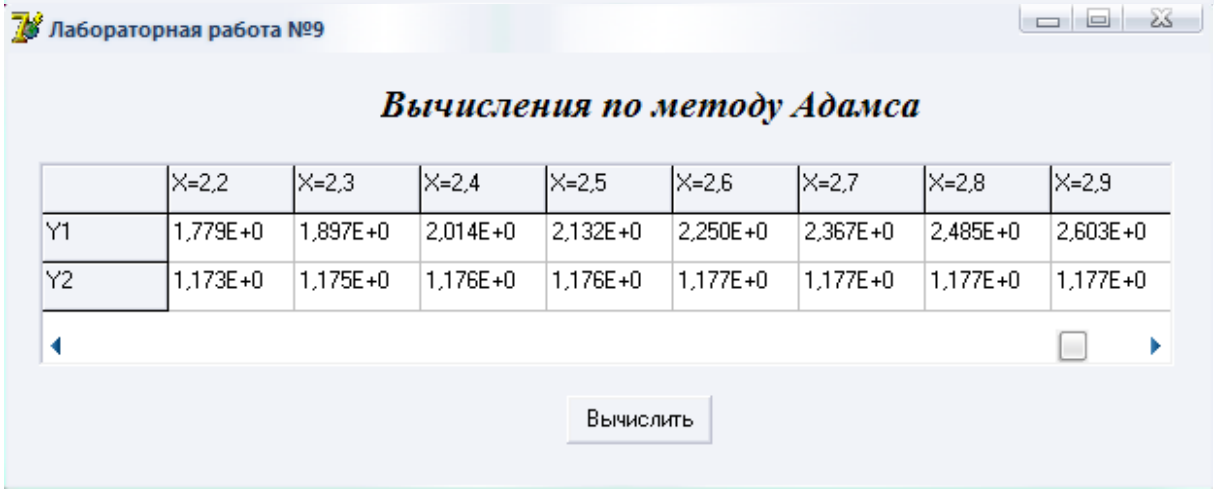
Вычисления по программе привели к следующим результатам:



	X=0	X=0,1	X=0,2	X=0,3	X=0,4	X=0,5	X=0,6	X=0,7
Y1	0,000E+0	0,000E+0	1,048E-2	3,139E-2	6,600E-2	1,105E-1	1,643E-1	2,272E-1
Y2	0,000E+0	9,967E-2	1,988E-1	2,973E-1	3,956E-1	4,917E-1	5,844E-1	6,724E-1


Лабораторная работа №9
Вычисления по методу Адамса
X=0,8 X=0,9 X=1 X=1,1 X=1,2 X=1,3 X=1,4 X=1,5
Y1 2,986E-1 3,778E-1 4,642E-1 5,568E-1 6,549E-1 7,575E-1 8,639E-1 9,732E-1
Y2 7,545E-1 8,295E-1 8,965E-1 9,550E-1 1,005E+0 1,046E+0 1,080E+0 1,106E+0
Вычислить


Лабораторная работа №9
Вычисления по методу Адамса
X=1,6 X=1,7 X=1,8 X=1,9 X=2 X=2,1 X=2,2 X=2,3
Y1 1,085E+0 1,198E+0 1,313E+0 1,429E+0 1,545E+0 1,662E+0 1,779E+0 1,897E+0
Y2 1,127E+0 1,142E+0 1,153E+0 1,161E+0 1,167E+0 1,170E+0 1,173E+0 1,175E+0
Вычислить


Лабораторная работа №9
Вычисления по методу Адамса
X=2,2 X=2,3 X=2,4 X=2,5 X=2,6 X=2,7 X=2,8 X=2,9
Y1 1,779E+0 1,897E+0 2,014E+0 2,132E+0 2,250E+0 2,367E+0 2,485E+0 2,603E+0
Y2 1,173E+0 1,175E+0 1,176E+0 1,176E+0 1,177E+0 1,177E+0 1,177E+0 1,177E+0
Вычислить

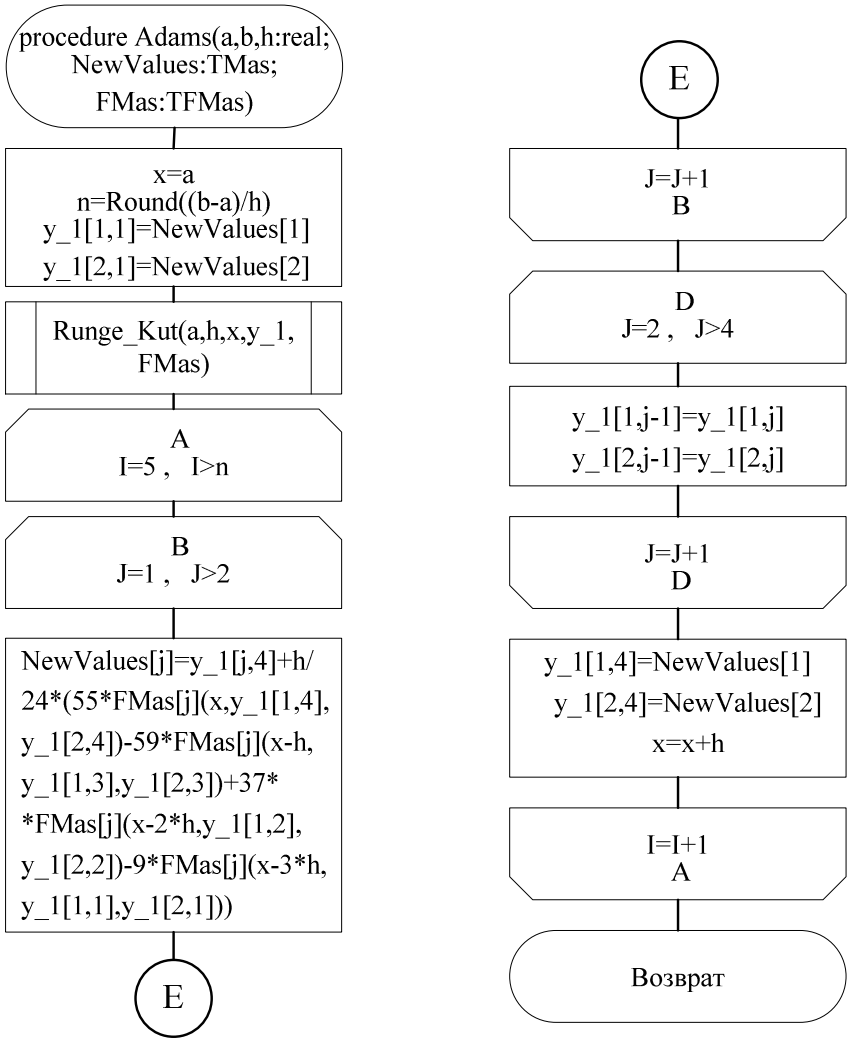


Рисунок 33 – Схема алгоритма явного метода Адамса

Варианты заданий

Таблица 7

№ варианта	$F_1(x,y_1,y_2)$	$F_2(x,y_1,y_2)$	$y_1(a)$	$y_2(a)$	a	b
1	2	3	4	5	6	7
1	$arctg(\ 1/\sqrt{1+y_1^2+y_2^2}\ )$	$sin(\ y_1 y_2\ )$	1	0	-1	1
2	$arctg(\ x^2+y_2^2\ )$	$sin(\ x+y_1\ )$	0,5	1,5	0	2
3	$(y_1+x)/(y_1^2+y_2^2)$	$cos(\ y_1+xy_2\ )$	-1	1	0	4
4	$e^{y_1 y_2}$	xy_1y_2	1	0	0	5
5	$x/\sqrt{1+x^2+y_2^2}$	$y_2/\sqrt{1+x^2+y_1^2}$	0,2	0	-1	1

Продолжение таблицы 7

№ варианта	$F_1(x,y_1,y_2)$	$F_2(x,y_1,y_2)$	$y_1(a)$	$y_2(a)$	a	b
1	2	3	4	5	6	7
6	$sin(x^2+y_2^2)$	$cos(xy_1)$	0	0	0	4
7	$\sin y_2$	$\cos y_1$	0,5	-0,5	1	3
8	$x\cos(y_1+y_2)$	$sin(y_1-y_2)$	-0,6	2	2	5
9	$\sin y_1 \cos^3 y_2$	$\cos y_1 \cos y_2)$	0	0	-1	3
10	$sin(y_1y_2)$	$cos(xy_1y_2)$	0,5	1,2	0	2
11	$arctg(1/(1+y_1^2+y_2^2))$	$sin(y_1y_2)$	1	1	1	4
12	$y_2/\sqrt{1+x^2+y_1^2}$	$y_2\cos x-\sin 2x$	0,8	3,5	2	3
13	$(y_1+x)/(y_1^2+y_2^2)$	$cos(y_1+xy_2)$	1	-1	2	4
14	$sin(xy_2)$	$cos(xy_1y_2)$	0	-3	2	5
15	$cos(y_1y_2)$	$sin(y_1+y_2)$	0	0	0	2
16	$y_2 ln\, x$	$sin(x-y_2)$	-2	-1	1	4
17	$\cos y_1 \cos y_2)$	$y_2/\sqrt{1+x^2+y_1^2}$	0	1	-1	1
18	$arctg(1/(1+y_1^2+y_2^2))$	$\sin y_1$	0	0	-2	1
19	$y1+y2$	$cos(xy_1)$	0	0	0	4
20	$x\cos(y_1+y_2)$	Y_1y_2	-1	1	0	4
21	$cos(y_1y_2)$	$y_2\cos x-\sin 2x$	-1	5	2	4
22	$y_2/\sqrt{1+x^2+y_1^2}$	$sin(y_1-y_2)$	1	1	0	3
23	$cos(x+y_2)$	$sin(x-y_2)$	0,7	-0,5	0	4
24	$x\cos(y_1+y_2)$	$y_2/\sqrt{1+x^2+y_1^2}$	0	0	0	2
25	$x/\sqrt{1+x^2+y_2^2}$	$y_2/\sqrt{1+x^2+y_1^2}$	0,2	0	0	3
26	$\sin y_2$	$cos(y_1y_2)$	1	-1	0	1
27	$y_2/\sqrt{1+x^2+y_1^2}$	$sin(y_1y_2)$	-2	3	-1	1
28	$sin(xy_2)$	$x+cos(xy_1)$	0	0	0	2
29	$y_2 ln\, x$	y_1y_2	0	0	-5	0
30	$sin(y_1y_2)$	$cos(xy_1y_2)$	-1	2	0	2

Лабораторная работа №10
Решение жестких систем ОДУ

Метод Гира.

Procedure Gir (*a,b:Real; n:Integer; x:Real;*
Var y_1:TMas; FMas:TFMas).

Входные параметры: *a* – начало отрезка, *b* – конец отрезка интегрирования; *n* – количество шагов приращения аргумента *X*; *x* – начальное значение *X*; *y_1* – массив, содержащий приближенные значения функций в точке *a*; *Fmas* – массив функций системы.

Выходные параметры: решение системы ОДУ на отрезке [*a,b*].

(Процедура использует процедуру *Runge_Kut* (*a,b: Real; n: Integer; x: Real; Var y_1: TMas; FMas: TFMas*), которая вычисляет начальные значения *Y* методом Рунге-Кутта для метода Гира. Процедура вызывается 4 раза для получения 4 начальных точек, по которым будет вычислено искомое значение *Y* методом Гира. Все параметры формируются процедурой *Gir*.

Входные параметры: *a* – начало отрезка, *b* – конец отрезка интегрирования; *n* – количество шагов приращения аргумента *X*; *x* – начальное значение *X*; *y_1* – массив, содержащий начальное значение функций в точке *a*; *Fmas* – массив функций системы.

Выходные параметры: *y_1* – массив, содержащий приближенные значения функций в точках отрезка.

Определим следующие типы:

```
Type
 TF=Function (x,y1,y2:Real):Real;
 TFMas=array [1..2] of TF;
 TMas=array [1..2] of Real;
```

Пример задания массива *TFMas*

```
Function F1 (x,y1,y2:Real):Real;
Begin
Result:=StrToInt(Form1.StringGrid2.Cells[1,1])*y1+
```

```
+ StrToInt(Form1.StringGrid2.Cells[2,1])*y2;
End;
Function F2(x,y1,y2:Real):Real;
Begin
Result:=StrToInt(Form1.StringGrid2.Cells[1,2])*y1+
        +StrToInt(Form1.StringGrid2.Cells[2,2])*y2;
End;

Var Fmas:TFMas;

//..
Fmas[1]:=F1;
Fmas[2]:=F2;
//..
```

Перед вычислением выполнить:

1. Задать интервал интегрирования (значения отрезка $[a,b]$, переменная x получит значение начала отрезка интегрирования a;
2. Задать количество шагов интегрирования (переменная n).

Схема алгоритма показана на рисунке 34.

Пример. Решить задачу Коши для системы:

$$\begin{cases} y_1' = -11y_1 + 9y_2 \\ y_2' = 9y_1 - 11y_2 \end{cases}$$

при начальных условиях $y1(0) = 1$, $y2(0) = 0$.

Текст программы:

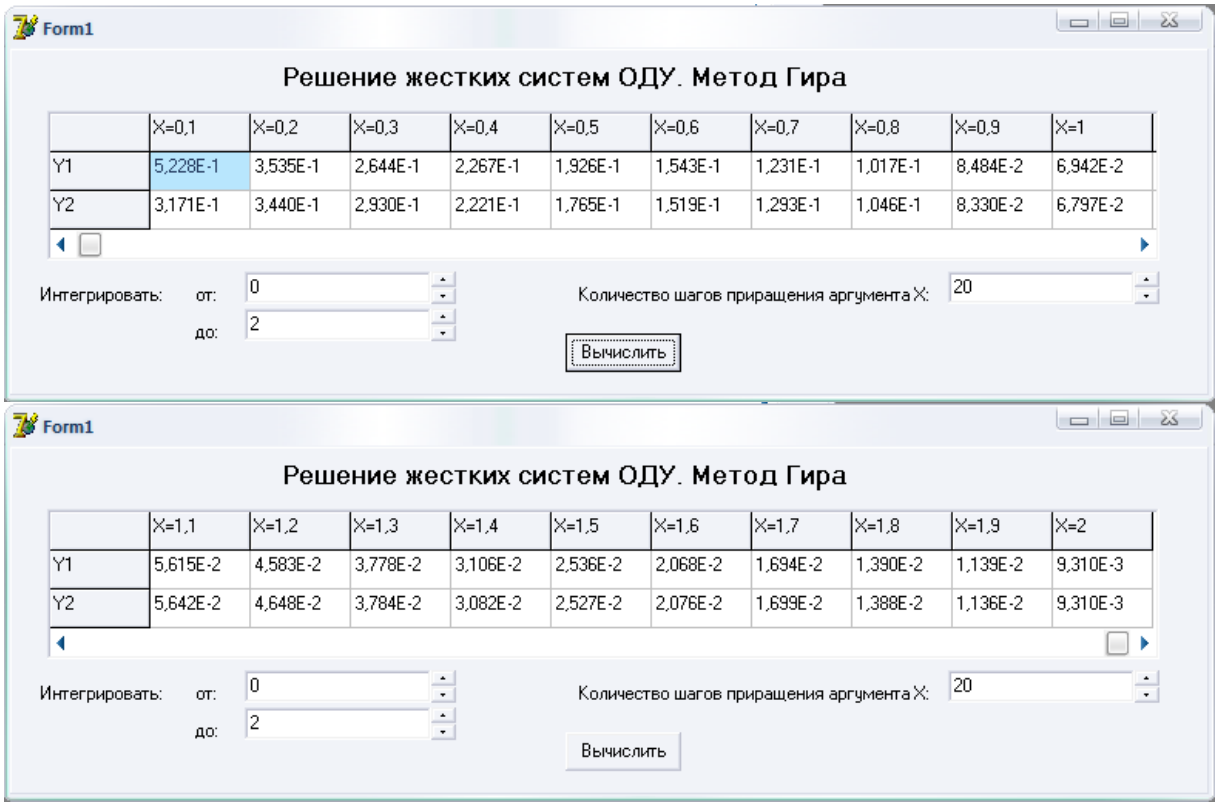
```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, StdCtrls, ExtCtrls, ComCtrls;
type
  TForm1 = class(TForm)
```

```
    Panel1: TPanel;
    Label1: TLabel;
    StringGrid1: TStringGrid;
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    UpDown1: TUpDown;
    UpDown2: TUpDown;
    Label5: TLabel;
    UpDown3: TUpDown;
    procedure Button1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;

implementation
{$R *.dfm}

Type
 TF=Function (x,y1,y2:Real):Real;
 TFMas=array [1..2] of TF;
 TMas=array [1..2] of Real;
Function F1 (x,y1,y2:Real):Real;
Begin
  Result:=-11*y1+9*y2;
End;
Function F2(x,y1,y2:Real):Real;
Begin
  Result:=9*y1-11*y2;
End;

Procedure Runge_Kut(a,b:Real;n:Integer;x:Real;Var y_1:TMas;FMas:TFMas);
Var
  j:Integer;
  k:array [1..4] of Real;
  h:Real;
  New:TMas;
Begin
  n:=n;
  h:=(b-a)/n;
  New[1]:=y_1[1];
  New[2]:=y_1[2];
  For j:=1 to 2 do
```

```
    Begin
    k[1]:=h*FMas[j](x,New[1],New[2]);
    k[2]:=h*FMas[j](x+h/2,New[1]+1/2*k[1],New[2]+1/2*k[1]);
    k[3]:=h*FMas[j](x+h/2,New[1]+1/2*k[2],New[2]+1/2*k[2]);
    k[4]:=h*FMas[j](x+h,New[1]+k[3],New[2]+k[3]);
    y_1[j]:=y_1[j]+h/6*(k[1]+2*k[2]+2*k[3]+k[4]);
    End;
End;

Procedure Gir(a,b:Real;n:Integer;x:Real;Var y_1:TMas;FMas:TFMas);
Var
  i,j,z:Integer;
  h,c1,c2:Real;
  Fun:array [1..2,1..4] of Real;
Begin
  z:=0;
  Form1.StringGrid1.ColCount:=n+1;
  Form1.StringGrid1.Cells[0,1]:='Y1';
  Form1.StringGrid1.Cells[0,2]:='Y2';
  h:=(b-a)/n;
  For i:=1 to 4 do
    Begin
    Runge_Kut(a,b,n,x,y_1,FMas);
    x:=x+h;
    For j:=1 to 2 do Fun[j,i]:=y_1[j];
    End;
  x:=a+0.1;
  For i:=1 to n do
    Begin
    c1:=(-48*Fun[1,4]+36*Fun[1,3]-16*Fun[1,2]+3*Fun[1,1])/1.2;
    c2:=(-48*Fun[2,4]+36*Fun[2,3]-16*Fun[2,2]+3*Fun[2,1])/1.2;
    y_1[1]:=(-9*c2-(25/1.2+11)*c1)/((25/1.2+11)*(25/1.2+11)-9*9);
    y_1[2]:=(9*y_1[1]-c2)/(25/1.2+11);
          // Данные формулы получены из решения системы линейных уравнений
          // по формуле Гира выражением одной переменной через другую
    Inc(z);
    For j:=1 to 2 do
      Begin
      Fun[j,1]:=Fun[j,2];Fun[j,2]:=Fun[j,3];Fun[j,3]:=Fun[j,4];Fun[j,4]:=y_1[j];
      End;
    Form1.StringGrid1.Cells[z,0]:='X='+FloatToStr(x);
    Form1.StringGrid1.Cells[z,1]:=FloatToStrF(y_1[1],ffExponent,4,6);
    Form1.StringGrid1.Cells[z,2]:=FloatToStrF(y_1[2],ffExponent,4,6);
    x:=x+h;
    End;
End;

procedure TForm1.Button1Click(Sender: TObject);
var
 FMas:TFMas;
 y_1:TMas;
begin
```

```
 y_1[1]:=1;
 y_1[2]:=0;
 FMas[1]:=f1;
 FMas[2]:=f2;
 Gir(StrToInt(Edit1.Text),StrToInt(Edit2.Text),StrToInt(Edit3.Text),0,y_1,FMas);
end;

procedure TForm1.FormShow(Sender: TObject);
begin
  Button1.SetFocus;
end;

end.
```

Вычисления по программе привели к следующим результатам:



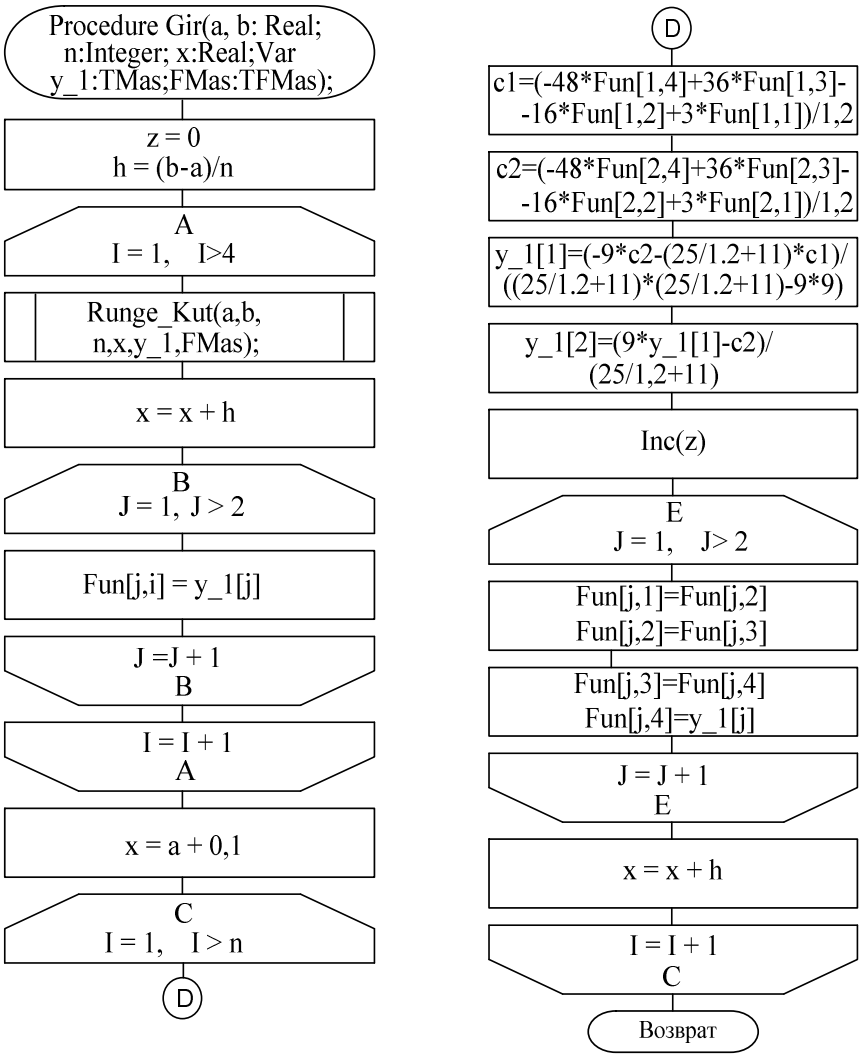


Рисунок 34 – Схема алгоритма метода Гира

Варианты заданий для решения жестких систем ОДУ приведены в таблице 8.

Метод Ракитского (матричной экспоненты).

Procedure MetRak (n: Word; Var H: Real; A: MasReal;
var Y: vector; Var Q: MasReal; Ind, nts: Integer);

Входные параметры:

n – размерность системы;

A – матрица коэффициентов системы типа *Mas Real=array* [1..2,1..2]*of real;*

H – шаг между точками интегрирования;

Y – массив размерности *n* содержит решение системы на предыдущем шаге;

Q – матрица размерности *n***n* содержит матричную экспоненту;

ind – режим работы процедуры: 0-вычисление матричной экспоненты, 1-вычисление очередного решения;

nts – номер шага.

Выходные параметры:

Y – массив размерности *n* содержит решение системы на текущем шаге;

Q – матрица размерности *n***n* содержит матричную экспоненту;

H – шаг между точками интегрирования.

(Процедура использует следующие процедуры:

procedure MulMat (*n: integer;a,b: masReal; var c: masReal*); – перемножение матриц *A* и *B* размерности *n***n*, *C*-выходная матрица;

procedure MatnaVec (*n: integer;a: masReal; b: vector; var c: vector*); – умножение матрицы *A* на массив(вектор)*B*, *C*-выходной массив;

procedure MulConst(*n: word; H:real; var B: masReal*); – умножение матрицы *A* на число *H*, *B* - выходная матрица.)

Схема алгоритма приведена на рисунке 35.

Пример. Решить задачу Коши для системы

$$\begin{cases} y_1' = -11y_1 + 9y_2 \\ y_2' = 9y_1 - 11y_2 \end{cases}$$

при начальных условиях $y_1(0)=1$, $y_2(0)=0$.

Текст программы:

```
uses crt;
Type
  masReal=array[1..2,1..2]of real;
  vector=array[1..2]of real;
var
  a1,f1,q1: masReal;
  n,nts,ind:integer;
  X,H:real;
  Y0:vector;
```

```
{перемножение матриц размерности n*n A и B, C-выходная матрица}
procedure MulMat(n:integer;a,b:masReal;var c:masReal);
var  i,j,k:byte;
begin
 for i:=1 to n do
   for j:=1 to n do
     c[i,j]:=0;
 for i:=1 to n do
   for j:=1 to n do
     for k:=1 to n do
       c[i,j]:=c[i,j]+a[i,k]*b[k,j];
end;

{умножение матрицы A на массив(вектор)B, C-выходной массив }
procedure MatnaVec(n:integer;a:masReal;b:vector ;var c:vector);
var  i,j,k:byte;
begin
 for i:=1 to n do c[i]:=0;
 for i:=1 to n do
   for j:=1 to n do
     c[i]:=c[i]+a[j,i]*b[j];
end;

{умножение матрицы A на число H, B-выходной массив }
procedure MulConst(n:word;H:real;var B:masReal );
var i,j:integer;
begin
for i:=1 to n do
  for j:=1 to n do
        b[i,j]:=b[i,j]*h
end;

{процедура реализующая метод Ракитского }
Procedure MetRak(n:Word;Var H:Real;A:MasReal;var Y:vector;Var
Q:MasReal;Ind,nts:Integer);
{n  -порядок системы
A  -матрица коэффициентов системы;
Y  -массив размерности n содержит решение системы
 Q  -матрица размерности n*n содержит матричную экспоненту
 ind-режим работы процедуры: 0-вычисление матричной экспоненты
                              1-вычисление очередного решения
 nts-номер шага
}
Var
   RabMas1: MasReal; {рабочая матрица }
   y0,y1,RabMas2: vector;{рабочие массивы}
   S  : Real;
   i,j,k  : Byte;
BEGIn
 IF Ind=0 Then
 Begin
    {построение матрицы exp(Ah)}
```

```
{ вычисление нормы матрицы A}
     S:=0;
     For i := 1 To n Do
      For j := 1 To n Do
        S:=S+A[i,j]*A[i,j];
     S:=Sqrt(S);
     H:=0.1/S;
     nts := 0;
     for i:=1 to n do
       for j:=1 to n do
              Q[i,j]:=0;
     mulconst(n,H,A);
     for k:=5 downto 1 do
      begin
       mulconst(n,1/k,Q);
{Q+E}
       for i:=1 to n do
             Q[i,i]:=Q[i,i]+1;
       mulmat(n,Q,A,Rabmas1);
       Q:=Rabmas1;
      end;
 {Q+E}
       for i:=1 to n do
            Q[i,i]:=Q[i,i]+1;
    {построение матрицы exp(AH)}
    for i:=1 to n do
      begin
       for j:=1 to n do y0[j]:=0;
       y0[i]:=1;
       for j:=1 to round(0.1/H) do
         begin
           MatnaVec(n,Q,Y0,y1);
           y0:=y1;
         end;
       for j:=1 to n do Rabmas1[j,i]:=y0[j];
      end;
      Q:=Rabmas1;
  End;
  {вычисление решения на очередном шаге}
 IF nts > 0 Then
   Begin
    MatnaVec(n,Q,Y,RabMas2);
    y:=RabMas2;
   End;
End;

begin
clrscr;

n:=2;
{ввод матрицы системы}
a1[1,1]:=-11; a1[1,2]:=  9;
```

```
a1[2,1]:=  9; a1[2,2]:=-11;
{ввод начального приближения}
Y0[1]:=1; Y0[2]:=0;
x:=0;
Writeln('  x  | ',' Y1       |  ', '   Y2 ');
{вычисление матричной экспонеты}
MetRak(n,H,A1,Y0,Q1,0,nts);
for nts:=1 to 21 do
 begin
  {вычисление решения на данном шаге X}
   MetRak(n,H,A1,Y0,Q1,1,nts);
   Writeln(x:2:2,' | ',Y0[1]:0,' |  ', Y0[2]:0);
   x:=x+0.1; end;end.
```

Вычисления по программе привели к следующим результатам:

x	*Y*1	*Y*2
0.00	4.8587E-0001	3.3448E-0001
0.10	3.4795E-0001	3.2503E-0001
0.20	2.7778E-0001	2.7431E-0001
0.30	2.2671E-0001	2.2619E-0001
0.40	1.8581E-0001	1.8573E-0001
0.50	1.5240E-0001	1.5239E-0001
0.60	1.2502E-0001	1.2502E-0001
0.70	1.0256E-0001	1.0256E-0001
0.80	8.4139E-0002	8.4139E-0002
0.90	6.9024E-0002	6.9024E-0002
1.00	5.6625E-0002	5.6625E-0002
1.10	4.6452E-0002	4.6452E-0002
1.20	3.8108E-0002	3.8108E-0002
1.30	3.1262E-0002	3.1262E-0002
1.40	2.5646E-0002	2.5646E-0002
1.50	2.1039E-0002	2.1039E-0002
1.60	1.7259E-0002	1.7259E-0002
1.70	1.4159E-0002	1.4159E-0002
1.80	1.1615E-0002	1.1615E-0002
1.90	9.5288E-0003	9.5288E-0003
2.00	7.8170E-0003	7.8170E-0003

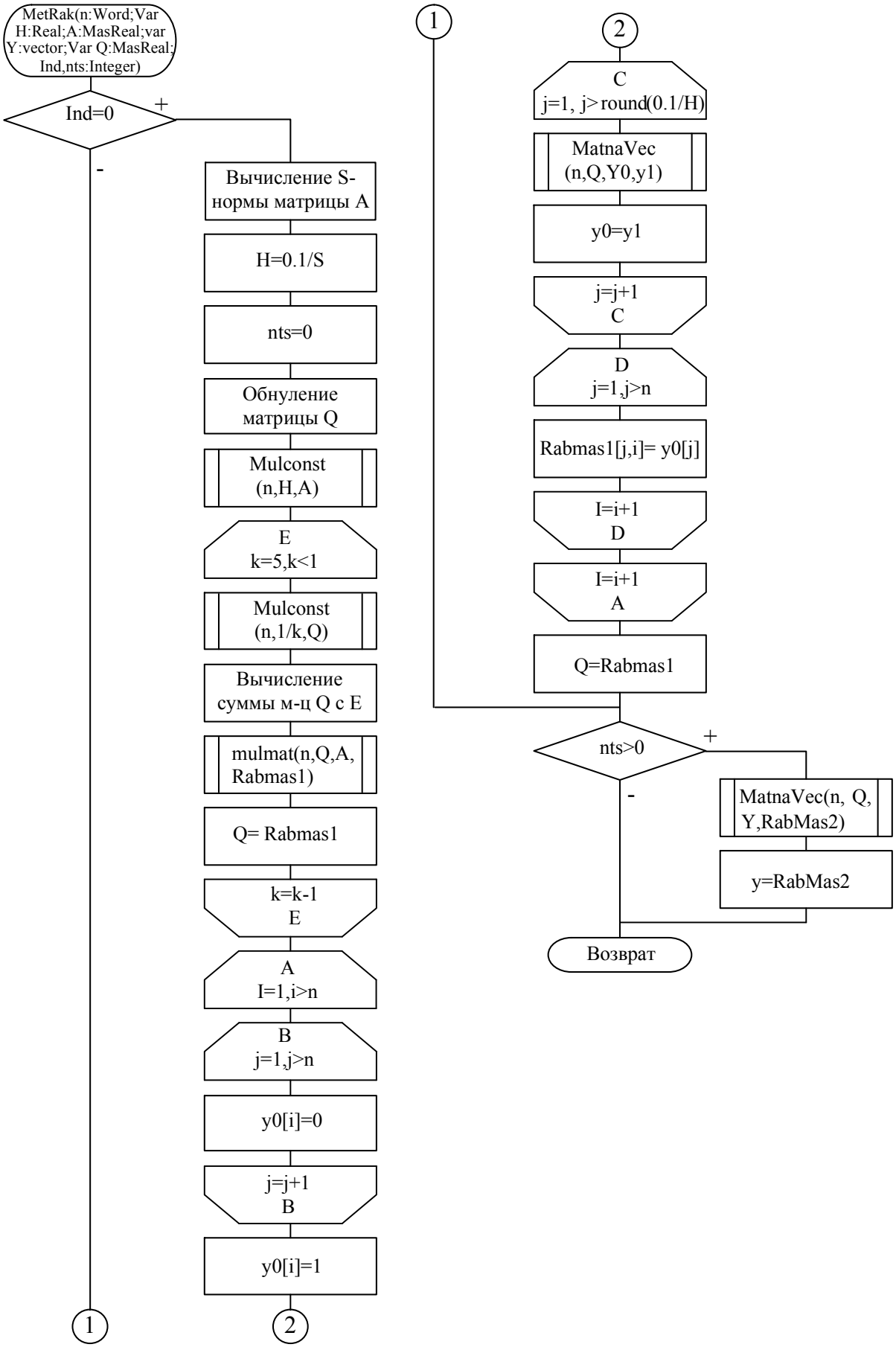


Рисунок 35 – Схема алгоритма метода Ракитского

Варианты заданий

Таблица 8

№ вари-анта	Матрица системы				Вектор начальных условий
1	2				3
1	-.18000E + 01 -.51350E + 02 .12175E + 02 .51650E + 02	.23200E+02 -.41500E+02 -.25000E+00 .16800E+02	.97000E + 01 -.40000E + 02 .50000E + 01 .30300E + 02	.83500E + 01 -.31350E + 02 .21750E + 01 .21500E + 02	.8415E+00 .5403E+00 -.4161E+00 .9093E+00
2	.22600E + 02 -.26800E + 02 -.46000E + 01 .22000E + 01	.47600E+02 -.22000E+02 -.17000E+02 -.27600E+02	.29600E + 02 -.20000E + 02 -.10000E + 02 -.96000E + 01	.27800E + 02 -.16800E + 02 -.96000E + 01 -.13OOOE+02	.9093E+00 -.4161E+00 -.6536E+00 -.7568E+00
3	.37000E + 02 -.18917E + 02 -.13042E + 02 -.20583E + 02	.65333E+02 -.15833E+02 -.27083E+02 -.52000E+02	.42833E + 02 -.13333E + 02 -.18333E + 02 -.29500E + 02	.40583E + 02 -.12250E + 02 -.16375E + 02 -.30833E + 02	.1411E+00 -.9900E+00 .9602E+00 -.2794E+00
4	.48900E + 02 -.15200E + 02 -.19400E + 02 -.36700E + 02	.81400E+02 -.13000E+02 -.35500E+02 -.71400E+02	.54400E + 02 -.10000E + 02 -.25000E + 02 -.44400E + 02	.51700E + 02 -.10200E + 02 -.21900E + 02 -.44500E + 02	-.7568E+00 -.6536E+00 -.1455E+00 .9894E+00
5	.59800E + 02 .13150E + 02 -.24925E + 02 -.50150E + 02	.96800E+02 -.11500E+02 -.43250E+02 -.88800E+02	.65300E + 02 -.80000E + 01 -.31000E + 02 -.57300E + 02	.62150E + 02 -.91500E + 01 -.26925E + 02 -.56500E + 02	-.9589E+00 .2837E+00 -.8391E+00 -.5440E+00
6	.70200E + 02 -.11933E + 02 -.30033E + 02 -.62267E + 02	.11187E+03 -.10667E+02 -.50667E+02 -.10520E+03	.75867E + 02 -.66667E + 01 -.36667E + 02 -.69200E + 02	.72267E + 02 -.86000E + 01 -.31700E + 02 -.67667E + 02	.9602E+00 .9602E+00 .8439E+00 -.5366E+00
7	.80314E + 02 .11193E + 02 -.34904E + 02 -.73621E + 02	.12674E+03 -.10214E+02 -.57893E+02 -.12103E+03	.86243E + 02 -.57143E + 01 .42143E + 02 -.80529E + 02	.82193E + 02 -.83357E + 01 -.36332E + 02 -.78357E + 02	.6570E+00 .7539E+00 .1367E+00 .9906E+00
8	.90250E + 02 -.10750E + 02 -.39625E + 02 -.84500E + 02	.14150E+03 -.10000E+02 -.65000E+02 -.13650E+03	.96500E + 02 -.50000E + 01 -.47500E + 02 -.91500E + 02	.92000E + 02 -.82500E + 01 -.40875E + 02 -.88750E + 02	.9894E+00 -.1455E+00 -.9577E+00 -.2879E+00
9	.10007E + 03 -.10506E + 02 -.44247E + 02 -.95061E + 02	.15618E+03 -.99444E+01 -.72028E+02 -.15173E+03	.10668E + 03 -.44444E + 01 -.52778E + 02 -.10223E + 03	.10173E + 03 -.82833E + 01 -.45358E + 02 -.98944E + 02	.4121E+00 -.9111E+00 .6603E+00 -.7510E+00

Продолжение таблицы 8

1	2	3	4	5	6
10	.10980E + 03 -.10400E + 02 -.48800E + 02 -.10540E + 03	.17080E+03 -.10000E+02 -.79000E+02 -.16680E+03	.11680E + 03 -.40000E + 01 -.58000E + 02 -.11280E + 03	.11140E + 03 -.84000E + 01 -.49800E + 02 -.10900E + 03	-.5440E+00 -.8391E+00 .4081E+00 .9129E+00
11	.11947E + 03 -.10395E + 02 -.53302E + 02 -.11558E + 03	.18538E+03 -.10136E+02 -.85932E+02 -.18175E+03	.12688E + 03 -.36364E + 01 -.63182E + 02 -.12325E + 03	.12103E + 03 -.85773E + 01 -.54211E + 02 -.11895E + 03	-.1000E+01 .4426E-02 -.1000E+01 -.8851E-02
12	.12910E + 03 -.10467E + 02 -.57767E + 02 -.12563E + 03	.19993E+03 -.10333E+02 -.92833E+02 -.19660E+03	.13693E + 03 -.33333E + 01 -.68333E + 02 -.13360E + 03	.13063E + 03 -.88000E + 01 -.58600E + 02 -.12883E + 03	-.5366E+00 .8439E+00 .4242E+00 -.9056E+00
13	.13869E + 03 -.10596E + 02 -.62202E + 02 -.13560E + 03	.21446E+03 -.10577E+02 -.99712E+02 -.21138E+03	.14696E + 03 -.30769E + 01 -.73462E + 02 -.14388E + 03	.14021E + 03 -.90577E + 01 -.62971 E + 02 -.13865E + 03	.4202E+00 .9074E+00 .6469E+00 .7626E+00
14	.14826E + 03 -.10771E + 02 -.66614E + 02 -.14549E + 03	.22897E+03 -.10857E+02 -.10657E+03 -.22611E+03	.15697E + 03 -.28571E + 01 -.78571E + 02 -.15411E + 03	.14977E+03 -.93429E + 01 -.67329E + 02 -.14843E + 03	.9906E+00 .1367E+00 -.9626E+00 .2709E+00
15	.15780E + 03 -.10983E + 02 -.71008E + 02 -.15532E + 03	.24347E+03 -.11167E+02 -.11342E+03 -.24080E+03	.16697E + 03 -.26667E + 01 -.83667E + 02 -.16430E + 03	.15932E + 03 -.96500E + 01 -.71675E + 02 -.15817E + 03	.6503E+00 -.7597E+00 .1543E+00 -.9880E+00
16	.16732E + 03 -.11225E + 02 -.75387E + 02 -.16510E + 03	.25795E+03 -.11500E+02 -.12025E+03 -.25545E+03	.17695E + 03 -.25000E + 01 -.88750E + 02 -.17445E + 03	.16885E + 03 -.99750E + 01 -.76013E + 02 -.16788E + 03	-.2879E+00 -.9577E+00 .8342E+00 .5514E+00
17	.17684E + 03 -.11491E + 02 -.79754E + 02 -.17484E + 03	.27242E+03 -.11853E+02 -.12707E+03 -.27007E+03	.18692E + 03 -.23529E + 01 -.93824E + 02 -.18457E + 03	.17837E + 03 -.10315E + 02 -.80343E + 02 -.17756E + 03	-.9614E+00 -.2752E+00 -.8486E+00 .5291E+00
18	.18633E + 03 -.11778E + 02 -.84111E + 02 -.18456E + 03	.28689E+03 -.12222E+02 -.13389E+03 -.28467E+03	.19689E + 03 -.22222E + 01 -.98889E + 02 -.19467E + 03	.18789E + 03 -.10667E + 02 - 84667E + 02 -.18722E + 03	-.7510E+00 .6603E+00 -.1280E+00 -.9918E+00
19	.19582E + 03 -.12082E + 02 -.88459E + 02 -.19424E + 03	.30135E+03 -.12605E+02 -.14070E+03 -.29924E+03	.20685E + 03 -.21053E + 02 -.10395E + 03 -.20474E + 03	.19740E + 03 -.11029E + 02 -.88986E + 02 -.19687E + 03	.1499E+00 .9887E+00 .9551E+00 .2964E+00
20	.20530E + 03 -.12400E + 02	.31580E+03 -.13000E+02	.21680E + 03 -.20000E + 01	.20690E + 03 -.11400E + 02	.9129E+00 .4081E+00

Продолжение таблицы 8

1	2	3	4	5	6
	-.92800E + 02	-.14750E+03	-.10900E + 03	-.93300E + 02	-.6669E+00
	-.20390E + 03	-.31380E+03	-.21480E + 03	-.20650E + 03	.7451E+00
21	.21477E + 03	.33025E+03	.22675E + 03	.21640E + 03	.8367E+00
	-.12731E + 02	-.13405E+02	-.19048E + 01	-.11779E + 02	-.5477E+00
	-.97135E + 02	-.15430E+03	-.11405E + 03	-.97611E + 02	-.4000E+00
	-.21354E + 03	-.32834E+03	-.22484E + 03	-.21612E + 03	-.9165E+00
22	.22424E + 03	.34469E+03	.23669E + 03	.22589E + 03	-.8851E-02
	-.13073E + 02	-.13818E+02	-.18182E + 01	-.12164E + 02	-.1000E+01
	-.10146E + 03	-.16109E+03	-.11909E + 03	-.10192E + 03	.9998E+00
	-.22316E + 03	-.34287E+03	-.23487E + 03	-.22573E + 03	.1770E-01
23	.23370E + 03	.35913E+03	.24663E + 03	.23538E + 03	-.8462E+00
	-.13424E + 02	-.14239E+02	-.17391E + 01	-.12554E + 02	-.5328E+00
	-.10579E + 03	-.16788E+03	-.12413E + 03	-.10622E + 03	-.4322E+00
	-.23277E + 03	-.35739E+03	-.24489E + 03	-.23533E + 03	.9018E+00
24	.24315E + 03	.37357E+03	.25657E + 03	.24487E + 03	-.9056E+00
	-.13783E + 02	-.14667E+02	-.16667E + 01	-.12950E + 02	.4242E+00
	-.11011E + 03	-.17467E+03	-.12917E + 03	-.11053E + 03	-.6401E+00
	-.24237E + 03	-.37190E+03	-.25490E + 03	-.24492E + 03	-.7683E+00
25	.25260E + 03	.38800E+03	.26650E + 03	.25435E + 03	-.1324E+00
	-.14150E + 02	-.15100E+02	-.16000E + 01	-.13350E + 02	.9912E+00
	-.11443E + 03	-.18145E+03	-.13420E + 03	-.11482E + 03	.9650E+00
	-.25195E + 03	-.38640E+03	-.26490E + 03	-.25450E + 03	-.2624E+00
26	.26205E + 03	.40243E-03	.27643E + 03	.26383E + 03	.7626E+00
	-.14523E + 02	-.15538E+02	-.15385E + 01	-.13754E + 02	.6469E+00
	-.11874E + 03	-.18823E+03	-.13923E + 03	-.11912E + 03	-.1630E+00
	-.26152E + 03	-.40089E+03	-.27489E + 03	-.26408E + 03	.9866E+00
27	.27149E + 03	.41686E+03	.28636E + 03	.27331E + 03	.9564E+00
	-.14523E + 02	-.15981E+02	-.14815E + 01	-.14161E + 02	-.2921E+00
	-.12305E + 03	-.19501E+03	-.14426E + 03	-.12342E+03	-.8293E+00
	-.27109E + 03	-.41538E+03	-.28488E + 03	-.27365E+03	-.5588E+00
28	.28093E + 03	.43129E+03	.29629E + 03	.28279E + 03	.2709E+00
	-.15286E + 02	-.16429E+02	-.14286E + 01	-.14571E + 02	-.9626E+00
	-.12736E + 03	-.20179E+03	-.14929E + 03	-.12771E + 03	.8532E+00
	-.28064E + 03	-.42986E+03	-.29486E + 03	-.28321E + 03	-.5216E+00
29	.29037E + 03	.44571E+03	.30621E + 03	.29226E + 03	-.6636E+00
	-.15674E + 02	-.16879E+02	-.13793E + 01	-.14984E + 02	-.7481E+00
	-.13166E + 03	-.20856E+03	-.15431E + 03	-.13201E + 03	.1192E+00
	-.29019E + 03	-.44433E+03	-.30483E + 03	-.29278E + 03	.9929E+00
30	.29980E + 03	.46013E+03	.31613E + 03	.30173E + 03	-.9880E+00
	-.16067E + 02	-.17333E+02	-.13333E + 0l	-.15400E + 02	.1543E+00
	-.13597E + 03	-.21533E+03	-.15933E + 03	-.13630E + 03	-.9524E+00
	-.29973E + 03	-.45880E+03	-.31480E + 03	-.30233E + 03	-.3048E+00

Лабораторная работа №11
Численное дифференцирование

Дифференцирование с помощью сплайнов

Procedure Spline3(N: integer; X,Y: mas; S0,SN: real;
Var A,B,C,D: mas);

Определяет коэффициенты сплайна, которые будут использованы процедурой *DifSpline* для получения численных значений производных в заданной точке.

Входные параметры:

N – число узлов сплайна;

X: mas – массив аргументов;

Y: mas – массив значений функции;

*S*0 – значение абсциссы начальной точки сплайна;

SN – значение абсциссы конечной точки сплайна;

Выходные параметры:

A,B,C,D: mas – массивы коэффициентов сплайна.

Массивы имеют тип *mas* = *array*[0..*n*+1] *of real*.

Схема процедуры *Spline3* показана на рисунке 36.

Текст процедуры:

```
procedure Spline3(N:integer;X,Y:mas;S0,SN:real; Var A,B,C,D:mas);
var F:mas;
   H2,H3,p:real;
   i,j:integer;
begin
   fillchar(a,sizeof(a),0);
   fillchar(b,sizeof(a),0);
   fillchar(c,sizeof(a),0);
   fillchar(d,sizeof(a),0);
   fillchar(f,sizeof(a),0);
   H2:=X[2]-X[1];
   H3:=X[3]-X[2];
   A[1]:=(2*(H2+H3))/H3;
   f[1]:=(6/h3)*(((y[3]-y[2])/h3)-((y[2]-y[1])/h2))-(h2*s0)/h3;
   for i:=4 to n-1 do begin
      h2:=x[i-1]-x[i-2];
      h3:=x[i]-x[i-1];
      a[i-2]:=(2/h3)*(h2+h3);
      b[i-2]:=h2/h3;
      f[i-2]:=(6/h3)*(((y[i]-y[i-1])/h3-((y[i-1]-y[i-2])/h2)));
```

```
    end;
    h2:=x[n-1]-x[n-2];
    h3:=x[n]-x[n-1];
    p:=2*(h2+h3);
    b[1]:=h2/p;
    f[n-2]:=(6/p)*(((y[n]-y[n-1])/h3)-((y[n-1]-y[n-2])/h2))-(h3*sn)/p;
    d[1]:=1/a[1]; c[1]:=f[1];
    For i:=2 to n-3 do begin
      d[i]:=1/(a[i]-b[i]*d[i-1]); c[i]:=f[i]-b[i]*d[i-1]*c[i-1];
    end;
    d[n-2]:=(f[n-2]-b[1]*d[n-3]*c[n-3])/(1-b[1]*d[n-3]);
    for i:=n-3 downto 1 do d[i]:=d[i]*(c[i]-d[i+1]);
    c[1]:=s0; c[n]:=sn;
    For i:=2 to n-1 do c[i]:=d[i-1];
    For i:=1 to n do begin

  a[i]:=0; b[i]:=0; d[i]:=0;
    end;
    For i:=2 to n do begin
      h2:=x[i]-x[i-1]; d[i]:=(c[i]-c[i-1])/h2;
      b[i]:=h2*c[i]/2-(Sqr(h2))*d[i]/6+(y[i]-y[i-1])/h2;
      a[i]:=y[i];
    end;
   //z:=f;
end;
```

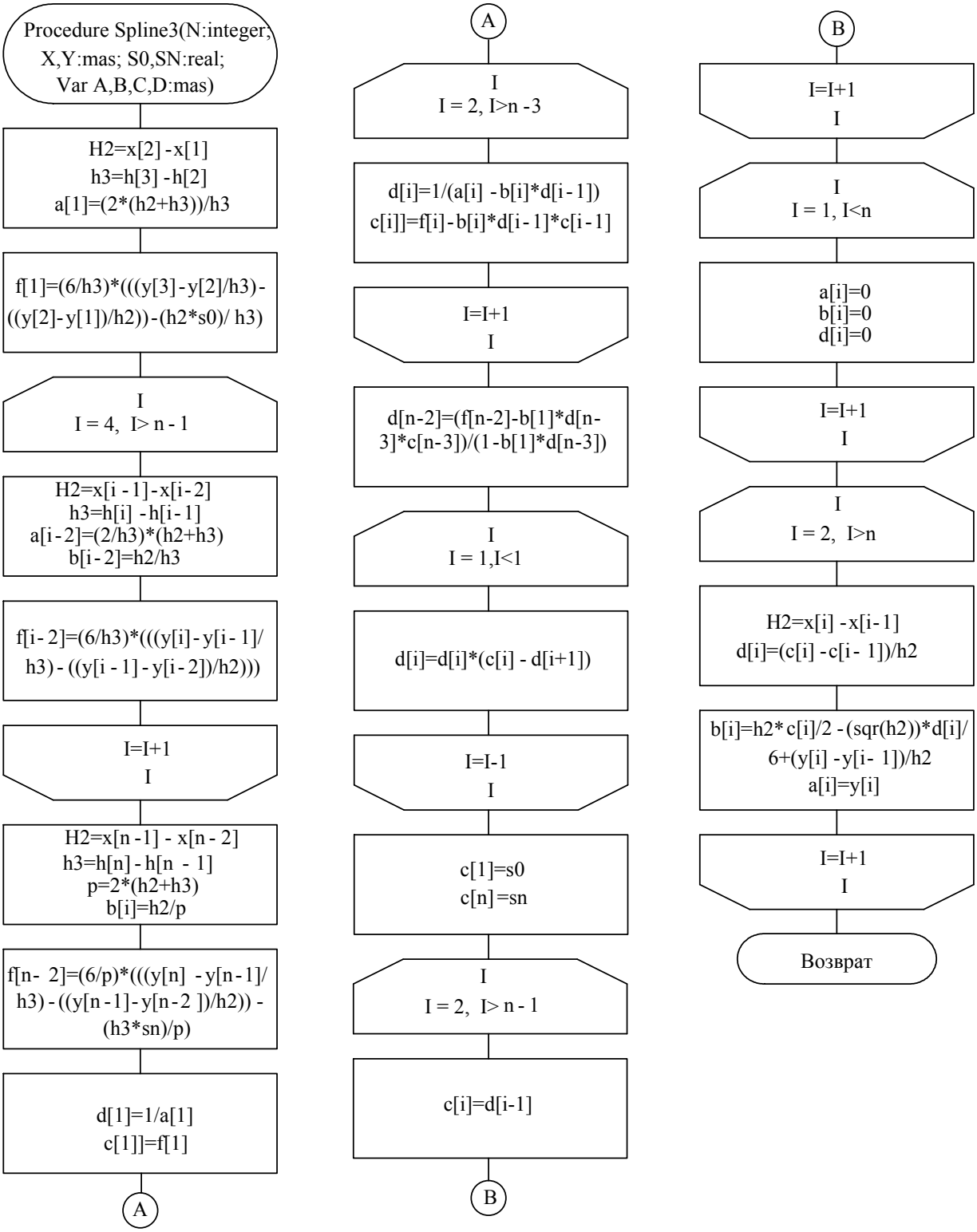


Рисунок 36 – Схема алгоритма вычисления коэффициентов сплайна

Procedure DifSline(*x,y,z:mas; xx:real; n:integer;*
*var s*1,*s*2*:real; error:boolean*);

Процедура на основе данных, полученных от процедуры *Spline3*, вычисляет значение производной в заданной точке.

Входные параметры:

n: integer – число узлов сплайна;

xx: real – абсцисса точки, для которой необходимо вычислить производную;

x: integer – массив аргументов;

y: mas – массив значений функции;

z: mas – массив коэффициентов сплайна.

Выходные параметры:

error: boolean – переменная будет иметь значение "истина", если точка *xx* находится за пределами интервала заданных значений аргументов.

Все массивы имеют тип *mas=array*[0..*n*+1] *of real*.

Схема процедуры *DifSline* показана на рисунке 37.

Текст процедуры:

```
procedure DifSline(x,y,z:mas; xx:real; n:integer; var s1,s2:real; error:boolean);
var i,j,k:integer;
    t,t1,t2,t12:real;
    hi:real;
begin
  i:=1;
  while (xx >= x[i])and(i<=n) do inc(i);
  if i>n then begin
    error:=true; exit;
  end;
  hi:=x[i]-x[i-1];
  t:=(xx-x[i-1])/hi; //(x[i]-x[i-1]);
  t1:=1-t;
  t2:=t*t;
  t12:=t1*t1;
  s1:=-6*t1*t*(y[i-1]-y[i])/hi;
  s1:=s1+t1*z[i-1]*(1-3*t)+z[i]*(3*t-2)*t;
  s2:=6*(y[i-1]-y[i])*(2*t-1)/hi;
  s2:=s2+(z[i-1])*(6*t-4)-z[i]*(2-6*t);
  s2:=s2/hi;
end;
```

Варианты заданий приведены в таблице 9.

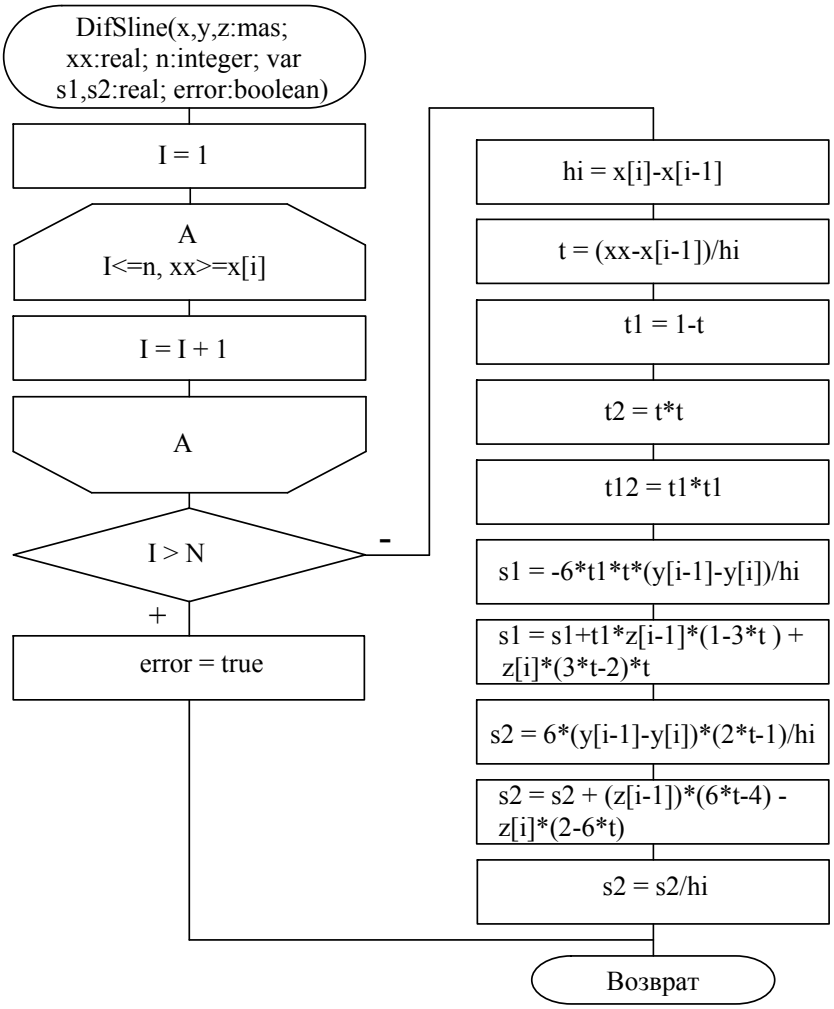
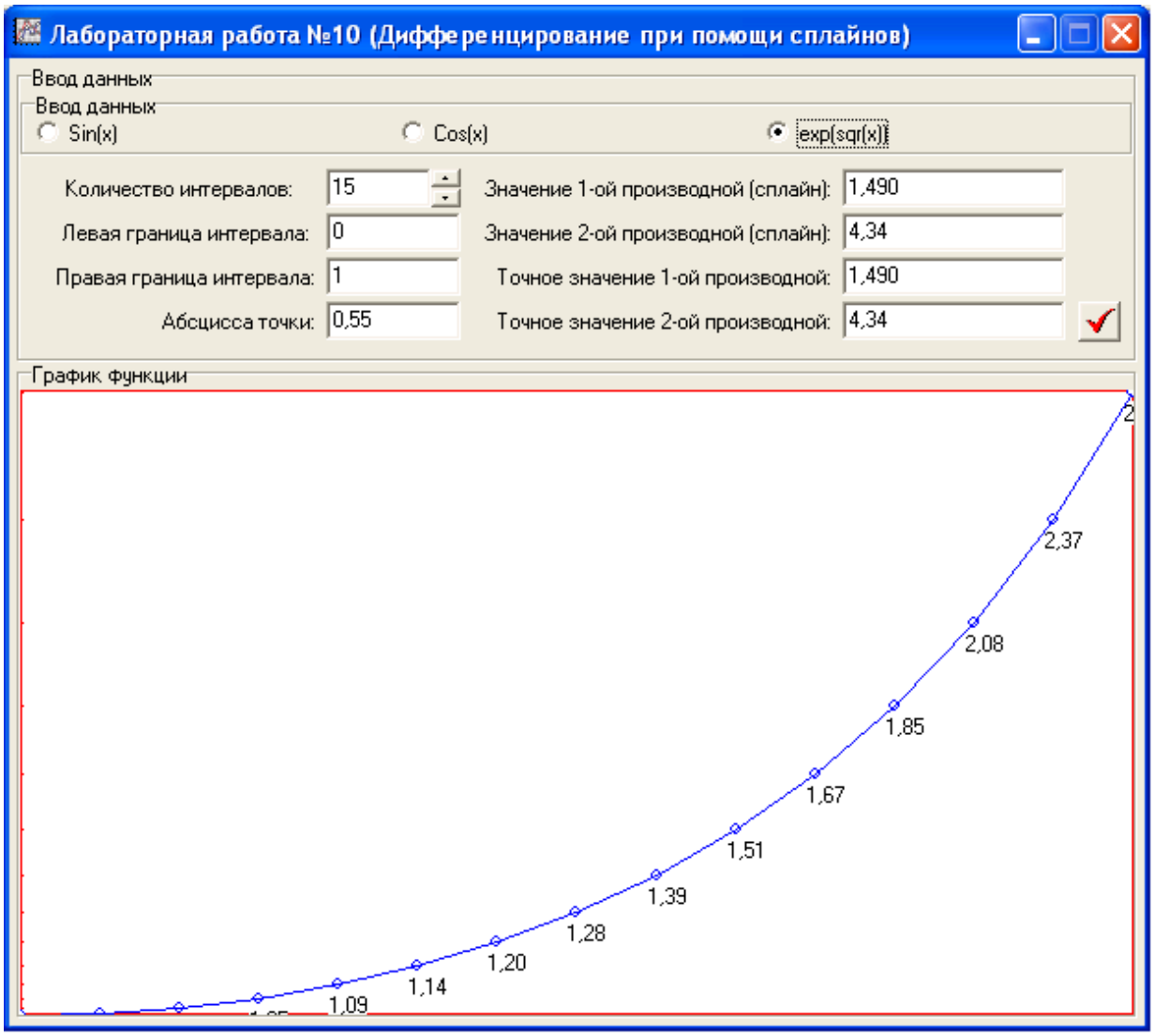


Рисунок 37 – Схема алгоритма дифференцирования с помощью сплайнов

Пример. Найти производные функции e^{x^2} в точке x=0,55. Функция задана таблично в 15 узлах.

Вычисления по программе привели к следующим результатам:



Текст программы:

```
program Lab10VM;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.Title := 'Лабораторная работа №10';
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, Grids, StdCtrls, ComCtrls, Buttons;

const n = 300;
type
  mas = array[0..n+1] of real;
  TForm1 = class(TForm)
    GroupBox1: TGroupBox;
    SpeedButton1: TSpeedButton;
    UpDown1: TUpDown;
    LabeledEdit1: TLabeledEdit;
    LabeledEdit2: TLabeledEdit;
    LabeledEdit3: TLabeledEdit;
    LabeledEdit4: TLabeledEdit;
    LabeledEdit5: TLabeledEdit;
    LabeledEdit6: TLabeledEdit;
    GroupBox2: TGroupBox;
    Image1: TImage;
    LabeledEdit7: TLabeledEdit;
    RadioGroup1: TRadioGroup;
    LabeledEdit8: TLabeledEdit;
    procedure ClearImage;
    procedure FormCreate(Sender: TObject);

 procedure SpeedButton1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  x,ys:mas;
  xc:real;
  yc,m:integer;
  dx,dt:real;
  a,b,max,min,koefx,koefy:real;
implementation

{$R *.dfm}
function f(x:real):real;
begin
  case Form1.RadioGroup1.ItemIndex of
  0: Result:=sin(x);
  1: Result:=cos(x);
  2: Result:=exp(sqr(x));
```

```
  else Result:=0;
  end;
end;

function fx(x:real):real;
begin
  case Form1.RadioGroup1.ItemIndex of
  0: Result:=cos(x);
  1: Result:=-sin(x);
  2:Result:=2*x*exp(sqr(x));
  else Result:=0;
  end;
end;

function fxx(x:real):real;
begin
  case Form1.RadioGroup1.ItemIndex of
  0: Result:=-sin(x);
  1: Result:=-cos(x);
  2: Result:=2*exp(sqr(x))*(1+2*x*x);
  else Result:=0;
  end;
end;

procedure GetParams1;
begin
 // xc:=Form1.Image1.Width div 2;
  yc:=Form1.Image1.Height div 2;
end;

procedure GetParams2;
var i:integer;
begin
  m:=strtoint(Form1.LabeledEdit1.Text);
  a:=strtofloat(Form1.LabeledEdit2.Text);
  b:=strtofloat(Form1.LabeledEdit3.Text);
if a<0 then xc:=(-a)*koefx else xc:=-a*koefx;
  dx:=(b-a)/(m-1);
  x[0]:=a-dx;   ys[0]:=f(x[0]);
  x[m+1]:=b+dx; ys[m+1]:=f(x[m+1]);
  x[1]:=a; ys[1]:=f(a);
  x[m]:=b; ys[m]:=f(b);
  for i:=2 to m do begin
    x[i]:=x[i-1]+dx;
    ys[i]:=f(x[i]);
  end;
  max:=f(a); min:=f(a);
  for i:=1 to m do begin
    if max<f(x[i]) then max:=f(x[i]);
    if min>f(x[i]) then min:=f(x[i]);
  end;
  dt:=(Form1.Image1.Width-1)/(b-a);
```

```
  koefy:=(Form1.Image1.Height-1)/(max - min);
  koefx:=(Form1.Image1.Width-1)/(b-a);
end;

procedure TForm1.ClearImage;
var i:integer;
begin
  with Image1.Canvas do begin
    Brush.Color:=clWhite;
    Pen.Color:=clRed;
    FillRect(Image1.Canvas.ClipRect);
    Rectangle(Image1.Canvas.ClipRect);
    Pen.Width:=1;
  end;
end;

procedure ClearImage2;
var x0,y0:integer;
    x1,y:integer;
    i:integer;
begin
  Form1.ClearImage;
  //y0:=round(Form1.Image1.Height-koefy*(min));
  y0:=round(Form1.Image1.Height-1-(Form1.Image1.Height-1)*(min/(min-max)));
  Form1.Image1.Canvas.MoveTo(0,y0);
  Form1.Image1.Canvas.LineTo(Form1.Image1.Width-1,y0);
  for i:=1 to m do begin
    x1:=round(i*dx*koefx);
    Form1.Image1.Canvas.MoveTo(x1,y0-2);
    Form1.Image1.Canvas.LineTo(x1,y0+2);
    //Form1.Image1.Canvas.TextOut(x1,y0-2,floattostrf(x[i],ffFixed,3,3));
  end;
  x0:=round((a*(Form1.Image1.Width-1)/(a-b)));
  Form1.Image1.Canvas.MoveTo(x0,0);
  Form1.Image1.Canvas.LineTo(x0,Form1.Image1.Height-1);
  for i:=1 to m do begin
    Form1.Image1.Canvas.MoveTo(x0-2,round((Form1.Image1.Height)-(ys[i]-
min)*koefy));
    Form1.Image1.Canvas.LineTo(x0+2,round((Form1.Image1.Height)-(ys[i]-
min)*koefy));
  end;
end;

procedure Spline3(N:integer;X,Y:mas;S0,SN:real; Var A,B,C,D:mas);
var F:mas;
    H2,H3,p:real;
    i,j:integer;
begin
    fillchar(a,sizeof(a),0);
    fillchar(b,sizeof(a),0);
    fillchar(c,sizeof(a),0);
    fillchar(d,sizeof(a),0);
```

```
    fillchar(f,sizeof(a),0);
    H2:=X[2]-X[1];
    H3:=X[3]-X[2];
    A[1]:=(2*(H2+H3))/H3;
    f[1]:=(6/h3)*(((y[3]-y[2])/h3)-((y[2]-y[1])/h2))-(h2*s0)/h3;
    for i:=4 to n-1 do begin
       h2:=x[i-1]-x[i-2];
       h3:=x[i]-x[i-1];
       a[i-2]:=(2/h3)*(h2+h3);
       b[i-2]:=h2/h3;
       f[i-2]:=(6/h3)*(((y[i]-y[i-1])/h3-((y[i-1]-y[i-2])/h2)));
    end;
    h2:=x[n-1]-x[n-2];
    h3:=x[n]-x[n-1];
    p:=2*(h2+h3);
    b[1]:=h2/p;
    f[n-2]:=(6/p)*(((y[n]-y[n-1])/h3)-((y[n-1]-y[n-2])/h2))-(h3*sn)/p;
    d[1]:=1/a[1]; c[1]:=f[1];
    For i:=2 to n-3 do begin
      d[i]:=1/(a[i]-b[i]*d[i-1]); c[i]:=f[i]-b[i]*d[i-1]*c[i-1];
    end;
    d[n-2]:=(f[n-2]-b[1]*d[n-3]*c[n-3])/(1-b[1]*d[n-3]);
for i:=n-3 downto 1 do d[i]:=d[i]*(c[i]-d[i+1]);
    c[1]:=s0; c[n]:=sn;
    For i:=2 to n-1 do c[i]:=d[i-1];
    For i:=1 to n do begin
      a[i]:=0; b[i]:=0; d[i]:=0;
    end;
    For i:=2 to n do begin
      h2:=x[i]-x[i-1]; d[i]:=(c[i]-c[i-1])/h2;
      b[i]:=h2*c[i]/2-(Sqr(h2))*d[i]/6+(y[i]-y[i-1])/h2;
      a[i]:=y[i];
    end;
   //z:=f;
end;

procedure DifSline(x,y,z:mas; xx:real; n:integer; var s1,s2:real; error:boolean);
var i,j,k:integer;
    t,t1,t2,t12:real;
    hi:real;
begin
   i:=1;
   while (xx >= x[i])and(i<=n) do inc(i);
   if i>n then begin
     error:=true; exit;
   end;
   hi:=x[i]-x[i-1];
   t:=(xx-x[i-1])/hi; //(x[i]-x[i-1]);
   t1:=1-t;
   t2:=t*t;
   t12:=t1*t1;
   s1:=-6*t1*t*(y[i-1]-y[i])/hi;
```

```
   s1:=s1+t1*z[i-1]*(1-3*t)+z[i]*(3*t-2)*t;
   s2:=6*(y[i-1]-y[i])*(2*t-1)/hi;
   s2:=s2+(z[i-1])*(6*t-4)-z[i]*(2-6*t);
   s2:=s2/hi;
end;

procedure DrawPoints(l:boolean);
var i:byte;
begin
   Form1.Image1.Canvas.Pen.Color:=clBlue;
   form1.Image1.Canvas.MoveTo(round(x[1]*koefx-xc),Form1.Image1.Height-
round((ys[1]-min)*koefy));
   for i:=1 to m do begin
     form1.Image1.Canvas.Ellipse(round((x[i])*koefx+xc)-3,
                          Form1.Image1.Height-round((ys[i]-min)*koefy)-3,
                          round((x[i])*koefx+xc)+3,
                          Form1.Image1.Height-round((ys[i]-min)*koefy)+3);
  form1.Image1.Canvas.TextOut(round((x[i])*koefx+xc-5),
                          Form1.Image1.Height-round((ys[i]-min)*koefy)+5,
                          floattostrf(ys[i],ffFixed,3,2));
   end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ClearImage;
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
var i:integer;
    z,s,c,d,z2:mas;
    xx,s1,s2:real;
    err:boolean;
begin
  GetParams2;
  ClearImage2;
  err:=false;
  xx:=strtofloat(LabeledEdit6.Text);
  Spline3(m,x,ys,x[0],x[m+1],z,s,c,d);
  DifSline(x,ys,s,xx,m,s1,s2,err);
  if err then showmessage('!!!');
  LabeledEdit4.Text:=floattostrf(s1,ffFixed,3,3);
  LabeledEdit5.Text:=floattostrf(s2,ffFixed,3,2);
  LabeledEdit7.Text:=floattostrf(fx(xx),ffFixed,3,3);
LabeledEdit8.Text:=floattostrf(fxx(xx),ffFixed,3,2);
  DrawPoints(true);
  form1.Image1.Canvas.MoveTo(round(x[1]*koefx+xc),
                        Form1.Image1.Height-round((ys[1]-min)*koefy));
  for i:=2 to m do begin
    a:=x[i]; b:=x[i+1];
    while a<=b do begin
      form1.Image1.Canvas.LineTo(round(a*koefx+xc),
```

```
                Image1.Height-round((z[i]+s[i]*(a-x[i])
                +(c[i]/2)*(a-x[i])*(a-x[i])
                +(d[i]/6)*(a-x[i])*(a-x[i])*(a-x[i])-min)*koefy));
          a:=a+dt;
        end;
      end;
    end;
        end.
```

Лабораторная работа № 12

Численное интегрирование

Пусть требуется найти определенный интеграл

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

где функция $f(x)$ непрерывна на отрезке $[a; b]$.

Для приближенного вычисления интегралов чаще всего подынтегральную функцию заменяют «близкой» ей вспомогательной функцией, интеграл от которой вычисляется аналитически. За приближенное значение интеграла принимают значение интеграла от вспомогательной функции.

Заменим функцию на отрезке $[a; b]$ ее значением в середине отрезка. Искомый интеграл, равный площади криволинейной фигуры, заменяется на площадь прямоугольника. Из геометрических соображений нетрудно записать *формулу прямоугольников*

$$\int_a^b f(x) \approx f\left(\frac{a+b}{2}\right)(b-a).$$

Приблизив $f(x)$ линейной функцией и вычислив площадь соответствующей трапеции, получим *формулу трапеций*

$$\int_a^b f(x) \approx \frac{1}{2}(f(a)+f(b))(b-a).$$

Если же приблизить подынтегральную функцию параболой, проходящей через точки $(a, f(a))$, $\left(\frac{a+b}{2}, f\left(\frac{a+b}{2}\right)\right)$, $(b, f(b))$, то получим *формулу Симпсона*

$$\int_a^b f(x) \approx \frac{1}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)(b-a)$$

Все три формулы хорошо иллюстрируются геометрически (рис. 38).

Для повышения точности интегрирования применяют составные формулы. Для этого разбивают отрезок [*a, b*] на четное $n = 2m$ число отрезков длины $h = (b - a)/n$ и на каждом из отрезков длины $2h$ применяют соответствующую формулу. Таким образом получают *составные формулы прямоугольников, трапеций и Симпсона.*

На сетке $x_i = a + ih, \quad y = f(x_i), \quad i = 0, 1, 2, ..., 2m,$ составные квадратурные формулы имеют следующий вид:

формула прямоугольников

$$\int_a^b f(x)dx = h\sum_{i=0}^{n-1}\left(f\left(x_i + \frac{h}{2}\right)\right) + R_1;$$

$$R_1 \approx \frac{h^2}{24}\int_a^b f''(x)dx = 0(h^2);$$

формула трапеций

$$\int_a^b f(x)dx = \frac{h}{2}\sum_{i=0}^{n-1}\left(f\left(x_i\right) + f\left(x_{i+1}\right)\right) + R_2;$$

$$R_2 \approx -\frac{h^2}{12}\int_a^b f''(x)dx = 0(h^2);$$

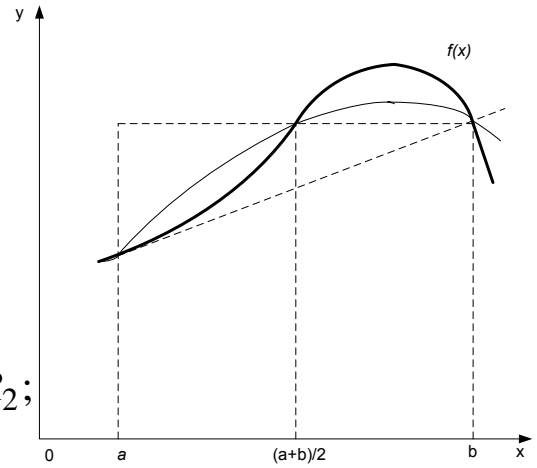


Рисунок 38

формула Симпсона

$$\int_a^b f(x)dx = \frac{h}{3}\sum_{i=0}^{m-1}\left(f\left(x_{2i}\right) + 4f\left(x_{2i+1}\right) + f\left(x_{2i+2}\right)\right);$$

$$R_3 \approx -\frac{h^4}{180}\int_a^b f^{(\mathrm{IV})}(x)dx = 0(h^4),$$

где R_1, R_2, R_3 – остаточные члены. Оценки остаточных членов получены в предположении, что соответствующие производные $f(x)$ непрерывны на [*a, b*]. Нетрудно показать, что при $n \to \infty$ приближённые значения интегралов для всех трёх формул (в предположении отсутствия погрешностей округления) стремятся к точному значению интеграла.

Для практической оценки погрешности квадратурной формулы можно использовать правило Рунге. Для этого проводят вычисления на сетках с шагом h и $h/2$, получают приближенные значения

интеграла I_h и $I_{h/2}$ и за окончательные значения интеграла принимают величины:

$I_{h/2} + (I_{h/2} - I_h)/3$—для формулы прямоугольников;

$I_{h/2} - (I_{h/2} - I_h)/3$—для формулы трапеций;

$I_{h/2} - (I_{h/2} - I_h)/15$—для формулы Симпсона.

При этом за погрешность приближенного значения интеграла принимаем величину

$|I_{h/2} - I_h|/3$—для формул прямоугольников и трапеций

и величину

$|I_{h/2} - I_h|/15$—для формулы Симпсона.

Такую оценку погрешностей применяют обычно для построения адаптивных алгоритмов, т. е. таких алгоритмов, которые автоматически так определяют величину шага *h*, что результат удовлетворяет требуемой точности.

В настоящей лабораторной работе предлагается с помощью правила Рунге найти по заданной погрешности ε наибольшее значение шага *h* для каждой приведенной квадратурной формулы (наименьшее значение *n*).

Пример. Вычислить интеграл $\int_0^1 \frac{\cos x - 1}{x^2} dx$

Вычисление подынтегральной функции:

```
function f(x: Real):Real;
begin
  Result := -0.5;
  if abs(x) > 1e-8 then
    Result := (cos(x) - 1)/(x*x)
end;
```

Формула прямоугольников

function RECT(*const a,b: Real; const n: Integer*)*: Real;*

Входные параметры:

a – нижний предел интегрирования;

b – верхний предел интегрирования;

n – число отрезков в сетке.

Выходные параметры: значение интеграла присваивается имени функции.

Текст функции:

```
function RECT(const a,b: Real; const n: Integer): Real;
var
  h, h2, s, x: Real;
  j : Integer;
begin
  h  := (b-a)/n;
  h2 := h/2.0;
  s  := 0.0;
  for j := 1 to n do begin
    x := a + j * h - h2;
    s := s + f(x);
  end;
  Result := s * h;
end;
```

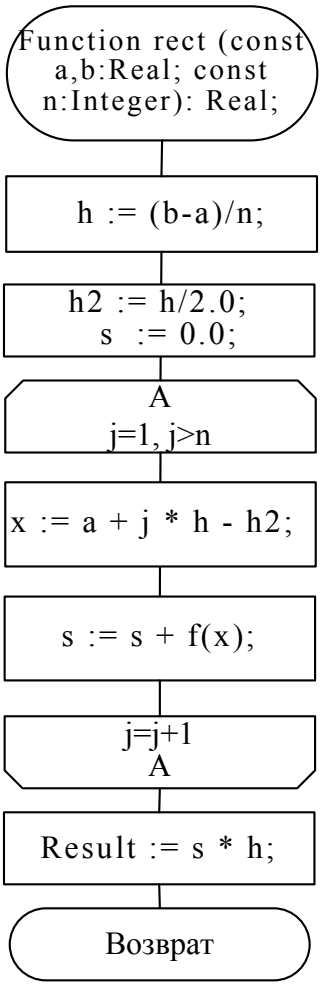


Рисунок 39– Схема алгоритма численного интегрирования по формуле прямоугольников.

Формула трапеций

function TRAP (*const a,b: Real; const n: Integer*)*: Real;*

Входные параметры:

a – нижний предел интегрирования;

b – верхний предел интегрирования;

n – число отрезков в сетке.

Выходные параметры: значение интеграла присваивается имени функции.

Текст функции:

```
function TRAP(const a,b: Real; const n: Integer): Real;
var
  h, s, x: Real;
  j : Integer;
begin
  h := (b-a)/n;
  s := (f(a)+f(b))*0.5;
  for j := 1 to n - 1 do begin
    x := a + j * h;
    s := s + f(x);
  end;
  Result := s * h;
end;
```

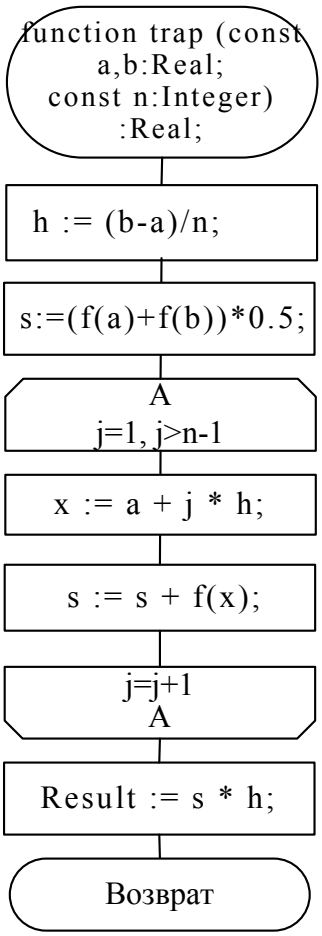


Рисунок 40 – Схема алгоритма численного интегрирования по формуле трапеций.

Формула Симпсона

function SIMPS (*const a,b: Real; const n: Integer*)*: Real;*

Входные параметры:

a – нижний предел интегрирования;

b – верхний предел интегрирования;

n – число отрезков в сетке.

Выходные параметры: значение интеграла присваивается имени функции.

Текст функции:

```
function SIMPS(const a,b: Real; const n: Integer): Real;
var
  h, s, x, z: Real;
  j, n1, n2 : Integer;
begin
  n2 := n * 2;
  n1 := n2 - 1;
  h  := (b-a)/n2;
  s  := f(a)+f(b);
  for j := 1 to n1 do begin
    z := 3.0 - Power((-1),j);
    x := a + j * h;
    s := s + z * f(x);
  end;
  Result := s * h / 3;
end;
```

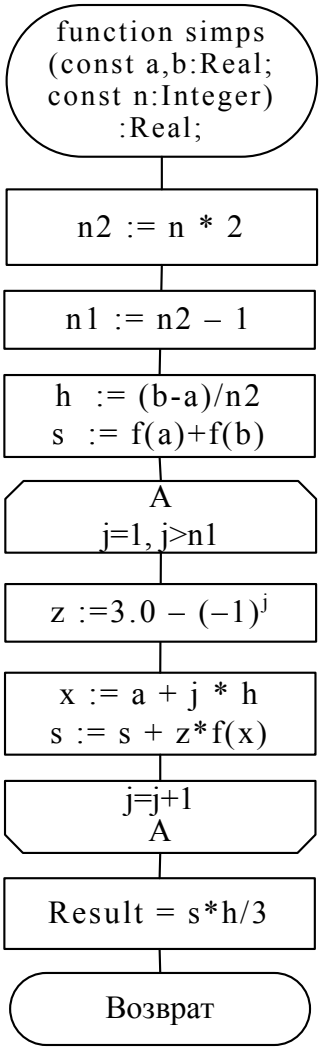


Рисунок 41 – Схема алгоритма численного интегрирования по формуле Симпсона.

Формула Гаусса

В квадратурной формуле Гаусса

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} A_i f(x_i)$$

узлы x_1, x_2, …, x_n и коэффициенты A_1, A_2, …, A_n подобраны так, чтобы формула была точна для всех многочленов степени $2n–1$. Можно показать, что если n – число узлов квадратурной формулы, то ее алгебраический порядок точности не может быть выше $2n–1$. Для приближенного вычисления интеграла по конечному отрезку [a, b] выполняем замену переменной $t = (a + b)/2 + (b – a)x/2$; тогда квадратурная формула Гаусса принимает вид

$$\int_a^b f(t)dt \approx \frac{b-a}{2}\sum_{i=1}^{n} A_i f(t_i),$$

где $t_i = (b+a)/2 + (b-a)x_i/2$; x_i – узлы квадратурной формулы Гаусса; A_i – гауссовы коэффициенты, i = 1, 2, …, n.

Можно показать, что узлы x_i квадратурных формул Гаусса являются корнями многочленов Лежандра степени n. Например, при n = 2 для узлов x_i; получаем $x_1 = -1/\sqrt{3}, x_2 = 1/\sqrt{3}$. При этом A_1=A_2 = 1. Таким образом, квадратурная формула Гаусса

$$\int_{-1}^{1} f(x)dx \approx f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

имеет такой же алгебраический порядок точности, что и формула Симпсона, но требует вычисления подынтегральной функции только в двух точках.

Если подынтегральная функция достаточно гладкая, то квадратурная формула Гаусса обеспечивает очень высокую точность при небольшом числе узлов, так как для погрешности R_n формулы Гаусса с n узлами справедлива оценка

$$\left|\mathrm{R_n}\right| \approx \frac{b-a}{2{,}5\sqrt{n}}\left(\frac{b-a}{3n}\right)^{2n} \max_{[a,b]}\left|f^{(2n)}(x)\right|.$$

В данной лабораторной работе предлагается вычислить интеграл по квадратурной формуле Гаусса с восемью узлами:

$x_1 = -x_8 = -0.96028986$, $A_1 = A_8 = 0.10122854$; $x_2 = -x_7 =$ $= -0.79666648$; $A_2 = A_7 = 0.22238103$; $x_3 = -x_6 = -0.52553242$, $A_3 = A_6$ $= = 0.31370664$; $x_4 = -x_5 = -0.18343464$, $A_4 = A_5 = 0.36268378$.

function GAUSS(const a,b: Real): Real;

Входные параметры:

a – нижний предел интегрирования;

b – верхний предел интегрирования.

Выходные параметры: значение интеграла присваивается имени функции.

(В функции используются массивы: ag – массив коэффициентов квадратурной формулы Гаусса; xg – узлы квадратурной формулы Гаусса.)

Текст функции:

```
function GAUSS(const a,b: Real): Real;
const
  ag : array [1..8] of Real = (0.10122854,   0.22238104,
```

```
                        0.31370664,   0.36278378,
                        0.36268378,   0.31370664,
                        0.22238104,   0.10122854);
  xg : array [1..8] of Real = (-0.96028986, -0.79666648,
                        -0.52553242, -0.18343464,
                         0.18343464,  0.52553242,
                         0.79666648,  0.96028986);
var
  a1, a2, g, x : Real;
  i: Integer;
begin
  a1 := (b+a)*0.5;
  a2 := (b-a)*0.5;
  g  := 0.0;
  for i := 1 to 8 do begin
    x := a1 + a2 * xg[i];
    g := g + ag[i] * f(x);
  end;
  Result := g * a2;
end;
```

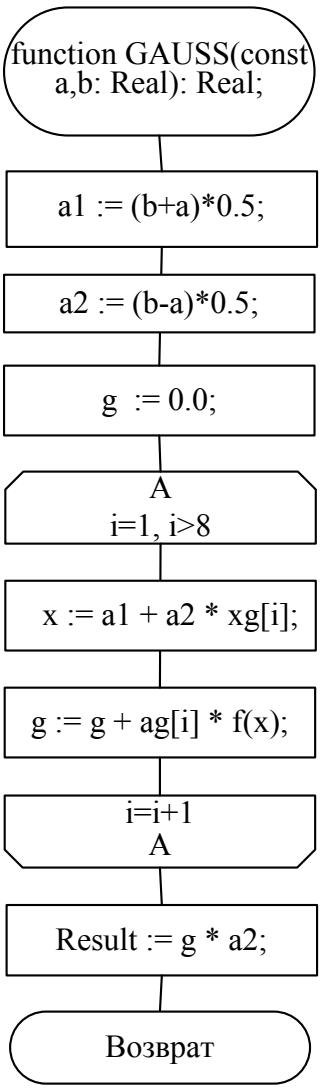
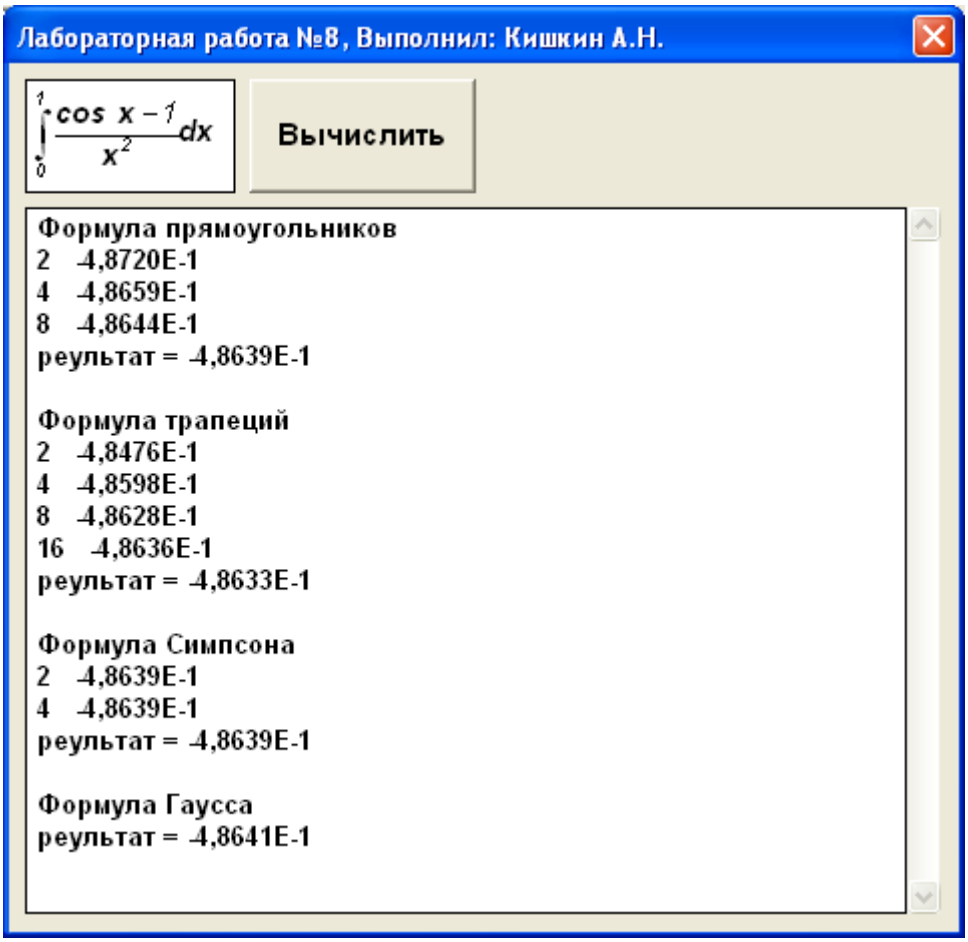


Рисунок 42. – Схема алгоритма численного интегрирования по формуле Гаусса.

Текст программы:

```
procedure TForm1.Button1Click(Sender: TObject);
  var
  eps, a, b, Res, zz, ts: Real;
  n: Integer;
begin
  eps := 0.0001;
  n   := 1;
  a   := 0.0;
  b   := 1.0;
  Memo1.Lines.Add('Формула прямоугольников');
  res := 1e+10;
  repeat
    n   := n * 2;
    zz  := Res;
    Res := RECT(a,b,n);
    Memo1.Lines.Add(IntToStr(n) + '   '+FloatToStrF(Res,ffExponent,5,5));
    ts  := abs(res - zz)/3;
  until ts < eps;
  Memo1.Lines.Add('реультат = '+FloatToStrF(Res + ts,ffExponent,5,5));
  Memo1.Lines.Add('');
  Memo1.Lines.Add('Формула трапеций');
  n   := 1;
  res := 1e+10;
  repeat
    n   := n * 2;
    zz  := Res;
    Res := TRAP(a,b,n);
    Memo1.Lines.Add(IntToStr(n) + '   '+FloatToStrF(Res,ffExponent,5,5));
    ts  := abs(res - zz)/3;
  until ts < eps;
  Memo1.Lines.Add('реультат = '+FloatToStrF(Res + ts,ffExponent,5,5));
  Memo1.Lines.Add('');
  Memo1.Lines.Add('Формула Симпсона');
  n   := 1;
  res := 1e+10;
  repeat
    n   := n * 2;
    zz  := Res;
    Res := SIMPS(a,b,n);
    Memo1.Lines.Add(IntToStr(n) + '   '+FloatToStrF(Res,ffExponent,5,5));
    ts  := abs(res - zz)/15;
  until ts < eps;
  Memo1.Lines.Add('реультат = '+FloatToStrF(Res + ts,ffExponent,5,5));
  Memo1.Lines.Add('');
  Memo1.Lines.Add('Формула Гаусса');
  Memo1.Lines.Add('реультат = '+FloatToStrF(Gauss(a,b),ffExponent,5,5));
end;
```

Результаты работы программы:



Варианты заданий

Таблица 9

Номер варианта	Задание	Номер варианта	Задание
1	$\int_0^1 \cos(x+x^3)\,dx$	2	$\int_0^1 \sin(x^4+2x^3+x^2)\,dx$
3	$\int_0^1 e^{\sin x}dx$	4	$\int_0^1 \sin x\, e^{-x^2}dx$
5	$\int_0^1 e^{\cos x}dx$	6	$\int_0^1 \mathrm{ch}\, x^2\,dx$
7	$\int_0^1 \cos x^2\,dx$	8	$\int_0^1 \sin(x+x^3)dx$
9	$\int_0^1 \cos x\, e^{-x^2}dx$	10	$\int_1^2 \sin 2x\, e^{-x^2}dx$
11	$\int_1^2 e^{-\left(x+\frac{1}{x}\right)}dx$	12	$\int_1^2 \ln x(x+1)^{-1}dx$
13	$\int_{\pi/2}^{\pi} \sqrt{x}e^{-x^2}\,dx$	14	$\int_0^1 \cos x^3\,dx$
15	$\int_0^1 \cos x^2\,dx$	16	$\int_{\pi/4}^{\pi/2} \ln\sin x\,dx$
17	$\int_0^{\pi} \cos(2\sin x)\,dx$	18	$\int_0^{\pi} x^2e^{-x^2}dx$
19	$\int_0^{\pi} x^4e^{-x^2}dx$	20	$\int_{\pi/2}^{\pi} \cos(x+x^3)dx$
21	$\int_1^2 \sin x^3\,dx$	22	$\int_1^2 x^{-1}\ln(1+x)\,dx$
23	$\int_1^2 x^{-1}e^x dx$	24	$\int_1^2 \mathrm{sh}\, x^2\,dx$
25	$\int_0^{\pi/4} x\sin x^3\,dx$	26	$\int_0^{\pi/4} \ln(1+\cos x)\,dx$
27	$\int_0^{\pi/4} x\cos x^3\,dx$	28	$\int_{0.1}^2 \frac{\cos x}{\sqrt{x}}\,dx$
29	$\int_{0.1}^2 \frac{\sin x}{\sqrt{x}}\,dx$	30	$\int_0^{\pi} \sin(2\cos x)\,dx$

Лабораторная работа № 13

Приближенное вычисление преобразования Фурье

При решении широкого круга прикладных задач, в частности в цифровом спектральном анализе, в цифровом моделировании фильтров, распознавании образов, анализе речевых сигналов, а также при решении многих задач численного анализа, возникает необходимость вычисления интегралов вида

$$\int_a^b f(x)e^{-i\omega x}dx, \qquad (1)$$

где ω — произвольное действительное число. Для вычисления этого интеграла можно применить многие известные классические правила интегрирования, такие, например, как формула трапеций, парабол и др., основанные на формулах Котеса, Гаусса и т. д. Однако все эти формулы имеют существенный недостаток.

Формулы, которые упоминались выше, получают с помощью замены интегрируемой функции алгебраическим многочленом невысокого порядка на всем отрезке интегрирования или его частях. Поэтому следует ожидать, что они будут давать хорошую точность, если интегрируемая функция достаточно гладка и не очень быстро меняется. В интеграле (1) интегрируемыми функциями являются произведения $f(x)\sin\omega x$ и $f(x)\cos\omega x$. Если параметр ω – большое число, то функции $\cos\omega x$ и $\sin\omega x$ быстро колеблются, и для того чтобы достаточно точно проследить за изменением произведений $f(x)\sin\omega x$ и $f(x)\cos\omega x$ даже при медленно меняющейся функции $f(x)$, нужно взять в квадратурной формуле большое число узлов. В результате вычисления могут стать трудными или даже невыполнимыми.

Чтобы построить квадратурную формулу, пригодную для вычисления интеграла (1) в широком диапазоне изменения параметра ω, необходимо учесть множители $\sin\omega x$ и $\cos\omega x$ $\left(e^{-i\omega x}\right)$ в подынтегральной функции.

В прикладных задачах чаще всего требуется вычислить интеграл (1) не при одном, а сразу при нескольких значениях параметра ω вида $\omega_k = k\Delta\omega,\ \ k = 0,\ldots,n-1$. Все это при-

водит к необходимости получить такую квадратурную формулу и алгоритм ее реализации, в которых вычисления проводятся не последовательно для каждого значения параметра ω_k а сразу для всей совокупности значений этого параметра.

В данной лабораторной работе используется квадратурная формула вида

$$I(\omega) = \sum_{j=0}^{n-1} f_j A_j , \qquad (2)$$

где

$$A_j = \int_{x_j}^{x_{j+1}} exp(-i\,\omega x)\,dx = \frac{1-e^{-i\omega h}}{i\omega} e^{-i\omega x_j};$$

$x_j = a + jh, \quad h = \frac{b-a}{n}, \quad j = 0, 1, \ldots, n-1, \quad x_j$ – узлы равномерной сетки на $[a,b]$; $f_j = f(x_j)$ значения функции $f(x)$ в узлах сетки.

Для значений $\omega_k = \frac{2\pi k}{b-a}, \quad \Delta\omega = \frac{2\pi}{b-a}$ формулу (2) можно переписать в виде

$$I(\omega) = \frac{1-e^{-i\omega_k h}}{i\omega_k h} h e^{-i\omega_k a} F(k), \qquad (3)$$

где

$$F(k) = \sum_{j=0}^{n-1} f_j e^{-i\frac{2\pi kj}{n}} . \qquad (4)$$

Переход от величин f_j к величинам $F(k)$ (j; k = 0, 1, …, n – 1) называется *дискретным преобразованием Фурье* (ДПФ).

Дискретное преобразование Фурье является составной частью решения многих прикладных задач. Замечательным свойством дискретного преобразования Фурье является возможность его обращения, т. е. восстановления величин f_j по известным значениям $F(k)$ (j, k, 1, …, n – 1). Эта операция осуществляется по формулам обратного преобразования Фурье

$$f_j = \frac{1}{n} \sum_{k=0}^{n-1} F(k) e^{i\frac{2\pi kj}{n}} . \qquad (5)$$

Непосредственно осуществление дискретных преобразований Фурье по формулам (4) или (5) требует $O(n^2)$ арифметических операций. Для сокращения объема вычислений были разработаны алгоритмы быстрого преобразования Фурье (БПФ). Основная идея *алгоритма быстрого преобразования Фурье* основана на том, что при составном n в правой части (4) или (5) можно выделить такие группы слагаемых, которые дают вклад во многие коэффициенты $F(k)$. Наибольшее распространение получил алгоритм БПФ, разработанный для случая $n = 2^m$. Для его реализации требуется $O(n \log_2 n)$ арифметических операций.

Процедура FF производит быстрое преобразование Фурье массива из n комплексных чисел.

procedure FF (*var xr, xi, yr, yi: TMas; const n, ind: Integer*);

Входные параметры: *xr, xi* – массивы размерности n, содержат соответственно действительные и мнимые части элементов заданного массива; *yr, yi* – рабочие массивы размерности n; n – количество заданных комплексных чисел (целая степень двойки);

ind – признак, указывающий направление преобразования Фурье: $ind > 0$ – производится прямое преобразование;

$ind < 0$ – производится обратное преобразование.

Выходные параметры: действительная и мнимая части дискретного преобразования Фурье хранятся в массивах *xr* и *xi*, соответственно

Схема алгоритма приведена на рис 43.

Текст процедуры:

```
procedure FF(var xr, xi, yr, yi: TMas; const n, ind: Integer);
var
  i,ni,j,jm,k,k1,m,mm, m2,log2,l,l1,it: Integer;
  xa,xb,w,si,co: Real;
begin
  k := n;
  log2 := 0;
  repeat
    k := k div 2;
    log2 := log2 + 1;
  until (k < 2);
```

```
mm := 1;
{ cycl 1 }
for m := 1 to log2 do
begin
  m2 := mm*2;
  k1 := trunc(power(2,(log2 - m)) - 1);
  l1 := mm - 1;
  { cycl 2 }
  for k := 1 to k1 + 1 do
  begin
    { cycl 3 }
    for l := 1 to l1 + 1 do
    begin
      j := m2*(k - 1) + l;
      i := mm*(k - 1) + l;
      w := pi*(l - 1)/mm;
      si := ind*sin(w);
      co := cos(w);
      ni := trunc(power(2,(log2 - 1)) + i);
      jm := trunc(j + power(2, (m - 1)));
      xa := xr[ni]*co + xi[ni]*si;
      xb := xi[ni]*co - xr[ni]*si;
      yr[j]  := xr[i] + xa;
      yi[j]  := xi[i] + xb;
      yr[jm] := xr[i] - xa;
      yi[jm] := xi[i] - xb;
    end;
    { cycl 3 }
  end;
  { cycl 2 }
{ cycl 5 }
for it := 1 to n do
begin
  xr[it] := yr[it];
  xi[it] := yi[it];
end;
{ cycl 5 }
mm := m2;
end;
{ cycl 1 }
if ind < 0 then exit;
{ cycl 4 }
for i := 1 to n do
begin
  xr[i] := xr[i]/n;
  xi[i] := xi[i]/n;
end;
{ cycl 4 } end;
```

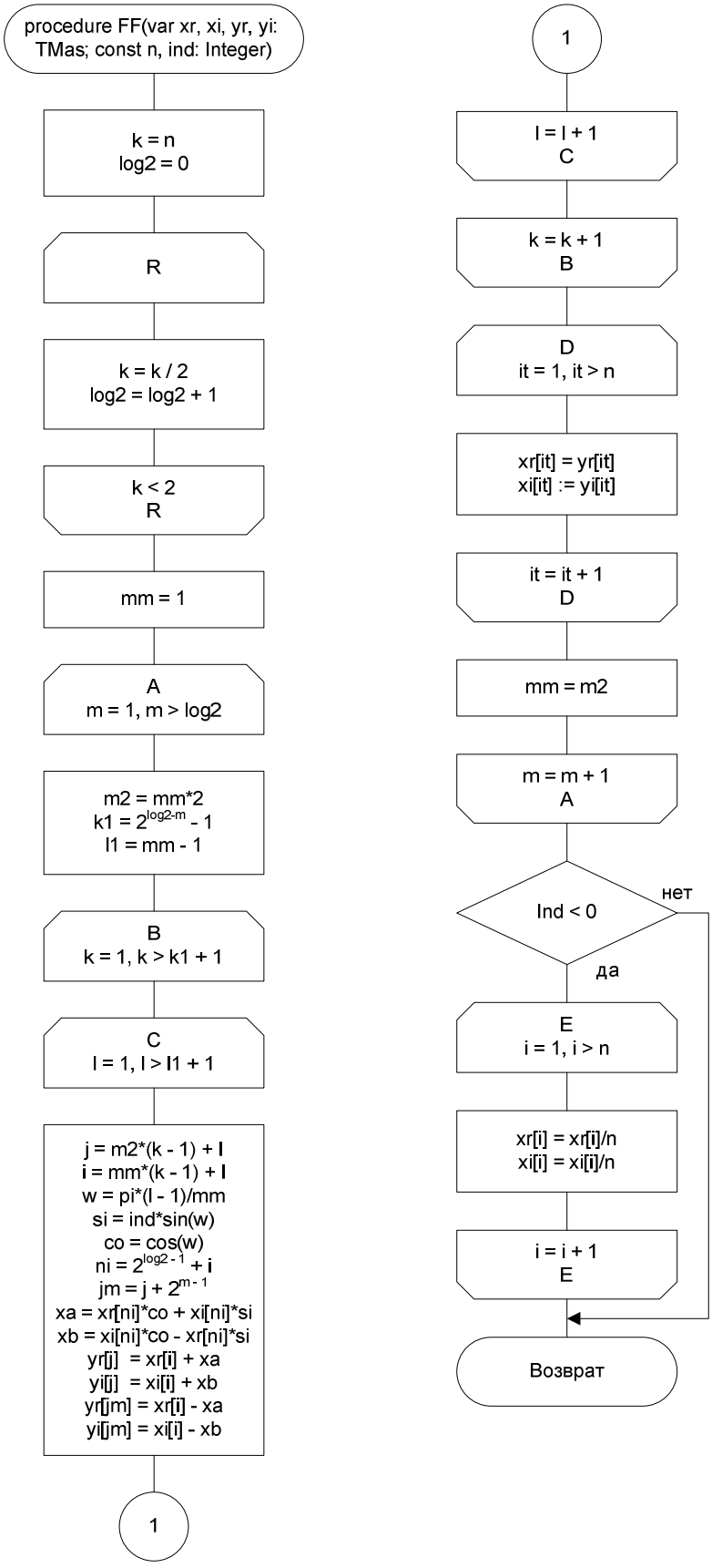


Рисунок 43 – Схема алгоритма прямого и обратного быстрого преобразования Фурье.

Процедура IFF реализует алгоритм вычисления интеграла (1) по формулам (2) – (4)

procedure IFF (*const a, b: Real; var ar, ai, xr, xi: TMas; const n: Integer;*

eps: Real; const ip: Integer);

В х о д н ы е п а р а м е т р ы *: a, b* – концы интервала интегрирования; *ar, ai* – массивы размерности *n*, содержат соответственно действительную и мнимую части значений функции $f(x)$ в равноотстоящих узлах $x_i = a + i(b - a)/n$, $i = 0, 1, ..., n - 1$ на $[a, b]$; *xr*, *xi* – рабочие массивы размерности *n*; *n* – количество узлов сетки на $[a, b]$; *eps* – параметр, регулирующий точность вычисления интеграла при близких к нулю значениях $\omega_k x$; *ip* – параметр, определяющий аргумент экспоненты:

ip = 1 – вычисляем интеграл от $f(x) = e^{-i\omega x}$;

ip = –1 – вычисляем интеграл от $f(x) = e^{i\omega x}$.

В ы х о д н ы е п а р а м е т р ы : действительная и мнимая части значения интеграла в точках $\omega_k = 2\pi k/(b-a)$ находятся в массивах *ar* и *ai* соответственно.

(Процедура использует процедуру *procedure FF* (*var xr, xi, yr, yi: TMas; const n, ind: Integer*); для вычисления быстрого преобразования Фурье.)

Схема алгоритма процедуры IFF приведена на рисунке 44.

Текст процедуры:

```
procedure IFF (const a, b: Real; var ar, ai, xr, xi: TMas; const n: Integer; eps: Real; const ip: In-
teger);
var
  i,k : Integer;
  h, w, c, s, w1, w2, w3, w4, w5, a1, b1: Real;
begin
  h := (b - a)/n;
  {Вычисление быстрого преобразования Фурье процедурой FF}
  FF(ar,ai,xr,xi,n,ip);
  for i := 1 to n do
  begin
    k := i - 1;
    w := k*2*pi/(b - a);
    c := cos(w*a);
    s := sin(w*a);
    w := w*h*(b - a);
```

```
    if w < eps then
    begin
      w4 := h - 2*pi*pi*k*k*power(h,3)/power((b - a),2);
      w5 := 2*pi*k*h*h/(b - a);
      a1 := w4*ar[i] + ip*w5*ai[i];
      b1 := w4*ai[i] - ip*w5*ar[i];
    end else
    begin
      if k=0 then begin
        w1 :=0;
        w3 := 0
      end else begin
        w1 := sin(2*pi*k/n)*(b - a)/(2*pi*k);
        w2 := 2*power(sin(pi*k/n),2);
        w3 := w2*(b - a)/(2*pi*k);
      end;
      w2 := 2*power(sin(pi*k/n),2);
      a1 := w1*ar[i] + ip*w3*ai[i];
      b1 := w1*ai[i] - ip*w3*ar[i];
    end;
  ar[i] := a1*c + b1*s*ip;
  ai[i] := b1*c - a1*s*ip;
  if ip > 0 then
  begin
    ar[i] := ar[i]*n*n;
    ai[i] := ai[i]*n*n;
  end;
  ar[i] := ar[i]/n;
  ai[i] := ai[i]/n;
  end;
end;
```

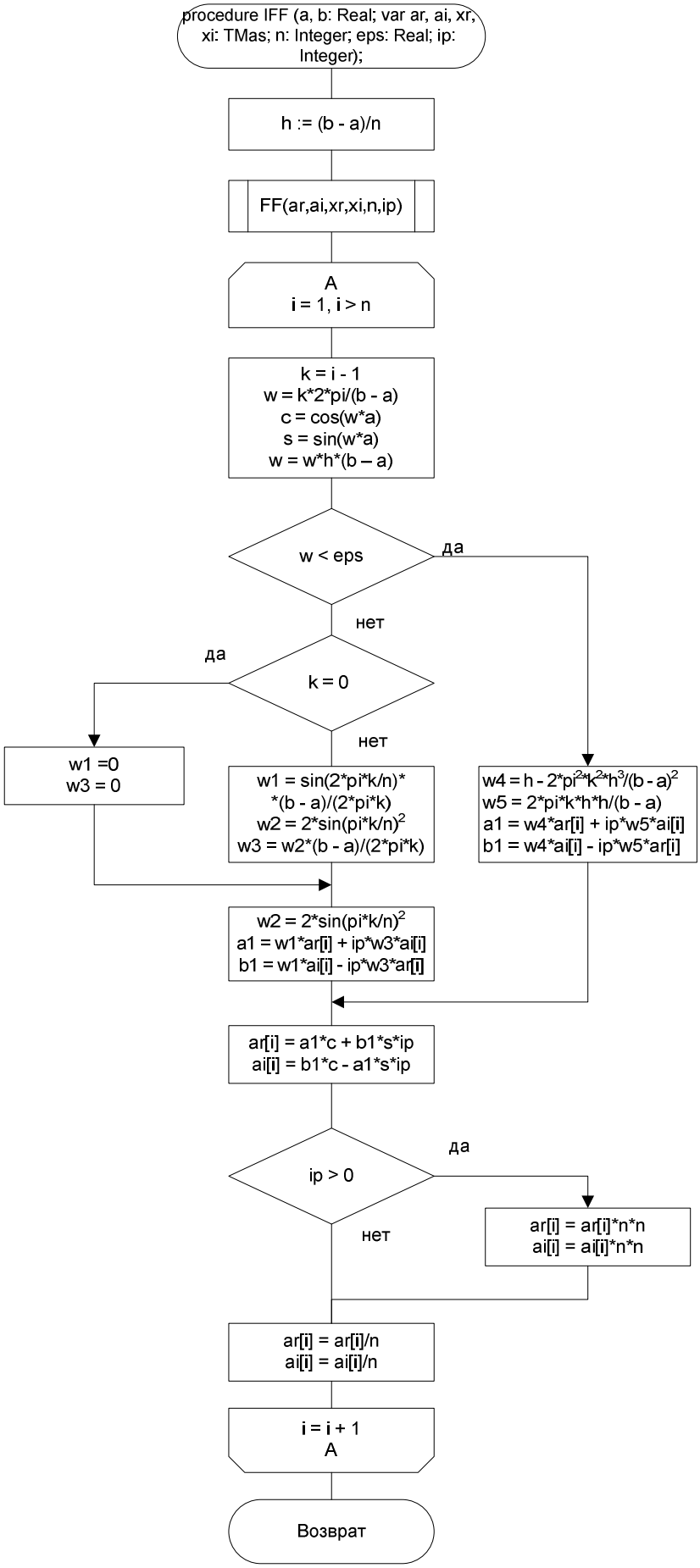


Рисунок 44 – Схема алгоритма процедуры IFF

Пример. Вычислить $\int_a^b e^{-x^2} e^{i\omega x} dx$, используя квадратурную формулу с 512 узлами (при $\omega = \omega_k = \frac{2\pi k}{b-a}, \quad k = 0,1,\ldots,n-1, n = 512$.)

Программа вычисления интеграла содержит вычисление значений *f*(*x*) в узлах равномерной сетки из *n* узлов на [*a,b*]. Вычисленные значения размещаются в соответствующих массивах *ar* и *ai.* Далее вызывается процедура IFF.

Схема алгоритма приведена на рисунке 45.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, Math;

type
  TForm1 = class(TForm)
    sgOut: TStringGrid;
    btnRun: TButton;
    procedure btnRunClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

type TMas = array of Real;

procedure FF(var xr, xi, yr, yi: TMas; const n, ind: Integer);
var
  i,ni,j,jm,k,k1,m,mm, m2,log2,l,l1,it: Integer;
  xa,xb,w,si,co: Real;
begin
  k := n;
  log2 := 0;
  repeat
```

```
    k := k div 2;
    log2 := log2 + 1;
  until (k < 2);
  mm := 1;
  { cycl 1 }
  for m := 1 to log2 do
  begin
    m2 := mm*2;
    k1 := trunc(power(2,(log2 - m)) - 1);
    l1 := mm - 1;
    { cycl 2 }
    for k := 1 to k1 + 1 do
    begin
      { cycl 3 }
      for l := 1 to l1 + 1 do
      begin
        j := m2*(k - 1) + l;
        i := mm*(k - 1) + l;
        w := pi*(l - 1)/mm;
        si := ind*sin(w);
        co := cos(w);
        ni := trunc(power(2,(log2 - 1)) + i);
        jm := trunc(j + power(2, (m - 1)));
        xa := xr[ni]*co + xi[ni]*si;
        xb := xi[ni]*co - xr[ni]*si;
        yr[j]  := xr[i] + xa;
        yi[j]  := xi[i] + xb;
        yr[jm] := xr[i] - xa;
        yi[jm] := xi[i] - xb;
      end;
      { cycl 3 }
    end;
    { cycl 2 }
  { cycl 5 }
  for it := 1 to n do
  begin
    xr[it] := yr[it];
    xi[it] := yi[it];
  end;
  { cycl 5 }
  mm := m2;
  end;
  { cycl 1 }
  if ind < 0 then exit;
  { cycl 4 }
  for i := 1 to n do
  begin
    xr[i] := xr[i]/n;
    xi[i] := xi[i]/n;
  end;
  { cycl 4 }
end;
```

```
procedure IFF (const a, b: Real; var ar, ai, xr, xi: TMas; const n: Integer; eps: Real; const ip: In-
teger);
var
  i,k : Integer;
  h, w, c, s, w1, w2, w3, w4, w5, a1, b1: Real;
begin
  h := (b - a)/n;
  {Вычисление быстрого преобразования Фурье программой FF}
  FF(ar,ai,xr,xi,n,ip);
  for i := 1 to n do
  begin
    k := i - 1;
    w := k*2*pi/(b - a);
    c := cos(w*a);
    s := sin(w*a);
    w := w*h*(b - a);
    if w < eps then
    begin
      w4 := h - 2*pi*pi*k*k*power(h,3)/power((b - a),2);
      w5 := 2*pi*k*h*h/(b - a);
      a1 := w4*ar[i] + ip*w5*ai[i];
      b1 := w4*ai[i] - ip*w5*ar[i];
    end else
    begin
      if k=0 then begin
        w1 :=0;
        w3 := 0
      end else begin
        w1 := sin(2*pi*k/n)*(b - a)/(2*pi*k);
        w2 := 2*power(sin(pi*k/n),2);
        w3 := w2*(b - a)/(2*pi*k);
      end;
      w2 := 2*power(sin(pi*k/n),2);
      a1 := w1*ar[i] + ip*w3*ai[i];
      b1 := w1*ai[i] - ip*w3*ar[i];
    end;
  ar[i] := a1*c + b1*s*ip;
  ai[i] := b1*c - a1*s*ip;
  if ip > 0 then
  begin
    ar[i] := ar[i]*n*n;
    ai[i] := ai[i]*n*n;
  end;
  ar[i] := ar[i]/n;
  ai[i] := ai[i]/n;
  end;
end;

procedure TForm1.btnRunClick(Sender: TObject);
var
  n, i : Integer;
```

```
  a, b, h, x, omg, eps : Real;
  xr, xi, ar, ai : TMas;
begin
  Form1.sgOut.Cells[0,0]:='№';
  Form1.sgOut.Cells[1,0]:='омега';
  Form1.sgOut.Cells[2,0]:='дейст-я часть';
  Form1.sgOut.Cells[3,0]:='мнимая часть';
  eps := 0.1E-6;
  a := 0;
  b := 20;
  n := 512;
  h := 20/n;
  omg := 0;
  SetLength(xr, n+1);
  SetLength(xi, n+1);
  SetLength(ar, n+1);
  SetLength(ai, n+1);
  for i := 1 to n do
  begin
    x := (i - 1)*h;
    ar[i] := exp(-x*x);
    ai[i] := 0;
  end;
  IFF(a,b,ar,ai,xr,xi,n,eps,1);
  sgOut.RowCount := n + 1;
  for i := 1 to n do
  begin
    omg := (i - 1)*2*pi/(b - a);
    sgOut.Cells[0,i] := IntToStr(i);
    sgOut.Cells[1,i] := FloatToStrF(omg,ffFixed,7,6);
    sgOut.Cells[2,i] := FloatToStrF(ar[i],ffExponent,7,7);
    sgOut.Cells[3,i] := FloatToStrF(ai[i],ffExponent,7,7);
  end;
end;

end.
```

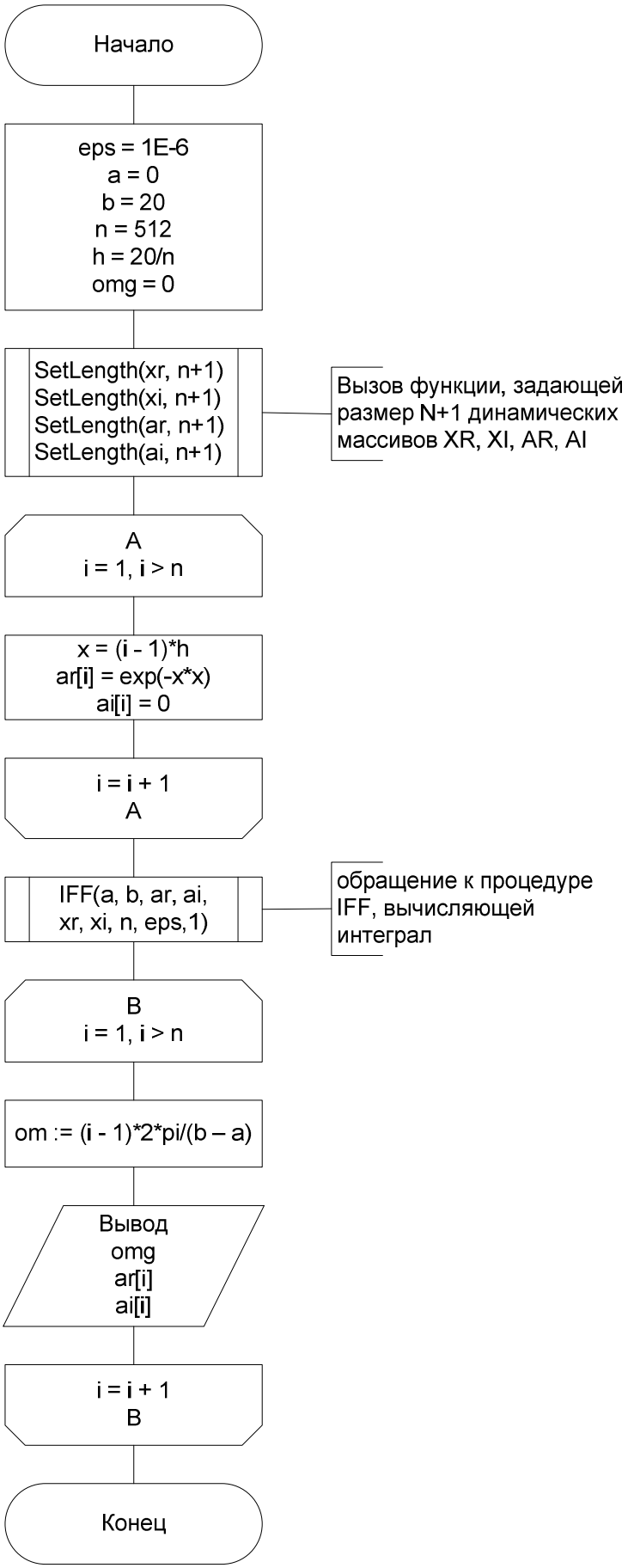


Рисунок 45 – Схема алгоритма вычисления интеграла.

Вычисления по программе привели к следующим результатам:

№	ω	Действительная часть	Мнимая часть
1	0,000000	9,057582E-1	0,000000E+0
2	0,314159	8,831889E-1	-1,599023E-1
3	0,628319	8,187761E-1	-3,042652E-1
4	0,942478	7,216180E-1	-4,205082E-1
5	1,256637	6,045151E-1	-5,011387E-1
6	1,570796	4,812212E-1	-5,445265E-1
7	1,884956	3,638563E-1	-5,542706E-1
8	2,199115	2,611314E-1	-5,375433E-1
9	2,513274	1,776797E-1	-5,030347E-1
10	2,827433	1,144008E-1	-4,591037E-1
11	3,141593	6,946244E-2	-4,125289E-1
12	3,455752	3,951928E-2	-3,679761E-1
13	3,769911	2,079204E-2	-3,280630E-1
14	4,084070	9,813039E-3	-2,937869E-1
15	4,398230	3,804643E-3	-2,650754E-1
16	4,712389	7,651842E-4	-2,412863E-1
17	5,026548	-6,231108E-4	-2,215746E-1
18	5,340708	-1,159402E-3	-2,051132E-1
. . .	. . .	. . .	. . .
493	154,566400	4,695544E-5	-6,865602E-3
494	154,880500	4,830774E-5	-6,908742E-3
495	155,194700	4,707561E-5	-6,966648E-3
496	155,508800	3,981786E-5	-7,044293E-3
497	155,823000	2,010035E-5	-7,147567E-3
498	156,137200	-2,309409E-5	-7,282281E-3
499	156,451300	-1,069578E-4	-7,451919E-3
500	156,765500	-2,556503E-4	-7,653768E-3
501	157,079600	-4,990090E-4	-7,873511E-3
502	157,393800	-8,676888E-4	-8,079315E-3
503	157,708000	-1,383714E-3	-8,217708E-3
504	158,022100	-2,046932E-3	-8,214580E-3
505	158,336300	-2,820312E-3	-7,984678E-3
506	158,650400	-3,619643E-3	-7,451095E-3
507	158,964600	-4,314501E-3	-6,572379E-3
508	159,278700	-4,745771E-3	-5,370084E-3
509	159,592900	-4,759961E-3	-3,945974E-3
510	159,907100	-4,253151E-3	-2,478437E-3
511	160,221200	-3,210886E-3	-1,193197E-3
512	160,535400	-1,728354E-3	-3,129204E-4

Варианты заданий

Вычислить $\int_{-\pi}^{\pi} f(x)e^{i\omega x}\,dx$

Таблица 10

Номер варианта	f(x)	Номер варианта	f(x)
1	$\sin x^2$	16	$\cos x^2$
2	$\sin\left(\sqrt{1+x^2}\right)$	17	$\cos\left(\sqrt{1+x^2}\right)$
3	$\operatorname{arctg}\left(\frac{1}{1+10x}\right)$	18	$\frac{1}{1+100x^2}$
4	$\arcsin\left(\frac{x+1}{10+x^2}\right)$	19	$\ln(x^2-\pi^2+1)$
5	$\sin x\cos x^2$	20	$\cos\left(x+\frac{\pi}{2}\right)\sin x^2$
6	$\sin x\ln(x^2+1)$	21	$\sqrt{\pi^2-x^2}\operatorname{arctg} x$
7	$\sqrt{\pi^2-x^2}\sin x^2$	22	$(x^2-\pi^2)\sin x^2$
8	$\sin x\, e^{-x^2}$	23	$e^{-10(x-\pi)^2}$
9	$\sqrt{\pi^2-x^2}\operatorname{arctg}(x^2+7)$	24	$\sin x\operatorname{sh}(x^2-\pi)$
10	$\operatorname{ch}(x^2+3)\sin x$	25	$\sin x\operatorname{ch}(\ln x+2\pi)$
11	$\sqrt{x^2+18x+20}\sin x^2$	26	$\frac{x^2+\operatorname{ch} x}{1+65x^2}$
12	$\ln\left(\frac{x+12}{4}\operatorname{ch} x\right)$	27	$\sqrt{\pi^2-x^2}\operatorname{ch} x$
13	$\cos(x-\pi)^2\sin x$	28	$\sin(x-\pi)^2\sin x$
14	$\ln\left(\sqrt{x^4+2}+\sin^2 x\right)$	29	$\frac{x^2-2x+23}{10x+100}\sin x$
15	$\sin x\cos x^2$	30	$\cos^2\left(x+\frac{\pi}{2}\right)\sin x^2$

Список использованной литературы

1. Амосов А.А., Дубинский Ю.А., Копченова Н.В. Вычислительные методы для инженеров – М.: Высшая школа, 1994–544 с.
2. Бахвалов Н.С. Численные методы – М.: Наука, 1973–631 с.
3. Вержбицкий В.М. Численные методы. Математический анализ и обыкновенные дифференциальные уравнения. – М.: Высшая школа, 2001 –400 с.
4. Волков Е.А. Численные методы – СПб.: Лань, 2004– 256 с.
5. Воробьева Г.Н., Данилова А.Н. Практика по численным методам – М.: Высшая школа, 1979 – 184 с.
6. Демидович Б.П., Марон И.А., Шувалова Э.З. Численные методы анализа – М.: Наука, 1967 – 368 с.
7. Ильин В.П. Численный анализ, Часть 1 –Новосибирск : ИВМ и МГСО РАН, 2004 –335с.
8. Калиткин Н.Н. Численные методы – М.: Наука,1978–512 с.
9. Копченова Н.В., Марон И.А. Вычислительная математика в примерах и задачах – М.: Наука, 1972 – 246 с.
10. Костомаров Д.П., Фаворский А.П Вводные лекции по численным методам – М.: Логос, 2004 –184с.
11. Лебедев В.И. Функциональный анализ и вычислительная математика – М.: Физматлит, 2005 –296с.
12. Марчук Г.И. Методы вычислительной математики – М.: Наука, 1989 – 372 с.
13. Плис А.И., Сливина Н.А. Лабораторный практикум по высшей математике – М.: Высшая школа, 1994 – 416 с.
14. Ракитин В.И. Руководство по методам вычислений и приложения MATHCAD –. М.: Физматлит, 2005 –264с.
15. Самарский А.А. Введение в численные методы – М.: Наука, 1982 – 287 с.
16. Самарский А.А., Гулин А.В. Численные методы – М.: Наука, 1989 – 429 с.
17. Турчак Л.И., Плотников П.В. Основы численных методов – М.: Физматлит, 2002 – 304с.
18. Тыртышников Е.Е. Методы численного анализа – М.:Академия, 2007 – 320с.
19. Хемминг Р.В. Численные методы для научных работников и инженеров - М.: Наука, 1972 – 400 с.

В.Н. ТАРАСОВ, Н.Ф. БАХАРЕВА

ЧИСЛЕННЫЕ МЕТОДЫ ТЕОРИЯ АЛГОРИТМЫ ПРОГРАММЫ

ИЗДАНИЕ ВТОРОЕ,
ПЕРЕРАБОТАННОЕ

Подписано в печать: 29.09.2008
Тираж: 120 экз. 15 усл. п.л. Заказ №
Отпечатано в типографии ГОУ ВПО ПГАТИ
443090, г. Самара, Московское шоссе, 77