МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ
Государственное бюджетное профессиональное образовательное учреждение Московской области
«Воскресенский колледж»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

ОП.10 «Численные методы»

Наименование специальности

09.02.07 «Информационные системы и программирование»

Квалификация выпускника

Программист

2020г.

Методические рекомендации по выполнению практических работ по учебной дисциплине разработаны на основе Федерального государственного образовательного стандарта (далее – ФГОС) по специальности среднего профессионального образования (далее – СПО)
09.02.07 «Информационные системы и программирование»

Организация разработчик: Государственное бюджетное профессиональное образовательное учреждение Московской области «Воскресенский колледж»

Разработчики:

Вострякова А.В., преподаватель компьютерных дисциплин

Методические рекомендации рассмотрены на заседании предметной (цикловой) комиссией компьютерных дисциплин

«___»_____2020г.

Председатель цикловой комиссии _____/Рязанцева О.В./

Утверждена зам директора по УР _____/Куприна Н.Л./

«___»_____2020 г.

Содержание

Введение 4

Практическая работа №1
Программирование интерполяционных алгоритмов 5

Практическая работа №2
Программирование алгоритмов решения дискретных моделей на примерах задач численного интегрирования 13

Практическая работа №3
Реализация прямых и итерационных алгоритмов решения СЛАУ, их приложения 20

Практическая работа №4
Программирование итерационных алгоритмов решения нелинейных уравнений 31

Практическая работа №5
Организация многооконного приложения для решения о.д.у. и систем о.д.у. одношаговыми разностными методами 37

Список использованной литературы 59

ВВЕДЕНИЕ

Данные методические указания для проведения лабораторных работ по ПМ.10 «Численные методы» предназначены для реализации ФГОС СПО по специальности 09.02.07 Информационные системы и программирование с целью закрепления теоретических знаний и практических умений.
В сборнике содержатся методические указания по выполнению лабораторных работ в интегрированной системе программирования C#.
При выполнении лабораторных работ студент должен
В результате освоения дисциплины обучающийся должен уметь:

– использовать основные численные методы решения математических задач;
– выбирать оптимальный численный метод для решения поставленной задачи;
– давать математические характеристики точности исходной информации и оценивать точность полученного численного решения;
– разрабатывать алгоритмы и программы для решения вычислительных задач, учитывая необходимую точность получаемого результата.

В результате освоения дисциплины обучающийся должен знать:

– методы хранения чисел в памяти электронно-вычислительной машины (далее – ЭВМ) и действия над ними, оценку точности вычислений;
– методы решения основных математических задач – интегрирования, дифференцирования, решения линейных и трансцендентных уравнений и систем уравнений с помощью ЭВМ.

Каждая лабораторная работа имеет следующую структуру: тема, цели, краткие теоретические сведения, порядок проведения работы, требования к составлению отчета.
После выполнения лабораторной работы студент должен представить отчет о проделанной работе. Оценку по практической работе студент получает, если студентом работа выполнена в полном объеме, студент может пояснить выполнение любого этапа работы, отчет выполнен в соответствии с требованиями к выполнению работы, студент отвечает на контрольные вопросы на удовлетворительную оценку и выше.
Зачет по выполнению лабораторных работ студент получает при условии выполнения всех предусмотренных программой лабораторных работ с отчетами по всем работам.

Отчет к лабораторной работе должен содержать:
- Тему работы

- Задание для выполнения, включая индивидуальное задание

- Описание алгоритма программы, (при необходимости - со схемой алгоритма)
- Описание переменных и структур данных, которые применяются в программе
- Описание ключевых программных решений, принятых при реализации алгоритма в тексте программы
- Текст программы
- Замечания к отладке программы
- Образец результатов программы
- Выводы

Практическая работа №1
Программирование интерполяционных алгоритмов

Цель работы

1. Освоить структуру Windows-приложения в C# и организацию простейшего интерфейса для приложения с использованием объектов; однострочный и многострочный редакторы, метки для текстовых пояснений и управляющая командная кнопка.
2. Изучить, запрограммировать и отладить в C# интерполяционный алгоритм для приближения функции.
3. Освоить программирование алгоритма вычисления многочленов по схеме Горнера. Уметь проверить получаемые результаты.

1. 1. Краткие сведения из теории

1.1.1. Понятия аппроксимации и интерполяции

На практике распространенным является случай, когда вид связи между аргументом x и значением функции $y = f(x)$ неизвестен, а имеется задание этой связи в виде таблицы. Это означает, что дискретному множеству значений аргумента x_i поставлено в соответствие множество значений y_i (i =0,1,2,... n). Эти значения – либо результаты расчетов, либо – экспериментальные данные. Могут понадобиться значения величины y и в других точках, отличных от узлов x_i. Этой цели и служит задача о приближении (аппроксимации) функции: данную функцию $f(x)$, заданную таблично, требуется заменить (аппроксимировать) некоторой функцией $\varphi(x)$ так, чтобы отклонение (в некотором смысле) $\varphi(x)$ от $f(x)$ в заданной области было наименьшим. Функция $\varphi(x)$ при этом называется аппроксимирующей.

Весьма важен случай аппроксимации многочленом

$$\varphi(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_i x^i + \ldots + a_m x^m, \quad i = 0, 1, 2, \ldots, n \qquad (1.1)$$

При этом коэффициенты a_i подбираются так, чтобы достичь наименьшего отклонения от истинной функции. Что касается самого понятия “малое отклонение” – это зависит от конкретного способа аппроксимации.

Если приближение строится на заданном дискретном множестве точек x_i, то аппроксимация называется точечной. Одним из видов точечной аппроксимации является интерполирование. Оно состоит в следующем: для данной функции $y = f(x)$ строится многочлен (1.1), принимающий в заданных точках x_i те же значения y_i, что и функция $f(x)$, т.е.

$$\varphi(x_i) = y_i, \quad i = 0,1,2,\ldots,n \qquad (1.2)$$

При этом предполагается, что среди значений x_i нет одинаковых. Точки x_i называются узлами интерполяции, а многочлен $\varphi(x)$ – интерполяционным.

Таким образом, близость интерполяционного многочлена к заданной функции состоит в том, что их значения совпадают на заданной системе точек.

Максимальная степень интерполяционного многочлена $m = n$. В этом случае говорят о глобальной интерполяции, т.к. один многочлен используется для интерполяции функции $f(x)$ на всем рассматриваемом интервале изменения аргумента x. Интерполяционные многочлены могут так же строится отдельно для разных частей рассматриваемого интервала $[x_0, x_n]$. В этом случае имеем локальную интерполяцию.

Простейший вид локальной интерполяции - линейная интерполяция. Она состоит в том, что заданные точки (x_i, y_i) ($i = 0,1,2,\ldots, n$) попарно соединяют прямолинейными отрезками, и функция $f(x)$ приближается ломанной с вершинами в данных точках. Так как имеется n интервалов (x_{i-1}, x_i), то для любого из них существует своя прямая, проходящая через эти точки, которая описывается следующим образом:

$$y = ax + b, \quad x_{i-1} \le x \le x_i \tag{1.3}$$

Из условия прохождения этой прямой через заданные две точки (два узла x_{i-1}, x_i) находим коэффициенты a и b, определяющие конкретный отрезок ломанной:

$$\begin{cases} y_{i-1} = ax_{i-1} + b \\ y_i = ax_i + b \end{cases}$$
$$a = (y_i - y_{i-1})/(x_i - x_{i-1}) \tag{1.4}$$
$$b = y_{i-1} - ax_{i-1}$$

(1.5)

Следовательно, при использовании линейной интерполяции нужно сначала определить интервал, в который попадает значение аргумента x, а затем подставить его в формулы (1.3) – (1.5) и найти приближенное значение функции в искомой точке.

Используя большее число соседних точек и аппроксимируя истинную кривую более сложной линией, можно уточнить полученный результат. Перейдем к случаю глобальной интерполяции, т.е. к построению интерполяционного многочлена единого для всего отрезка интерполирования $[x_0, x_n]$. Существуют различные методы отыскания такого многочлена: методы Лагранжа, разностные и т. д.

Случай глобальной интерполяции реализует интерполяционный многочлен Лагранжа:

$$L(x) = \sum_{i=0}^{n} y_i \frac{(x - x_0)(x - x_1)....(x - x_{i-1})(x - x_{i+1})...(x - x_n)}{(x_i - x_0)(x_i - x_1)...(x_i - x_{i-1})(x_i - x_{i+1})...(x_i - x_n)} \tag{1.6}$$

Из формулы (1.6) легко получить различные частные случаи, например, для квадратичной интерполяции, т.е. для случая использования трех узлов, когда многочлен Лагранжа имеет вид:

$$L(x) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \tag{1.7}$$

Точность интерполяции по формуле Лагранжа оценивается остаточным членом многочлена Лагранжа $R_L(x)$:

$$R_L(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_n)}{(n-1)!} f^{(n+1)}(x_*) .$$

(1.8)

Здесь (x_*) – производная $(n+1)$-го порядка функции $f(x)$ в некоторой точке $x = x_*$, $x_* \in [x_0, x_n]$.

Блок-схема, соответствующая линейной интерполяции в явном виде (1.3) - (1.5), приведена ниже (рис.1.1).

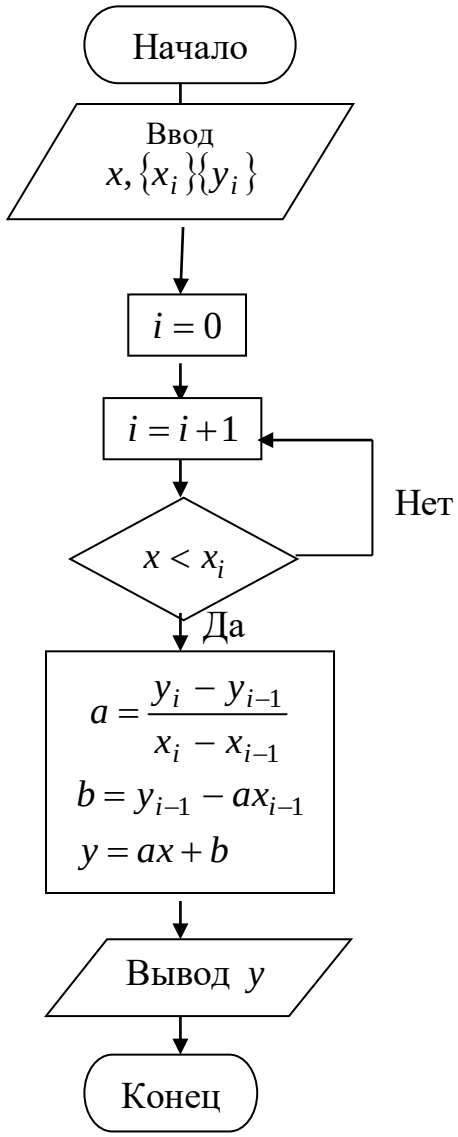


Рис.1.1. Блок-схема линейной интерполяции

1.1.2. Вычисление многочленов по схеме Горнера

При аппроксимации функции, а также в других задачах приходится вычислять значения многочлена (1.1). Если производить вычисления в “лоб”, то при больших степенях многочлена потребуется выполнить огромное число операций и, кроме того, это ведет к потере точности за счет погрешностей округлений. Устранить эти два недостатка позволяет вычисление по схеме Горнера:

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n = \\ = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + a_n x))\ldots) \quad . \quad (1.9)$$

$\overbrace{\qquad\qquad}^{n \text{ скобок}}$

Заметим, что формула (1.9) значительно сокращает объем вычислений потому, что каждый последующий член здесь выражен через предыдущий, тем самым значительно сокращается количество операций – умножений. При этом такая задача легко программируется (см. блок-схему рис.1.2).

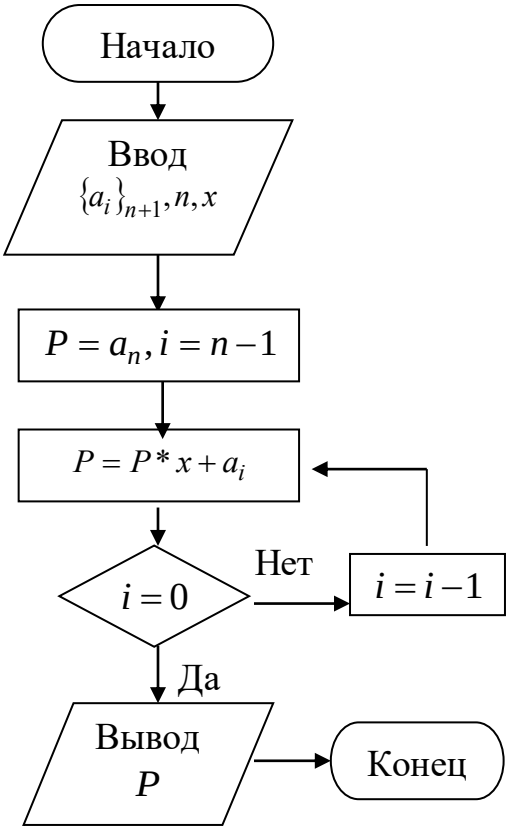


Рис.1.2. Блок-схема метода Горнера

1.2. Выполнение работы

Исходные данные для выполнения работы

Задача 1

Написать программу для аппроксимации функции, заданной в виде таблицы (таблице 1.1) и представленной в описании работы, в соответствии с условием своего варианта (таблице 1.2 – 1.4). Оформить интерполяционный метод как процедуру-реакцию на событие *onClick* в Windows-приложении. Для организации ввода исходных данных функции, заданной в виде таблицы, использовать объекты класса *TMemo*.

Произвести отладку программы.

Таблица 1.1

x	0	0.2	0.4	0.6	0.8	1.0	1.2
y	1.763	1.917	2.143	2.362	2.601	3.001	3.477

Таблица 1.2

Вариант	Аргумент функции *x*	Вид интерполяции
1	0.25	линейная (в явном виде)
2	0.65	линейная по Лагранжу
3	0.65	квадратичная по Лагранжу
4	0.93	квадратичная по Лагранжу
5	(0.15, 0.25, 0.5, 0.98)	квадратичная по Лагранжу
6	(0.15, 0.25, 0.5, 0.98)	линейная по Лагранжу
7	(0.15, 0.25, 0.5, 0.98)	линейная (в явном виде)
8	(0.34, 0.65, 0.82)	линейная по Лагранжу
9	(0.34, 0.65, 0.82)	квадратичная по Лагранжу

Таблица 1.3

x	0	0.3	0.6	0.9	1.2	1.5	1.8	2.1
y	1.763	1.936	2.264	2.561	2.758	3.067	3.298	3.401

Таблица 1.4

Вариант	Аргумент функции x	Вид интерполяции
1	0.78	квадратичная по Лагранжу
2	0.62	линейная в явном виде ($y = ax + b$)
3	0.06, 0.25, 0.39, 0.64	интерполяционная формула Лагранжа
4	0.25, 0.37, 0.48, 0.99, 1.06	квадратичная по Лагранжу
5	0.35	интерполяционная формула Лагранжа
6	0.801	квадратичная по Лагранжу
7	0.25, 0.37, 0.78, 0.99, 1.26	линейная в явном виде ($y = ax + b$)
8	0.78	линейная по Лагранжу

Задача 2

Написать программу (Windows-приложение в C#) для вычисления полинома по схеме Горнера (см. свой вариант таблице 1.5).

Отладить программу, используя тестовый пример.

Таблица 1.5

№ п/п	Многочлен
1	$1.723x^6 + 0.137x^4 - 0.814x^3 - 0.176x + 3.962$
2	$1.514x^7 - 0.124x^4 - 0.548x^3 + 3.214x^2 - 1.124x + 2.258$
3	$0.853x^5 - 0.143x^3 + 1.217x^2 - 2.243x + 2.415$
4	$1.116x^8 + 0.127x^6 + 1.164x^5 - 2.273x - 1.123$
5	$0.374x^5 + 0.242x^4 - 1.413x^3 + 0.746x^2 + 3.183x - 0.678$
6	$1.316x^6 - 0.144x^4 - 0.572x^3 + 1.854x^2 - 2.713x + 1.625$
7	$0.613x^7 + 0.318x^5 - 1.216x^3 + 2.517x^2 - 3.712x + 0.454$
8	$0.278x^6 - 0.763x^4 + 1.072x^3 + 1.613x^2 - 2.312x - 1.418$
9	$1.172x^5 - 0.534x^4 + 1.283x^3 + 1.615x - 2.652$
10	$1.087x^8 - 1.243x^7 + 0.656x^5 - 0.783x^2 + 2.574x + 0.564$
11	$0.475x^6 + 0.612x^5 + 1.134x^3 + 1.183x^2 - 3.154x + 0.844$
12	$0.683x^7 + 0.514x^5 - 0.817x^3 + 2.432x^2 + 1.072x - 0.833$
13	$1.028x^6 - 0.731x^4 - 1.072x^3 + 1.625x^2 - 3.184x - 1.546$
14	$0.243x^5 - 1.065x^4 - 0.364x^3 + 2.445x^2 - 1.265x + 0.318$
15	$0.831x^9 - 0.722x^7 + 1.157x^5 + 1.615x^2 - 2.844x - 0.685$
16	$0.354x^5 + 0.583x^4 - 1.072x^3 + 1.548x^2 - 0.367$
17	$1.273x^7 + 0.172x^5 - 0.788x^3 + 1.453x^2 - 2.813x + 3.154$
18	$0.421x^5 - 0.544x^4 - 1.213x^3 + 0.683x^2 + 3.145x - 0.185$
19	$1.342x^8 - 0.254x^6 + 0.872x^5 + 1.273x^2 - 1.483x + 0.584$
20	$1.418x^6 - 1.547x^5 + 0.418x^3 + 1.783x^2 - 2.517x + 2.434$

Произвести счет программ и оформить работу. Уметь доказать правильность полученных результатов. Защитить работу преподавателю.

1.3. Контрольные вопросы

1. Объясните назначение свойства *Text* для однострочного и многострочного редактора. Как можно изменить значение этого свойства?
2. Как можно получить заготовку *C#* на процедуру, как реакцию на событие *onClick* для объекта *Batton1*?
3. Сформулируйте постановку задачи аппроксимации функций.
4. Понятие интерполяции, как вида точечной аппроксимации. Основное отличие от других способов аппроксимации функций.
5. Каким образом можно повысить точность производимой аппроксимации?
6. Как осуществить глобальную интерполяцию заданной табличной функции?
7. С какой целью при программировании вычислений значений полиномов применяют схему Горнера?

Практическая работа № 2

Программирование алгоритмов решения дискретных моделей на примерах задач численного интегрирования

Цель работы

1. Освоить понятие сходимости в методах дискретизации. Запрограммировать и отладить алгоритм для вычисления определенного интеграла с заданной точностью.
2. Изучить работу с графическим компонентом Image и объединяющей панелью Panel , применить их в своем приложении.
3. Научиться использовать блоки *try…except*, *try…finally* для защиты фрагментов программ.

2.1. Краткие сведения из теории

Численное интегрирование применяют в тех случаях, когда интеграл не удается вычислить в аналитическом виде или когда этот вид достаточно сложен. А также, когда нужно найти интеграл от табулированной функции, получаемой в эксперименте.

В общем виде задача состоит в нахождении величины

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

Универсальные методы интегрирования основаны на аппроксимации подынтегральной функции $f(x)$, с помощью интерполяционных многочленов.

В этом случае определенный интеграл может быть представлен в форме:

$$\int_a^b f(x)dx = \sum_{i=0}^{n} A_k f(x_k) + R,$$

где x_k – узлы интерполяции;

A_k – коэффициенты, зависимые от смысла формулы и выбора узлов;

R – погрешность формулы.

Чтобы не иметь дело с полиномами высоких степеней, строят интерполяционные полиномы не для всего отрезка $[a,b]$, а для отдельных его частей. На каждой части функция $f(x)$ может быть заменена полиномами нулевой (формула прямоугольников), первой (формула трапеций), второй (формула Симпсона) степени. Величина интеграла на отрезке $[a,b]$ получается суммированием результатов, вычисленных для всех частей деления $[a,b]$.

2.1.1. Метод прямоугольников

По методу прямоугольников кривая подынтегральной функции заменяется ломанной линией, отрезки которой параллельны оси абсцисс. В данном методе

используется кусочно-постоянное интерполирование. Формула метода прямоугольников для случая левых прямоугольников:

$$\int_a^b f(x)dx = h_1 y_0 + h_2 y_1 + h_3 y_2 + ... + h_n y_{n-1} = h\sum_{k=0}^{n-1} y_k, \quad \text{где } h_i = const = h.$$

Формула для случая правых прямоугольников:

$$\int_a^b f(x) = h_1 y_1 + h_2 y_2 + ... + h_n y_n = h\sum_{k=1}^{n} y_k, \quad \text{где } h_i = const = h.$$

Формула легко программируется. Точность метода порядка h, где $h << 1$. Широко распространенным и более точным является вид формулы прямоугольников, использующий функции в средних точках элементарных отрезков (в полуцелых узлах). Формула интегральной суммы по методу средних точек имеет вид:

$$\int_a^b f(x)dx \approx h\sum_{i=1}^{n} f(x_{i-1/2}) = h\left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_{i-1/2})\right), \qquad (2.1)$$

где $x_{i-1/2} = \frac{x_{i-1} + x_i}{2} = x_{i-1} + \frac{h}{2}$, $i = 1,2,...,n$.

В дальнейшем под методом прямоугольников будем понимать последний алгоритм.

Погрешность метода прямоугольников:

$R_n \approx \frac{1}{24} h_i^3 f''(x_{i-1/2})$, где f'' – максимальная производная второго порядка.

2.1.2. Метод трапеций

Метод трапеций использует линейную интерполяцию, т.е. график функции $y = f(x)$ представляется в виде ломаной, соединяющей точки (x_i, y_i). В этом случае площадь всей фигуры складывается из прямоугольных трапеций. Площадь каждой такой трапеции равна произведению полусуммы оснований на высоту:

$$S_i = \frac{y_{i-1} + y_i}{2} h_i, \quad i = 1,2,...,n.$$

Складывая все S_i, получаем формулу трапеций для численного интегрирования:

$$\int_a^b f(x)dx = h\left(\frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i\right) = h\left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right) \qquad (2.2)$$

Легко видеть, что разница между формулами трапеций и средних лишь в выборе узлов, в которых вычисляются значения функции. Отсюда, и точность метода трапеций того же порядка (h^3), что и у метода средних. Погрешность метода трапеций:

$$R_n = -\frac{1}{12} h^3 f''(x_i),$$

где $f''(x_i)$ – максимальное значение производной второго порядка.

2.1.3. Метод Симпсона

В этом методе кривая подынтегральной функции заменяется кусочно-непрерывной линией, состоящих из отрезков квадратичных парабол, следовательно, в методе используется квадратичная интерполяция для аппроксимации подынтегральной функции:

$$f(x) \approx \varphi_i(x) = a_i x^2 + b_i x + c_i \quad , \quad x_{i-1} \le x \le x_{i+1}.$$

В качестве $\varphi_i(x)$ можно принять интерполяционный многочлен Лагранжа второй степени, проходящий через точки M_{i-1}, M_i, M_{i+1}:

$$\varphi_i(x) = \frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} \cdot y_{i-1} + \frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})} \cdot y_i + \frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)} \cdot y_{i+1}.$$ Эле

ментарная площадь S_i может быть вычислена с помощью определенного интеграла. Учитывая равенство $x_{i+1} - x_i = x_i - x_{i-1} = h$, получаем:

$$S_i = \int_{x_{i-1}}^{x_{i+1}} \varphi_i(x)dx = \frac{1}{2h^2} \int_{x_{i+1}}^{x_{i+1}} [(x-x_i)(x-x_{i+1})y_{i-1} - 2(x-x_{i-1})y_i +$$

$$+(x-x_{i-1})(x-x_i)y_{i+1}]dx = \frac{h}{3}(y_{i-1} + 4y_i + y_{i-1}).$$

Такие вычисления проводятся для каждого элементарного отрезка (x_{i-1}, x_{i+1}).

Просуммируем полученные выражения:

$$S = \sum_{i=1}^{n} S_i = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + ... + 2y_{n-2} + 4y_{n-1} + 4y_n).$$

Полученное выражение S принимается в качестве значения определенного интеграла.

$$\int_a^b f(x)dx \approx \frac{h}{3}[y_0 + y_n + 4(y_1 + y_3 + ... + y_{n-1}) + 2(y_2 + y_4 + ... + y_{n-2})] \qquad (2.3)$$

Данное соотношение и есть формула Симпсона.

Метод Симпсона обладает более высокой точностью по сравнению с методами трапеций и прямоугольников.

$$R_n = -\frac{h^4}{180} f^4(x),$$

где $f^4(x)$ – наибольшее по абсолютной величине значение четвертой производной функции, которая интегрируется. Оценить четвертую производную можно через четвертые разности функции: $f^4(x) \approx \frac{\Delta^4 y}{h^4}$.

Блок-схема метода средних с фиксированным числом разбиений приведена на рис. 2.1.

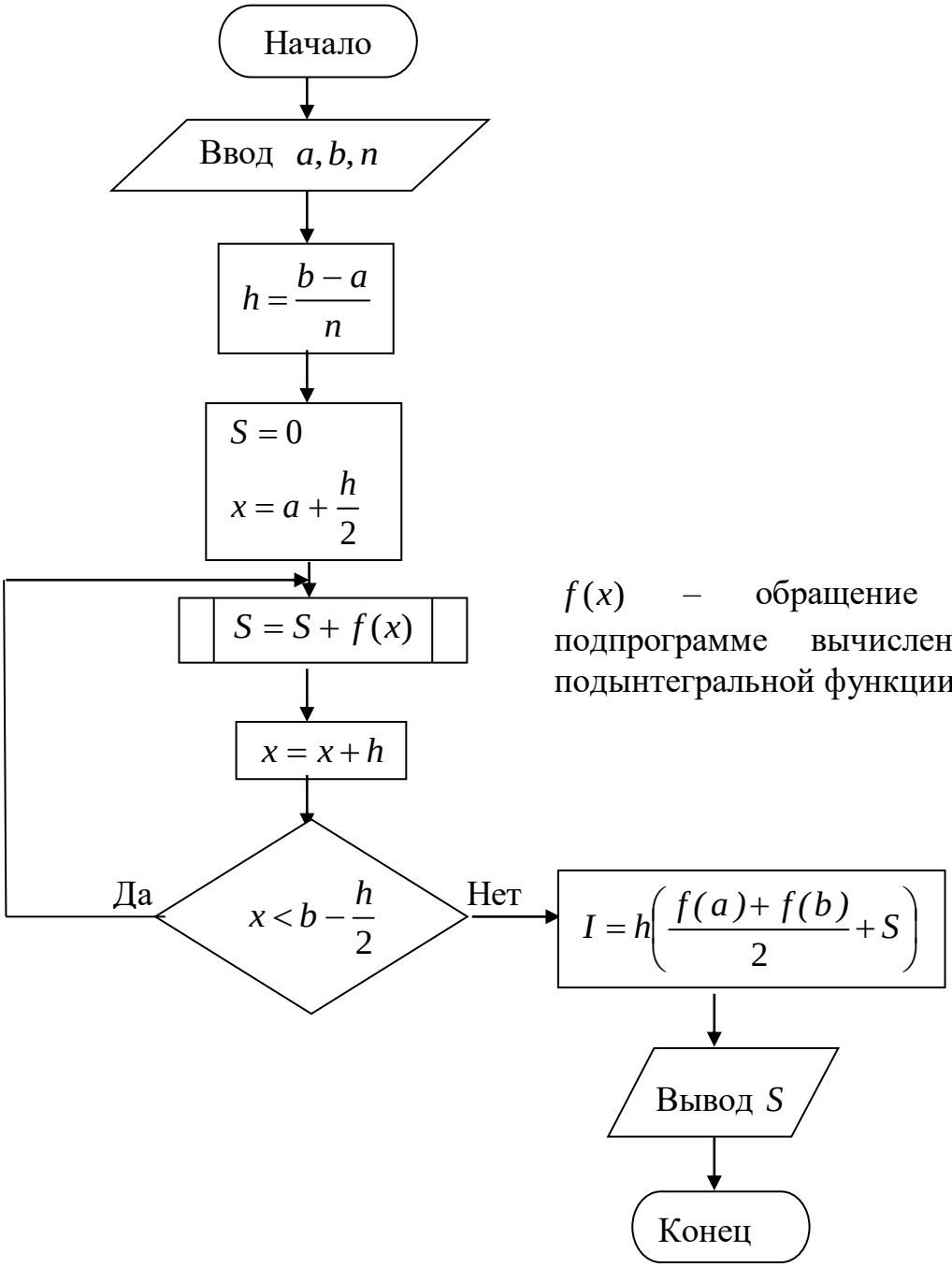


Рис.2.1. Блок-схема метода средних с фиксированным числом разбиений.

2.1.4. Алгоритм автоматического выбора шага

Этот алгоритм используется для достижения необходимой точности (ε) при вычислении определенных интегралов. Повышение точности происходит за счет уменьшения шага интегрирования на каждой новой итерации вычисления значения интеграла определенным численным методом. Блок-схема алгоритма приведена на рис.2.2.

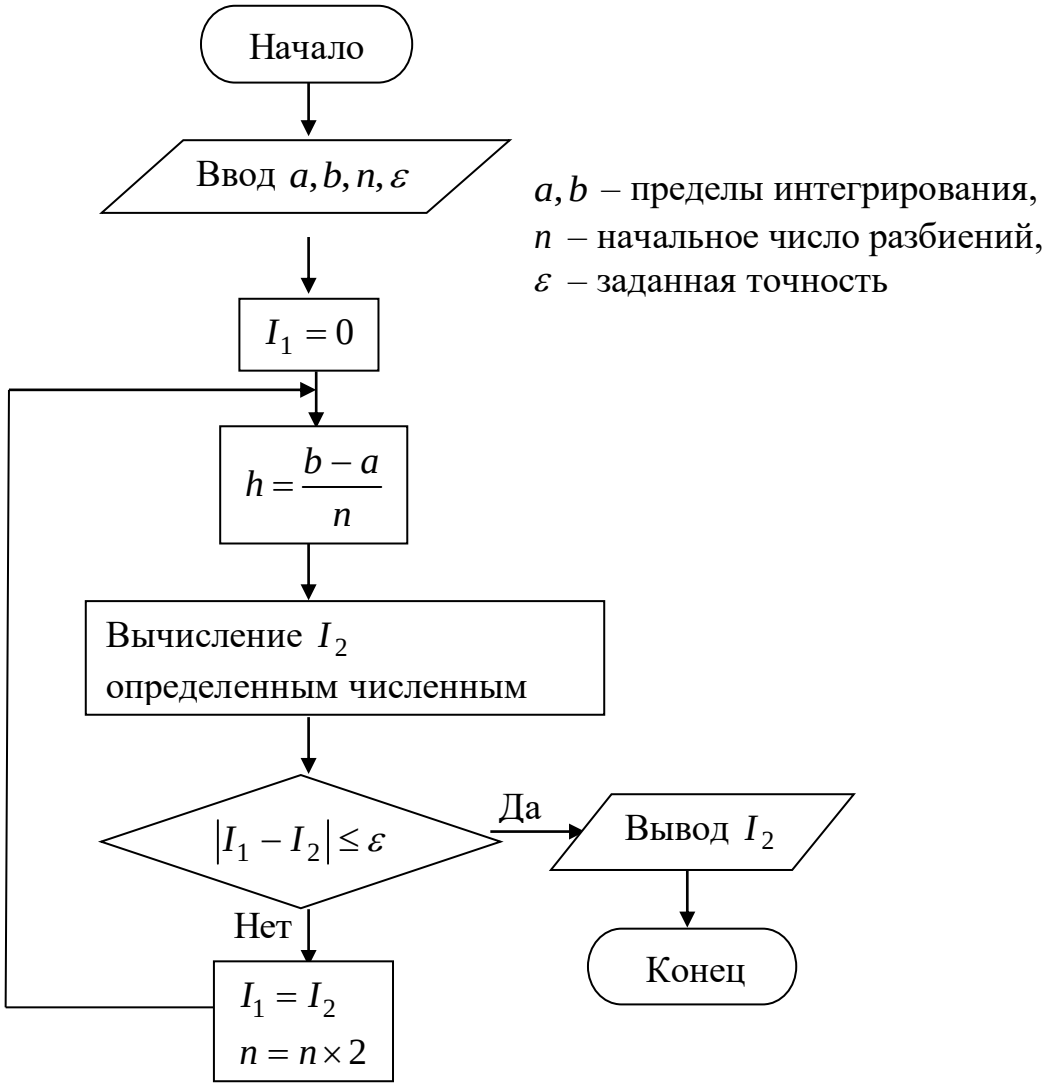


Рис. 2.2. Блок-схема алгоритма автоматического выбора шага.

2.2. Выполнение работы

Порядок выполнения работы

1. Написать программу вычисления определенного интеграла в соответствии с номером варианта для своей задачи.
2. Объединить на форме элементы ввода с помощью объекта класса *TPanel*.

3. Отобразить на форме условие своего примера, применив объект класса *TImage*.
4. Применить в программе блок защиты для операторов итерационного цикла достижения точности.
5. Отладить программу, используя тестовые примеры.
6. Оформить отчет и защитить работу преподавателю.
7. Сделать контрольный просчет по отлаженной программе в соответствии с заданием.

Исходные данные для выполнения работы

Вычислить определенный интеграл

$$\frac{1}{2\pi}\int_{-1}^{2} e^{-x^2}\,dx.$$

Варианты заданий помещены в таблице 2.1.

Таблица 2.1

Вариант	Метод	Кол-во интервалов разбиения	Точность
1	прямоугольников (в целочисленных узлах)	автоматический выбор шага	10-3
2	прямоугольников (в целочисленных узлах)	50	—
3	прямоугольников (в целочисленных узлах)	автоматический выбор шага	10-3
4	трапеций	автоматический выбор шага	10-4
5	трапеций	автоматический выбор шага	10-3
6	трапеций	автоматический выбор шага	10-4
7	прямоугольников (в полуцелых узлах)	автоматический выбор шага	10-4
8	прямоугольников (в полуцелых узлах)	автоматический выбор шага	10-2

Вычислить определенный интеграл методом Симпсона.

$$\int_{1}^{2} x^3 \cos^2 x\,dx$$

Варианты заданий помещены в таблице 2.2.

Таблица 2.2

Вариант	Кол-во интервалов разбиения	Точность
1	автоматический выбор шага	10-4
2	50	—
3	автоматический выбор шага	10-3
4	50	—
5	50	—
6	50	—
7	автоматический выбор шага	10-4
8	50	—

2.3. Контрольные вопросы

1. Как осуществить вывод изображения в объект класса *TImage*?
2. Назначение объектов класса *TPanel*.
3. Объясните понятие "сходимость в методах дискретизации".
4. Объясните назначение оператора ***on…*** и организацию правильной последовательности этих операторов в части ***except.*** блока защиты ***try…except***.
5. Дайте графическую интерпретацию каждого из предлагаемых в работе методов численного интегрирования.
6. Как влияет на точность численного интегрирования величина шага интегрирования?
7. Как влияет вид используемой интерполяции на точность получаемого результата?

Практическая работа № 3

Реализация в прямых и итерационных алгоритмов решения СЛАУ, их приложения

Цель работы

1. Освоить программирование с визуальными компонентами табличного представления данных (*StringGrid*) для организации ввода/вывода матричных структур.
2. Изучение прямых и итерационных методов решения систем линейных алгебраических уравнений на примере методов Гаусса и Гаусса-Зейделя.
3. Условия применения каждой группы методов.
4. Программирование методов. Составление тестовых примеров для отладки программ.

3.1. Краткие сведения из теории

3.1.1. Обзор методов решения систем решения линейных алгебраических уравнений

С точки зрения обычной математики система линейных алгебраических уравнений (далее – линейных уравнений) всегда является невырожденной (решение существует и оно единственное) или вырожденной (вообще не имеет решения или имеет бесчисленное множество решений). С точки зрения практических вычислений могут существовать почти вырожденные системы, при решении которых получаются недостоверные значения неизвестных (плохо обусловленные системы). Для таких систем найти численное решение трудно, а точность его весьма сомнительна.

Методы решения систем линейных уравнений делятся на две группы: прямые и итерационные. Прямые методы используют конечные формулы для вычисления неизвестных и дают решение после выполнения заранее известного числа операций (метод Гаусса). Итерационные методы – это методы последовательных приближений, позволяющие получать корни системы с заданной точностью путём сходящихся бесконечных процессов (метод Гаусса-Зейделя).

Прямые методы имеют характерные недостатки, что не всегда делает возможным их применение:

- они не учитывают структуру матрицы коэффициентов, что заставляет занимать значительное место в памяти машины нулевыми элементами;
- в процессе вычислений накапливается вычислительная погрешность, что опасно для больших систем, а также для плохо обусловленных систем.

В перечисленных случаях применяют итерационные методы. Эффективность применения итерационных методов от выбора начального приближения и быстроты сходимости процесса. Поэтому при применении

итерационного метода необходимо всегда учитывать условия его сходимости. Итерационные методы часто применяются для уточнения решений, полученных с помощью прямых методов. Эти смешанные алгоритмы особенно эффективны для плохо обусловленных систем.

Отметим, что метод Гаусса используется не только для решения систем линейных уравнений, но и для вычисления обратной матрицы и определителя прямой матрицы.

3.1.2. Метод Гаусса

Пусть дана система линейных алгебраических уравнений порядка n:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases} \qquad (3.1)$$

Необходимо решить систему (3.1), т.е. найти значение неизвестных $x_1, x_2, \ldots, x_n$.

Метод основан на приведении данной системы к треугольному виду. Это достигается последовательным исключением неизвестных из уравнений системы. Сначала с помощью первого уравнения исключается x_1 из всех последующих уравнений системы. Затем с помощью второго уравнения исключается x_2 из третьего и всех последующих уравнений. Этот процесс называется прямым ходом Гаусса. Он продолжается до тех пор, пока в левой части последнего (n-ого) уравнения не останется лишь один член с неизвестным x_n. Необходимо помнить, что к такому виду приводится лишь невырожденная матрица.

Обратный ход метода состоит в последовательном вычислении искомых неизвестных: решая последнее уравнение, находят единственное неизвестное x_n. Далее, используя найденное значение x_n, из предыдущего уравнения вычисляют x_{n-1} и т.д. последним вычисляют x_1 из первого уравнения.

Данный метод предлагаем дополнять алгоритмом поиска ненулевого ведущего элемента или поиска главного элемента.

Вычисление определителя

Легко вычисляется определитель треугольной матрицы: он равен произведению ее диагональных элементов.

Для приведения матрицы к прямоугольному виду используем прямой ход метода Гаусса. Знак определителя меняется на противоположный при перестановке его строк или столбцов. Значение определителя после приведения матрицы A к треугольному виду вычисляется по формуле:

$$\det A = \pm \prod_{k=1}^{n} a_{kk} .$$

Здесь элементы a_{kk} берутся из преобразованной треугольной матрицы. Знак зависит от того, чётной или нечётной была суммарная перестановка строк матрицы при её приведении к треугольному виду (для получения ненулевого ведущего элемента на каждом этапе исключения).

Комментарий к блок-схеме метода Гаусса с поиском ненулевого ведущего элемента

Исходными данными при составлении алгоритма являются порядок системы n и массив действительных переменных – коэффициентов и свободных членов матрицы A, состоящий из n строк и $n+1$ столбцов, причём свободные члены b_k размещаются в конце каждой k-ой строки, т.е. обозначаются как элементы массива A индексами ($k, n+1$).Промежуточные и окончательные значения элементов в процессе преобразования матрицы располагаются и том же массиве. Значения диагональных элементов матрицы коэффициентов перед началом преобразования строк, присваивается промежуточной переменной C и сохраняется до окончания преобразования строки.

Результаты вычисления неизвестных x_k накапливаются в одномерном массиве X , содержащем n элементов.

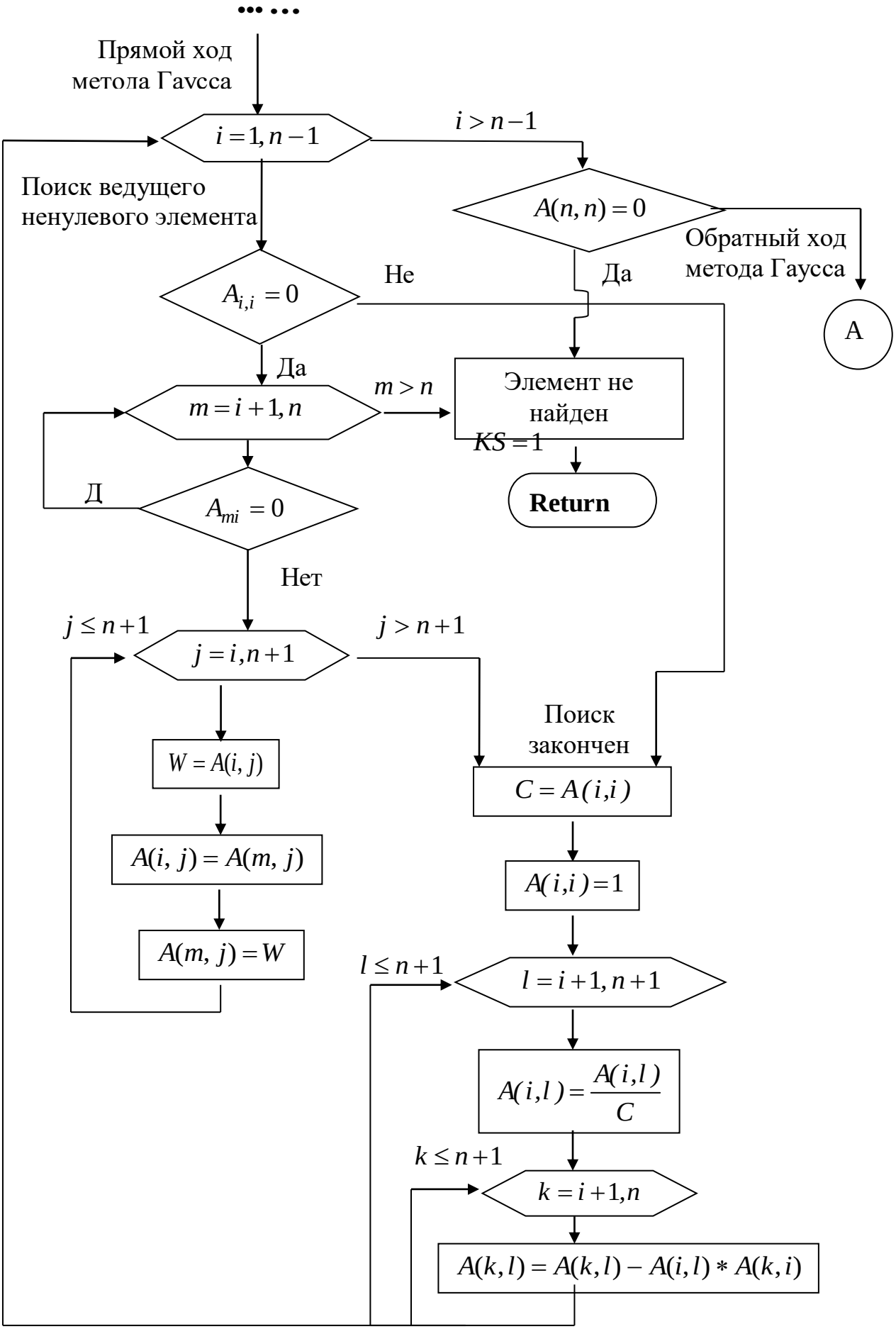


Рис. 3.1. Блок-схема прямого хода метода Гаусса с поиском ненулевого ведущего элемента

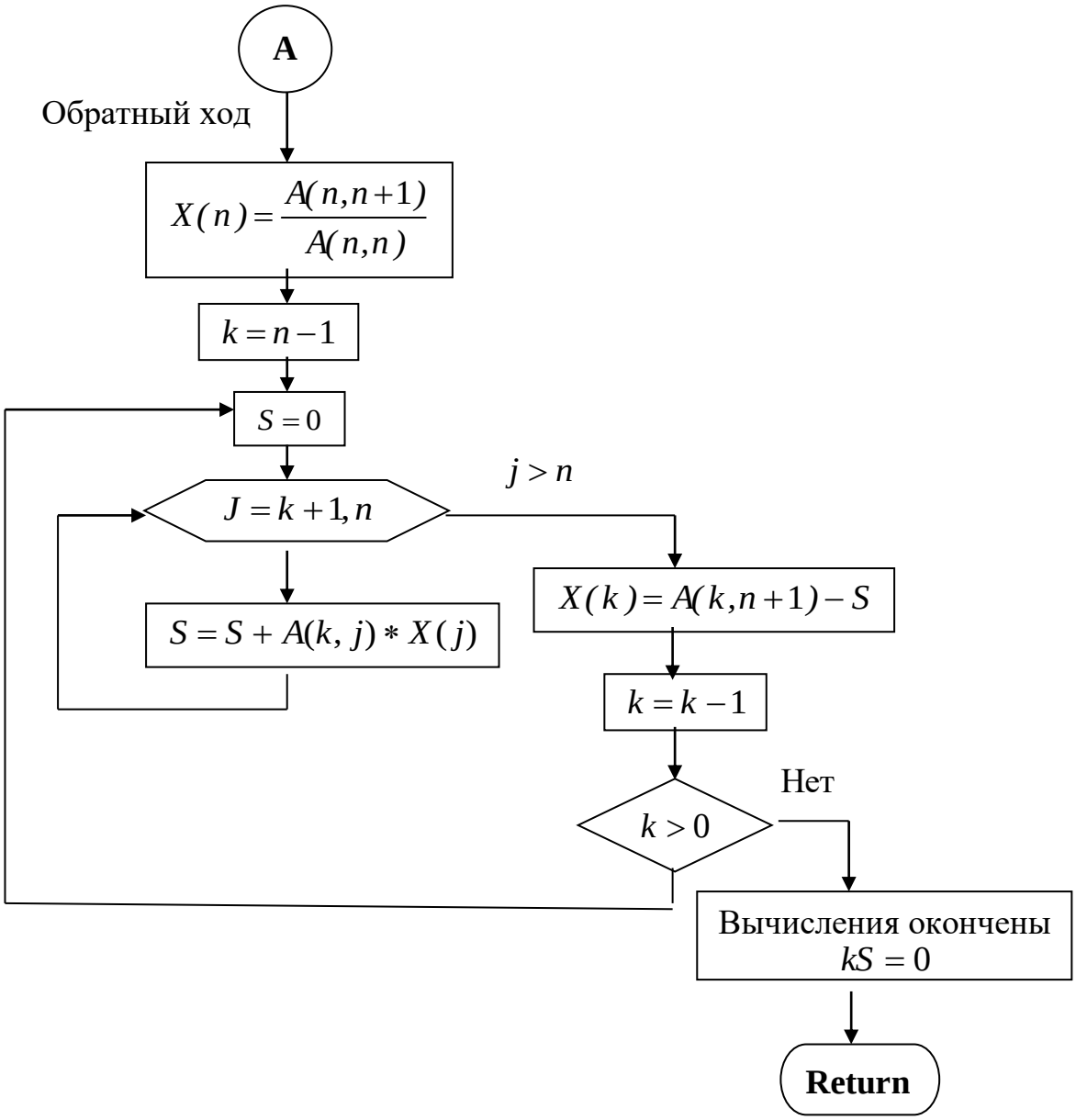
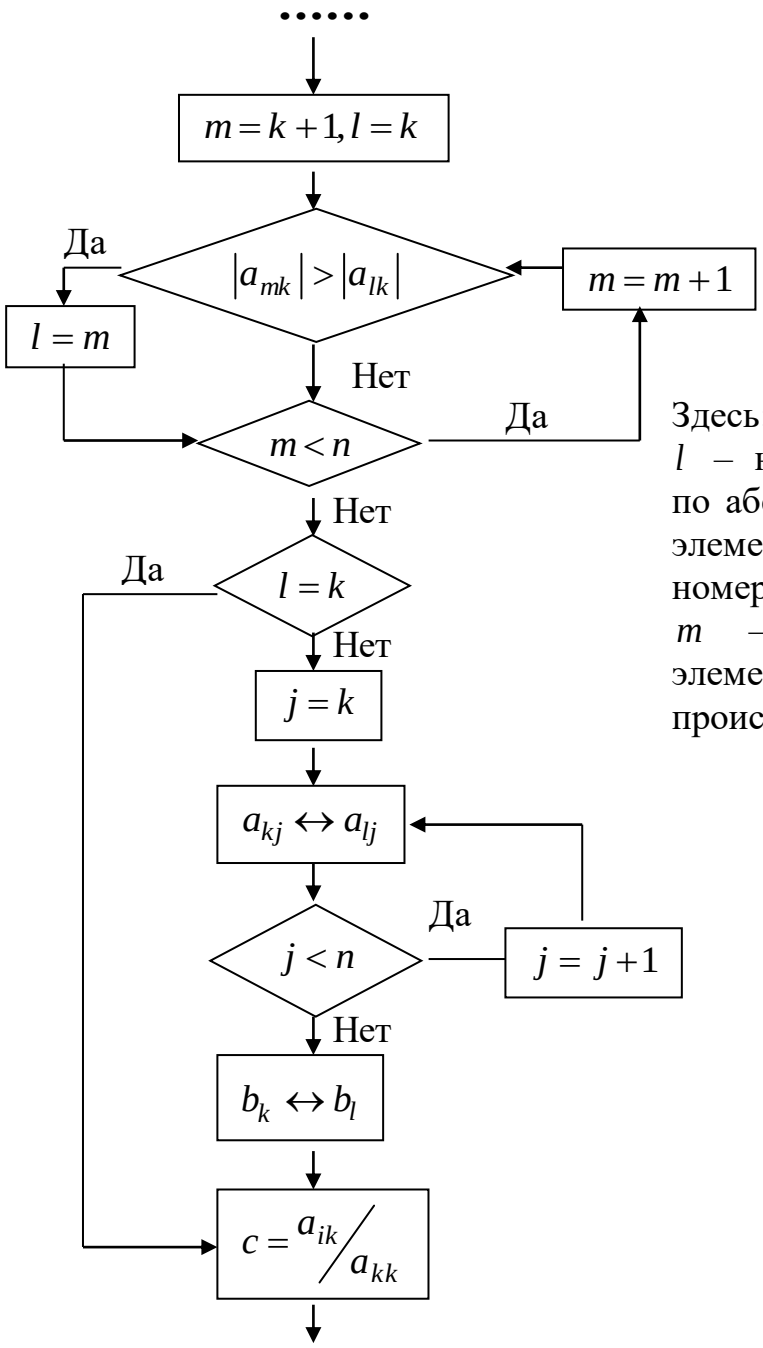


Рис. 3.2. Блок-схема обратного хода метода Гаусса

Блок-схема алгоритма выбора главного элемента

Эта схема состоит в том, что требование неравенства нулю диагональных элементов a_{kk}, на которые происходит деление в процессе исключения, заменяется более жестким: из всех оставшихся в k-ом столбце элементов нужно выбрать наибольший по модулю и переставить уравнения так, чтобы этот элемент оказался на месте элемента a_{kk}.



Здесь:
l – номер наибольшего по абсолютной величине элемента матрицы с номером k,
m – текущий номер элемента, с которым происходит сравнение.

Рис.3.3. Блок-схема алгоритма выбора главного элемента

3.1.3. Метод Гаусса-Зейделя

Это наиболее распространённый из итерационных методов. В нём каждое приближение $x_i^{(k)}$ вычисляется по формуле:

$$x_i^{(k)} = \frac{1}{a_{ii}}(b_i - a_{i1}x_i^{(k)} - \ldots - a_{i,i-1}x_{i-1}^{(k)} - a_{i,i+1}x_{i+1}^{(k-1)} - \ldots - a_{in}x_n^{(k-1)}),\ i = 1,2,\ldots,n,$$

где i – номер корня,

n – порядок системы,

k – номер текущей итерации.

Количество итераций зависит от требуемой точности, т.е. итерационный процесс продолжаем до тех пор, пока все $x_i^{(k)}$ не станут достаточно близки к $x_i^{(k-1)}$. Критерий близости используется абсолютный или относительный.

Критерий близости можно, например, задать в следующем виде:

(1) $M^{(k)} = max\left|x_i^{(k)} - x_i^{(k-1)}\right| < \varepsilon$, где $\varepsilon > 0$ – заданная допустимая погрешность.

Здесь определяется максимальное значение разности для всех i. При выполнении критерия итерационный процесс следует остановить.

Этот критерий по абсолютным отклонениям можно заменить критерием по относительным разностям т.е. критерий близости будет выглядеть следующим образом:

(2) $$max\left|\frac{x_i^{(k)} - x_i^{(k-1)}}{x_i^{(k)}}\right| < \varepsilon$$

При выполнении либо условия (1), либо условия (2) итерационный процесс метода Гаусса-Зейделя называется сходящимся.

Достаточными условиями сходимости итерационного процесса по методу Гаусса-Зейделя являются следующие условия:

1. Модули диагональных коэффициентов для любого уравнения системы должны быть не меньше сумм модулей всех остальных коэффициентов: $|a_{ii}| \geq \sum_{i \neq j}|a_{ij}|$, $i = 1,2,\ldots,n$. При этом хотя бы для одного уравнения неравенство должно выполняться строго.
2. Система линейных уравнений должна быть неприводима.

Заметим, что диагональные коэффициенты a_{ii} должны быть отличными от нуля. Поэтому данный алгоритм необходимо дополнить алгоритмом поиска ненулевого ведущего элемента.

Блок-схема метода Гаусса-Зейделя

Блок-схема приведена на рис. 3.4 а. и 3.4 б. На первом рисунке изображены дейсствия, связанные с вводом исходных данных и подготовкой к вычислениям. Начальное приближение принимается равным нулю, а счётчику количества итераций (ITER) присваивается 1.

Переменная BIG используется для того, чтобы определять наибольшее значение разности между $x_i^{(k)}$ и $x_i^{(k-1)}$. Сначала этой переменной присваивается значение нуль, а затем с ней сравниваются абсолютные значения разностей $\left|x_i^{(k)} - x_i^{(k-1)}\right|$.

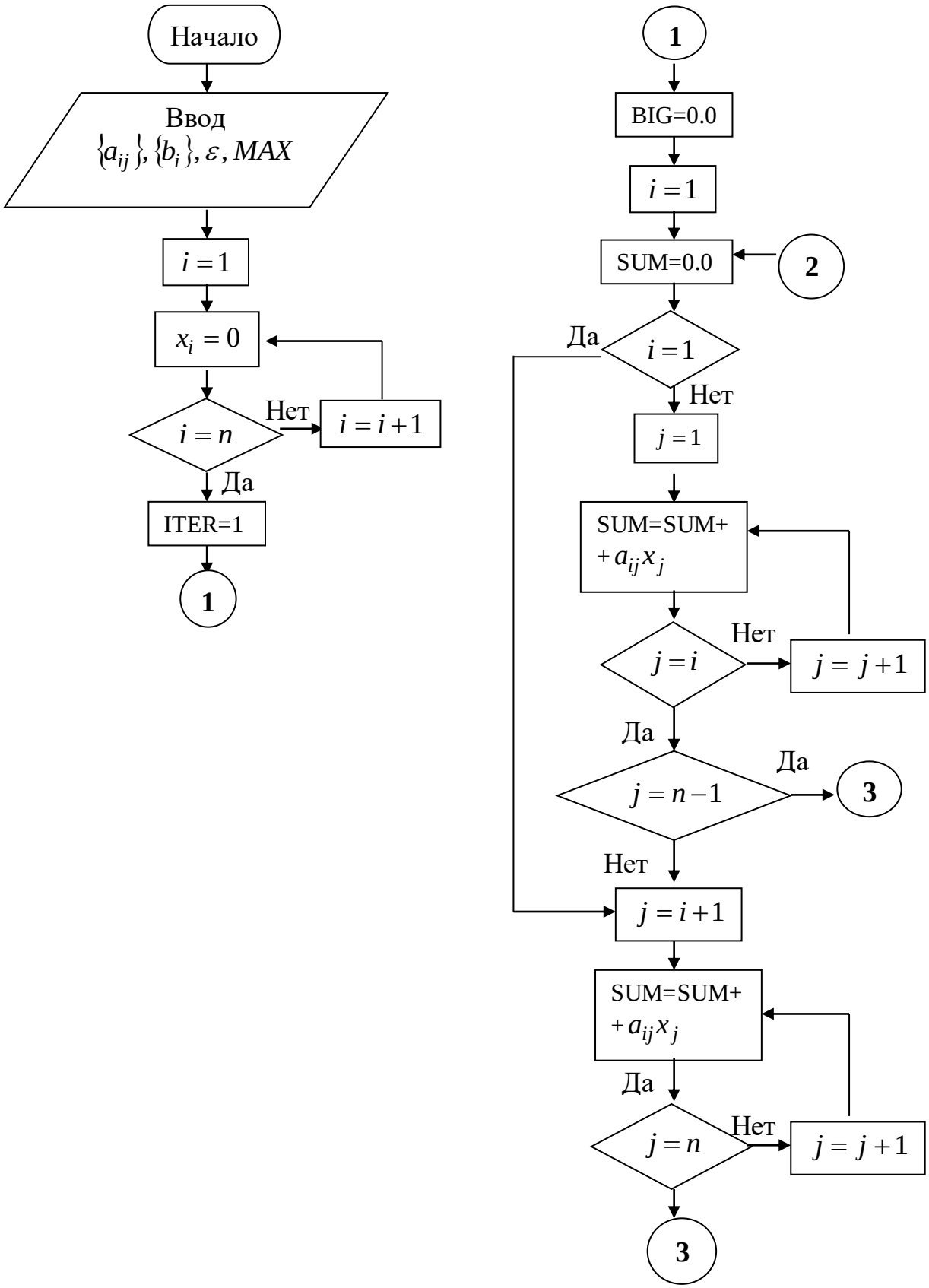


Рис. 3.4.а. Блок-схема метода Гаусса-Зейделя

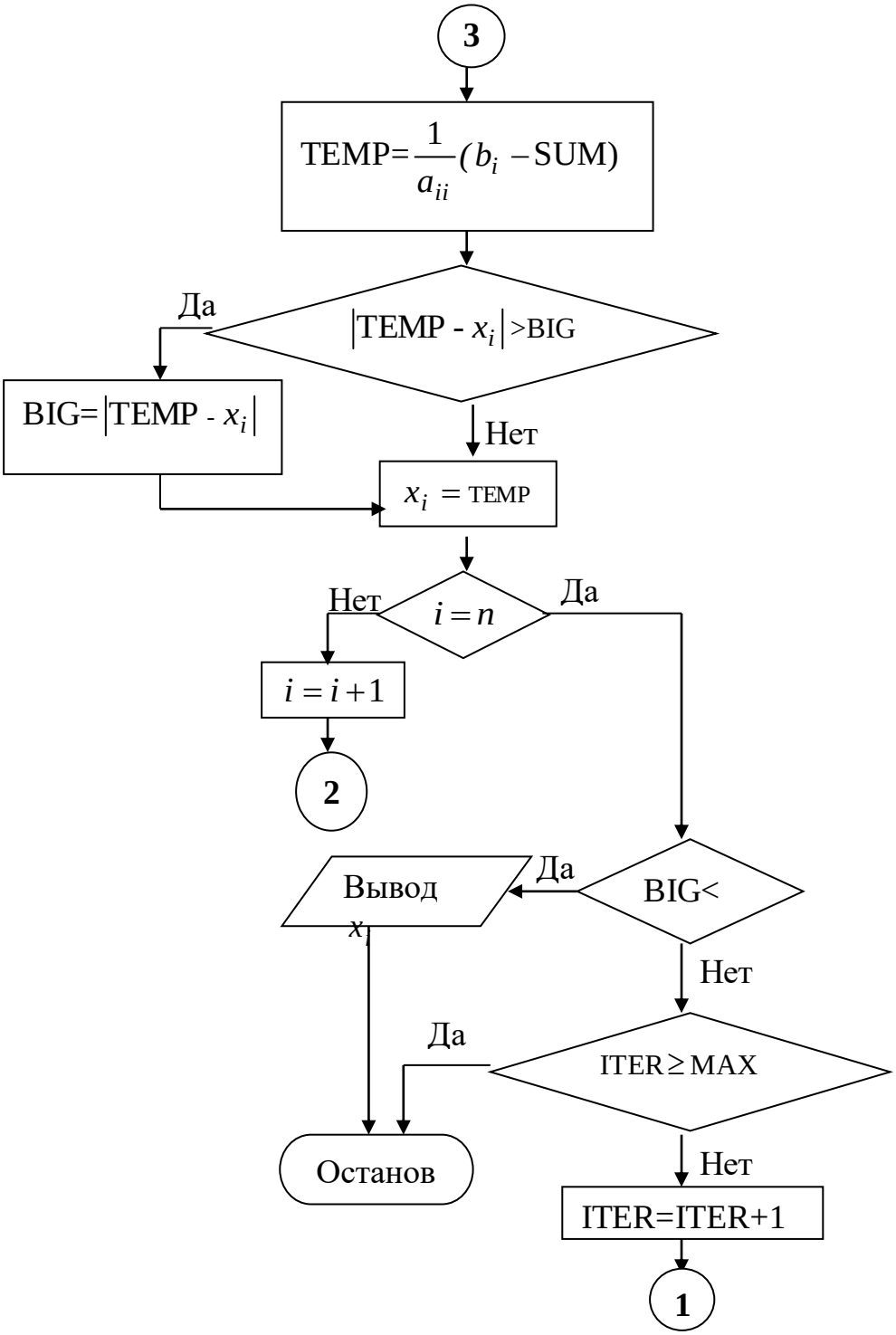


Рис.3.4.б. Блок-схема метода Гаусса-Зейделя (продолжение)

Если какая-либо разность оказывается по абсолютной величине больше BIG, то прежнее значение BIG заменяется этой разностью. После вычисления всех $x_i^{(k)}$ наибольшая разность будет равна значению переменной BIG.

Затем в блок-схеме следует группа действий, с помощью которых вычисляется сумма всех членов уравнения, кроме диагонального. По ходу программы удобнее сначала вычислять и суммировать члены, стоящие перед диагональным (т.к. в них используются новые значения $x_i^{(k)}$), а уже потом – члены, идущие после него (т.к. в них используются старые значения $x_i^{(k-1)}$).

Необходимо несколько усложнить логику, т.к. в первом и соответственно в последнем уравнениях отсутствуют элементы, стоящие соответственно до и после диагонального.

В правой части блок-схемы указана группа действий, начинающаяся с вычисления нового значения $x_i^{(k)}$; определяется максимальная разность и запоминается новое значение $x_i^{(k)}$, которое вычисляется под именем TEMP.

Если это было не последнее уравнение, то мы увеличиваем индекс *i* и начинаем вычислять очередное приближение для следующего неизвестного. В случае последнего уравнения мы сравниваем максимальную разность с ε и печатаем результаты, если процесс сошёлся.

Но процесс в действительности не всегда сходится. Причины могут быть различные – от ошибок в программе до неверных исходных данных. Поэтому в начале программы было введено целое число MAX, которое определяет максимально допустимое число итераций. (Ориентировочно MAX = 50 для системы из 50 уравнений.) Тогда, если по какой-то причине процесс не сошёлся, ЭВМ не будет считать бесконечно долго.

3.2. Выполнение работы

Порядок выполнения работы

1. Изучить методы Гаусса и Зейделя, научиться их программировать по предложенным блок-схемам.
2. Изучить алгоритм поиска ненулевого ведущего элемента.
3. Составить программу в соответствии с условием своего варианта. Использовать объект класса *TStringGrid* для ввода коэффициентов заданной СЛАУ.
4. Отладить программу на тестовом примере.
5. Произвести счёт для контрольного примера.
6. Оформить отчёт и ответить на контрольные вопросы.

Исходные данные для выполнения работы

Таблица 3.1

Вариант	Контрольная система	Применяемый метод
1	$\begin{cases} 0.64x_1 + 1.05x_2 - 2.93x_3 = 1.11 \\ 1.05x_1 - 1.41x_2 - 0.16x_3 = -0.27 \\ -2.93x_1 + 0.16x_2 - 1.51x_3 = 0.72 \end{cases}$	Гаусса для вычисления корней системы
2	См. вар.1	Гаусса для вычисления определителя
3	См. вар.1	Зейделя с точностью $\varepsilon = 10^{-3}$

Таблица 3.1

(продолжение)

Вариант	Контрольная система	Применяемый метод
4	$\begin{cases} 0.24x_1 - 0.02x_2 + 0.03x_3 - 0.01x_4 = 0.54 \\ 0.01x_1 + 0.22x_2 - 0.02x_3 - 0.01x_4 = -0.61 \\ 0.01x_1 + 0x_2 + 0.28x_3 - 0.02x_4 = 0.28 \\ 0x_1 + 0.01x_2 - 0.02x_3 + 0.20x_4 = -0.45 \end{cases}$	Гаусса для вычисления корней системы
5	См. вар.4	Гаусса для вычисления неизвестных и определителя
6	См. вар.4	Гаусса для вычисления определителя
7	См. вар.4	Зейделя с точностью $\varepsilon = 10^{-3}$
8	$\begin{cases} 0.10x_1 + 0.01x_2 - 0.02x_3 + 0.20x_4 = -0.45 \\ 0.24x_1 - 0.02x_2 + 0.02x_3 + 0.01x_4 = 0.54 \\ 0.01x_1 + 0.21x_2 + 0.02x_3 - 0.01x_4 = -0.61 \\ 0.01x_1 + 0.02x_2 + 0.28x_3 - 0.02x_4 = 0.28 \end{cases}$	Зейделя с точностью $\varepsilon = 10^{-3}$
9	См. вар.8	Гаусса для вычисления определителя
10	См. вар.8	Гаусса для вычисления корней системы

3.3. Контрольные вопросы

1. Объясните назначение фиксированной и рабочей областей объектов класса *TStringGrid*.
2. Какие значения свойств объектов *StringGrid* вы меняли статически, какие динамически?
3. В чём сущность прямого хода метода Гаусса? Как он применяется для вычисления определителя?
4. Сформулируйте достаточные условия сходимости метода Гаусса-Зейделя.
5. Какие Вы знаете условия выхода из итерационного процесса по методу Гаусса-Зейделя?
6. В каких случаях лучше применять прямые или итерационные методы?
7. Что такое алгоритм поиска ненулевого ведущего элемента и почему при программировании методов возникает необходимость в его применении?

Практическая работа №4

Программирование итерационных алгоритмов решения нелинейных уравнений

Цель работы

1. Освоить работу с объектами класса TChart.
2. Изучить программную организацию графики с помощью графического инструментария.
3. Освоить организацию меню в Windows-приложении.
4. Изучение и программирование методов уточнения корней нелинейных уравнений (бисекции, хорд, касательных и простой итерации).
5. Анализ сходимости методов.

4.1. Краткие сведения из теории

Нелинейные уравнения бывают алгебраическими и трансцендентными. Общая форма задания таких уравнений: $f(x)=0$, где $f(x)$ – некоторая непрерывная функция.

Приближенное нахождение корней нелинейных уравнений состоит из двух этапов:

1. отделение корней;
2. уточнение приближённых корней до некоторой заданной степени точности.

Отделение корня – это отыскание приближённого значения корня или содержащего его отрезка. В данной работе предлагаем воспользоваться графическим способом отыскания начального приближения.

Затем переходим к итерационным методам уточнения этого начального приближения корня. Рассмотрим итерационные методы – бисекции, хорд, касательных и простой итерации.

4.1.1. Метод бисекции

В методе бисекции (деления отрезка пополам) в качестве начального приближения корня x_0 принимают середину отрезка, содержащего корень. Затем исследуют функцию $f(x)$ на знак на концах двух полученных отрезков. Отрезок, на концах которого $f(x)$ принимает значения разных знаков, содержит искомый корень. Его принимают в качестве нового отрезка исследования, второй отрезок отбрасывают. В качестве приближённого значения корня первой итерации x_1 принимают середину нового отрезка.

Снова исследуют $f(x)$ на знак на концах нового отрезка и т.д. После каждой итерации отрезок, содержащий корень, уменьшается вдвое. Это медленная сходимость, но важно, что метод сходится всегда. Условие выхода из итерационного процесса, например, может быть следующим:

$$|f(x_n)| \le \varepsilon,$$

где x_n – приближение корня на n-ной итерации,

ε – заданная точность.

Блок-схема метода бисекции приведена на рис. 4.1. В данном алгоритме сужение отрезка производится путём замены границ a или b на текущее значение корня c. Значение $f(a)$ вычисляется лишь один раз, поскольку нам нужен только знак функции $f(x)$ на левой границе, а он в процессе итераций не меняется.

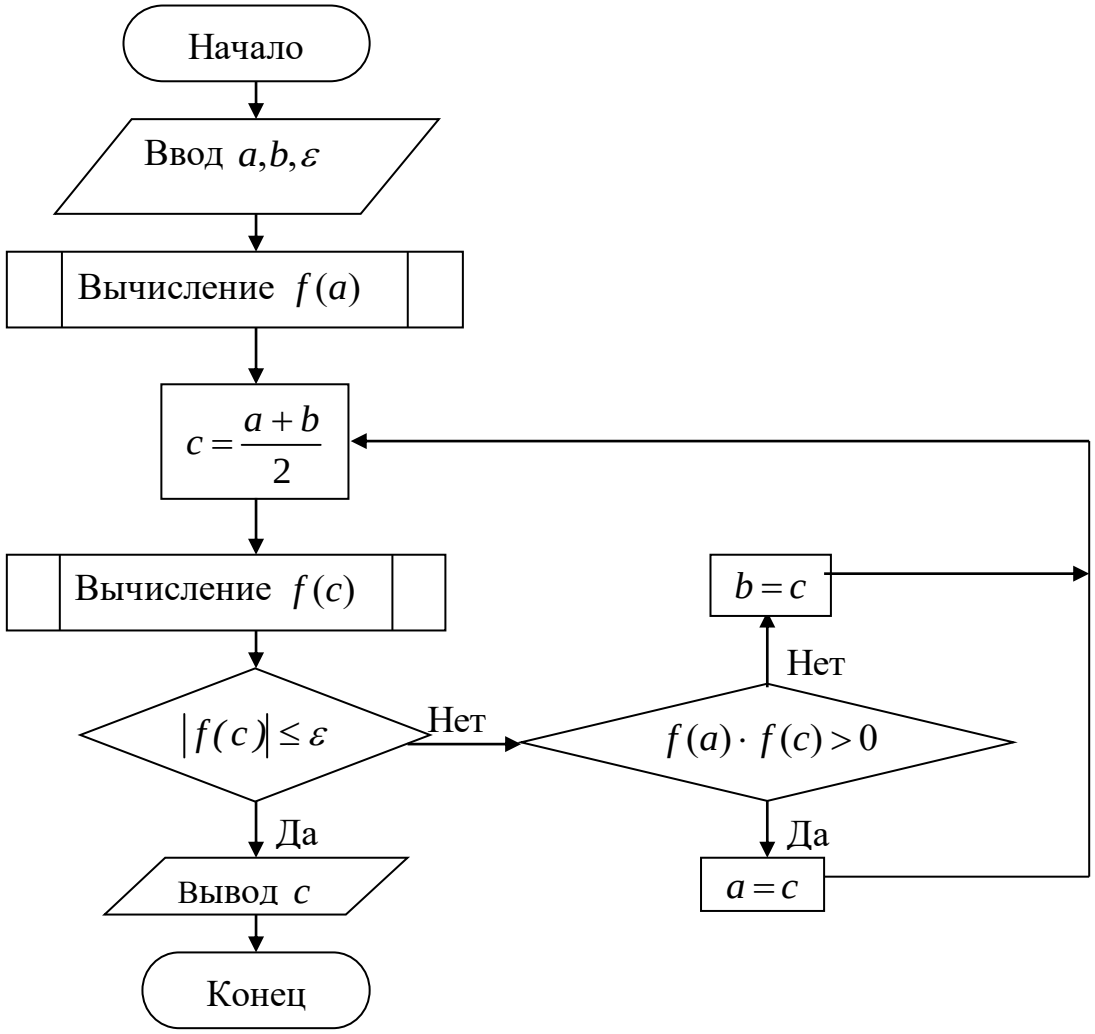


Рис. 4.1. Блок-схема метода бисекции

4.1.2. Метод хорд

В данном методе процесс итераций состоит в том, что в качестве приближений к корню уравнения $f(x)=0$ принимаются значения $c_0, c_1, \ldots$ точек пересечения хорды с осью абсцисс. Уравнение хорды AB, где $A(a, f(a))$, $B(b, f(b))$:

$$\frac{y-f(a)}{f(b)-f(a)}=\frac{x-a}{b-a}\ . \qquad (4.1)$$

Для точки пересечения хорды с осью абсцисс ($x=c_0, y=0$):

$$c_0=a-\frac{b-a}{f(b)-f(a)}\cdot f(a). \qquad (4.2)$$

Далее сравниваем знаки величин $f(a), f(c_0)$. Отрезок, на котором функция меняет знак (например, $f(a)\cdot f(c_0)<0$) оставляем, вторую часть отбрасываем. Следующая состоит в определении нового приближения c_1, как точки пересечения новой хорды с осью абсцисс и т.д. Итерационный процесс продолжаем до тех пор, пока значение $f(c_n)$ не станет по модулю меньше заданного ε: $|f(c_n)|\le\varepsilon$.

Блок-схема метода хорд аналогична приведённой для метода бисекции, но вместо вычисления приближения по формуле $c=\frac{a+b}{2}$ необходимо использовать формулу (4.2). Так же в блок-схему необходимо ввести операторы вычислений значений $f(x)$ на границах новых отрезков.

4.1.3. Метод касательных

При уточнении корня по методу касательных (Ньютона) в точке начального приближения x_0 проводится касательная к функции $f(x)$. Точка x_1 пересечения касательной с осью абсцисс принимается за новое уточнённое значение корня. Затем касательная к $f(x)$ проводится в x_1 и т.д. Уточнение корня выполняется по формуле:

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}\ . \qquad (4.3)$$

Если процесс сходится, то $\lim\limits_{n\to\infty}x_n=a$, где a – значение корня.

Заметим, что данный метод очень быстро сходится, но, чтобы в результате очередной итерации не выйти за пределы отделения корня, необходимо выбирать начальное приближение довольно близким к решению, иначе очень вероятна расходимость.

4.1.4. Метод простой итерации

Уточнение корня по этому методу сводится к замене уравнения $f(x)=0$ ему равносильным: $x=\varphi(x)$. Найденное начальное приближение x_0 подставляют в правую часть уравнения и получают новое значение корня по формуле:

$$x_1=\varphi(x_0) \ \ldots \ x_{n+1}=\varphi(x_n).$$

Достаточное условие сходимости метода простой итерации: $|\varphi'(x)|<1$.

Блок-схема метода простой итерации представлена на рис. 4.2. На схеме c – начальное приближение корня и результат последующей итерации, x – значение корня после каждой итерации.

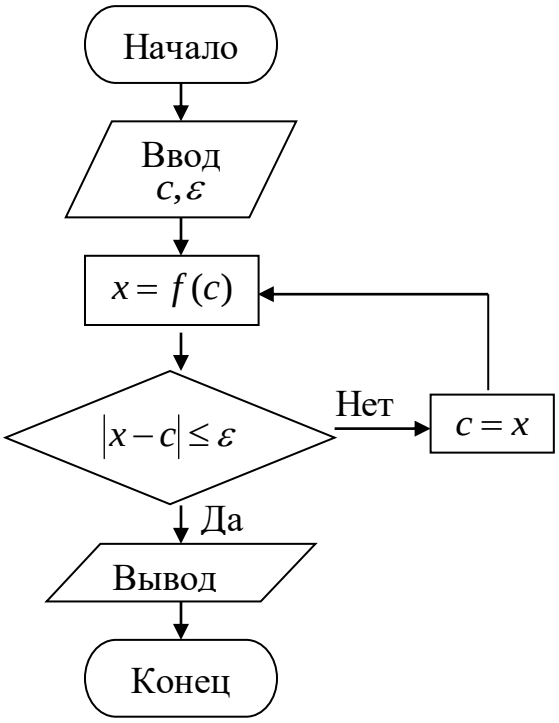


Рис. 4.2. Блок-схема метода простой итерации

4.2. Выполнение работы

Порядок выполнения работы

1. Написать программу для графического отделения действительных корней нелинейного уравнения.
2. Написать программу решения нелинейного уравнения в соответствии с номером варианта (см. таблице 4.1). Общее управление приложением организовать с помощью меню.
3. Проверить, если необходимо, достаточное условие сходимости.
4. Отладить программу на тестовом примере.

5. Произвести счёт и оформить отчёт.

Исходные данные для выполнения работы

Таблица 4.1

Вариант	Уравнение	Метод
1	$x^4 + 7x^3 - x + 12 = 0$	Бисекции, $\varepsilon = 10^{-1}$
2	См. вар. 1	Хорд, $\varepsilon = 10^{-2}$
3	См. вар. 1	Ньютона, $\varepsilon = 10^{-3}$
4	$e^x - \cos 2x = 0$	Ньютона, $\varepsilon = 10^{-3}$
5	См. вар. 4	Простой итерации
6	$\arcsin 4x - 0.3x = -1$	Простой итерации
7	См. вар. 6	Бисекции, $\varepsilon = 10^{-2}$
8	$x^3 - 0.2x^2 + 0.5x = -1.5$	Ньютона, $\varepsilon = 10^{-2}$
9	$5x - 8\ln x = 8$	Простой итерации
10	См. вар. 9	Ньютона, $\varepsilon = 10^{-3}$

4.3. Контрольные вопросы

1. Объясните назначение класса *TCanvas*. С помощью каких инструментов можно запрограммировать вывод изображения на канве?
2. Что такое мастер изображений и как с его помощью работать с объектами класса *TChart*?
3. Как организовать меню в Windows-приложении?
4. Из каких этапов состоит алгоритм нахождения корня нелинейного уравнения?
5. В чём состоит суть методов бисекции, Ньютона, простой итерации.
6. Дайте графическую интерпретацию каждого метода.
7. Сформулируйте достаточные условия сходимости методов Ньютона и простой итерации.
8. В чём состоят преимущества и недостатки применения каждого из методов?

Практическая работа №5

Организация многооконного приложения для решения о.д.у. и систем о.д.у. одношаговыми разностными методами

Цель работы:

1. Изучение и программирование алгоритмов одношаговых методов решения обыкновенных дифференциальных уравнений в постановке задачи Коши.
2. Освоение технологии программирования многооконного приложения .
3. Программирование работы с файлами данных . Объектный подход при работе с файлами.

5.1. Краткие сведения из теории

Обыкновенным дифференциальным уравнением (далее – ОДУ) называется такое уравнение, которое содержит одну или несколько производных от искомой функции $y = y(x)$:

$$f(x, y, y', y'', ..., y^{(n)}) = 0 \qquad (5.1)$$

Здесь x – независимая переменная, n – наивысший в уравнении порядок производной от y, называется порядком уравнения, само ОДУ при этом называется уравнением n-го порядка.

Если вместе с уравнением задаются дополнительные условия в какой-либо точке, называемые начальными, то задача решения ОДУ ставится в виде так называемой задачи Коши. Например, задача Коши для ОДУ первого порядка:

$$\frac{dy}{dx} = f(x, y) \qquad (5.2)$$

$$y(x_0) = y_0 \qquad (5.3)$$

Требуется среди всех решений уравнения (5.2) найти такое решение $y = y(x)$, которое удовлетворяет начальному условию (5.3), т.е. требуется найти функцию $y(x)$, график, которой проходит через заданную точку $M_0(x_0, y_0)$.

Рассмотрим одношаговые методы решения ОДУ. В одношаговых методах для вычисления значения y_{i+1} на следующем шаге используется лишь одно ранее найденное значение на предыдущем шаге y_i. Наиболее распространенными среди одношаговых методов являются методы Эйлера и Рунге-Кутта.

5.1.1. Метод Эйлера

Метод основан на разложении функции $y = y(x)$ в ряд Тейлора в окрестности x_0:

$$y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{1}{2}h^2 y''(x_0) + \ldots.$$

Если h мало, то члены, содержащие h во второй и более высоких степенях, являются малыми и ими пренебрегают. Тогда

$$y(x_0 + h) = y(x_0) + hy'(x_0).$$

Значение $y'(x_0)$ находим из дифференциального уравнения, подставив в него начальные условия. Таким образом можно получить приближённое значение зависимой переменной при малом смещении h от номинальной точки. Этот процесс можно продолжать, используя соотношение

$$y_{i+1} = y_i + hf(x_i y_i), \; i = 1,2,\ldots, \qquad (5.4)$$

делая сколь угодно много шагов.

В результате получен простейший алгоритм решения задачи Коши, который называется методом Эйлера, или методом ломаных. Последнее связано с геометрической интерпретацией процесса: искомая функция $y(x)$ заменяется ломаной линией, представляющей собой отрезки касательных к этой функции в узлах $x_0, x_1, \ldots$.

При достаточно малой величине шага h метод Эйлера даёт решение с большой точностью, т.к. погрешность решения близка к h^2 на каждом шаге интегрирования. На рис. 5.1 приведена блок-схема решения задачи Коши методом Эйлера.

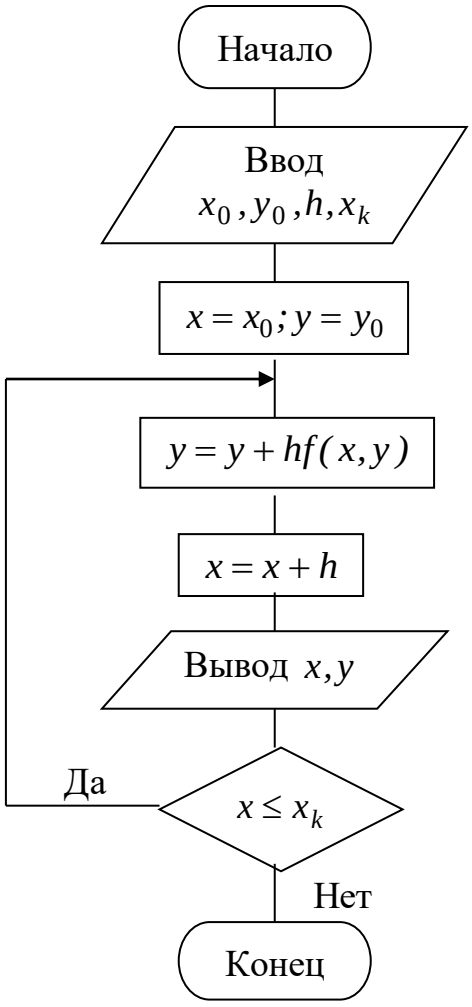


Рис. 5.1 Блок-схема метода Эйлера для задачи Коши

5.1.2. Метод Эйлера с пересчётом

Значение правой части $f(x,y)$ уравнения (5.2) возьмём равным среднему арифметическому между $f(x_i,y_i)$ и $f(x_{i+1},y_{i+1})$. Используя схему метода Эйлера (5.4), получим:

$$y_{i+1} = y_i + \frac{h}{2}[f(x_i,y_i) + f(x_{i+1},y_{i+1})],\ i = 1,2,... \qquad (5.5)$$

Это неявная схема, т.к. y_{i+1} входит и в правую и левую части. Здесь применяют один из итерационных методов. Но если имеется хорошее начальное приближение y_i, то можно построить решение с использованием двух итераций следующим образом: считая y_i начальным приближением, вычисляем первое приближение $\tilde{y}_{i+1}$ по формуле метода Эйлера (5.4):

$$\tilde{y}_{i+1} = y_i + hf(x_i y_i).$$

Новое значение $\tilde{y}_{i+1}$ подставляем вместо y_{i+1} в (5.5) и находим окончательное значение y_{i+1}:

$$y_{i+1} = y_i + \frac{h}{2}[f(x_i,y_i) + f(x_{i+1},\tilde{y}_{i+1})]. \qquad (5.6)$$

Это модификация метода Эйлера, называемая методом Эйлера с пересчётом.

Метод Эйлера с пересчётом можно получить и иначе, используя разложение функции в ряд Тейлора:

$$y_{i+1} = y_i + hy_i' + \frac{h}{2!}y_i'' + O(h^3).$$

Аппроксимируем вторую производную с помощью отношения конечных разностей:

$$y_i'' = \frac{y_{i+1}' - y_i'}{h} + O(h) \qquad \Rightarrow$$

$$\Rightarrow\ y_{i+1} = y_i + hy_i' + \frac{h}{2}(y_{i+1}' - y_i') + O(h^3) = y_i + \frac{h}{2}(y_{i+1}' + y_i') + O(h^3).$$

Заменяя $y_i' = f(x_i,y_i)$ и $y_{i+1}' = f(x_{i+1},\tilde{y}_{i+1})$, где $\tilde{y}_{i+1}$ найдено по методу Эйлера, приходим к формуле метода Эйлера с пересчётом. Здесь мы получили оценку погрешности метода: на каждом шаге локальная погрешность равна $O(h^3)$, суммарная – имеет порядок h^2.

Для метода Эйлера с пересчётом рационально вводить автоматический выбор шага в каждом узле: если величина $|y_{i+1} - \tilde{y}_{i+1}| < \varepsilon$, то шаг увеличиваем, если $|y_{i+1} - \tilde{y}_{i+1}| > \varepsilon$, то шаг уменьшаем.

На рис. 5.2 приведена блок-схема метода Эйлера с пересчётом.

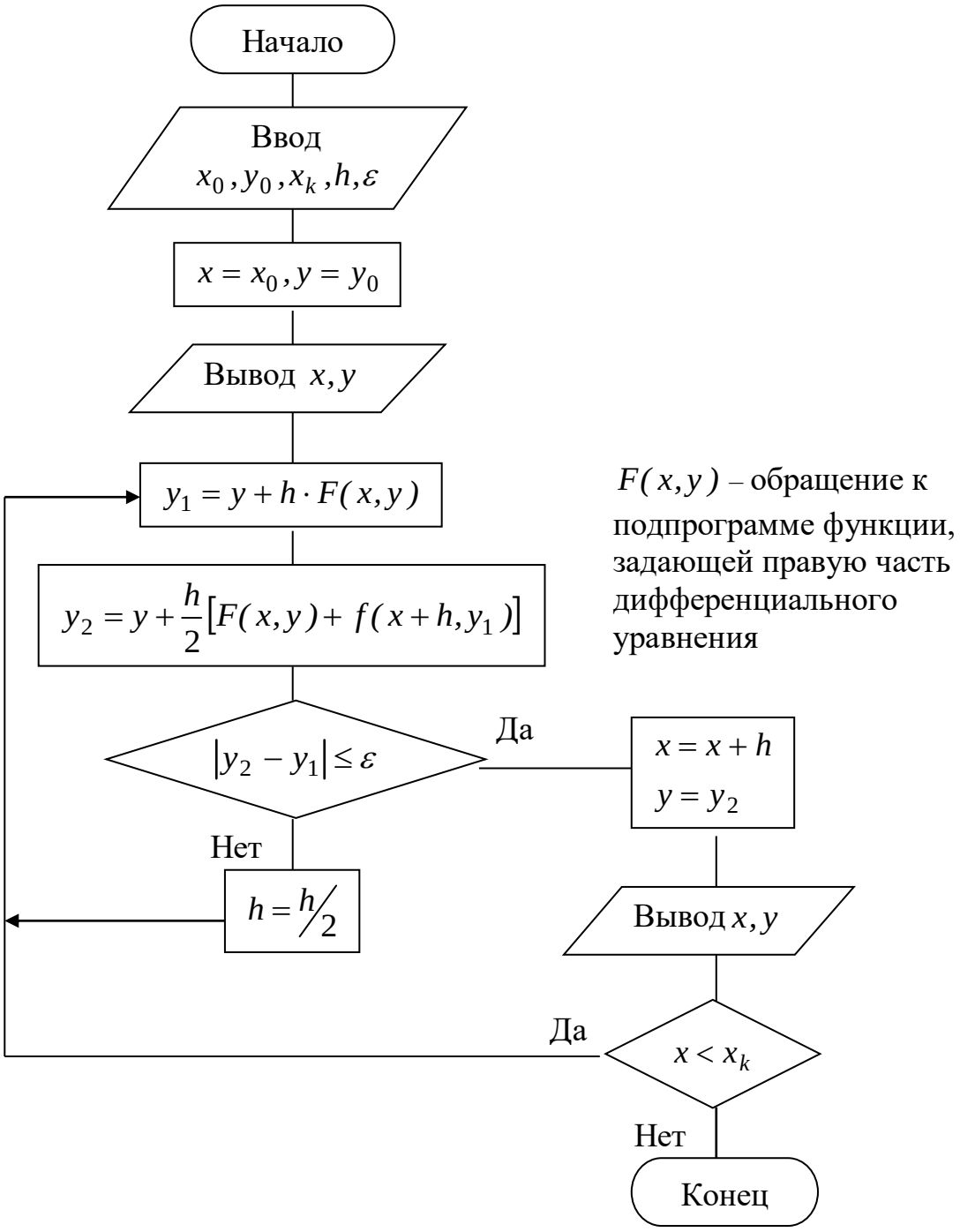


Рис. 5.2 Блок-схема метода Эйлера с пересчётом

5.1.3. Метод Рунге-Кутта

Это очень распространённый явный одношаговый метод. На его основе могут быть построены разностные схемы разного порядка точности. Приведём схему Рунге-Кутта 4-го порядка. Запишем алгоритм в виде:

$y_{i+1} = y_i + \tilde{f}_i$, ($\tilde{f}_i$ - усреднённая первая производная)

$$\tilde{f}_i = \frac{h}{6}(k_{1i} + 2k_{1i} + 2k_{3i} + k_{4i}),$$

$$k_{1i} = f(x_i, y_i), \qquad k_{1i} = f(x_i + \frac{h}{2}, y_i + \frac{k_{1i} h}{2}),$$

$$k_{3i} = f(x_i + \frac{h}{2}, y_i + \frac{k_{2i}h}{2}), \qquad k_{4i} = f(x_i + h, y_i + k_{3i}h).$$

Погрешность метода Рунге-Кутта оценивается величиной $\varepsilon \approx h^5$. Уточнение достигается за счёт специального подбора координат четырёх точек, в которых вычисляется первая производная $f(x,y)$. Вместо первой производной $hf(x_i, y_i)$, используемой в формуле Эйлера, вычисляется усреднённая первая производная $\widetilde{f}_i$. Поэтому метод Рунге-Кутта требует на каждом шаге четырёхкратного вычисления правой части уравнения $f(x,y)$. Значит, удобно выделить в подпрограмму вычисление правых частей по формуле $F = f(x,y)$, придавая аргументам последовательно нужные значения.

Блок-схема метода Рунге-Кутта приведена на рис.5.3.

Метод Эйлера и его модифицированный вариант (с пересчётом) могут рассматриваться как методы Рунге-Кутта первого и второго порядков. Метод Рунге-Кутта требует большого объёма вычислений, но это окупается повышенной точностью, что даёт возможность проводить счёт с большим шагом.

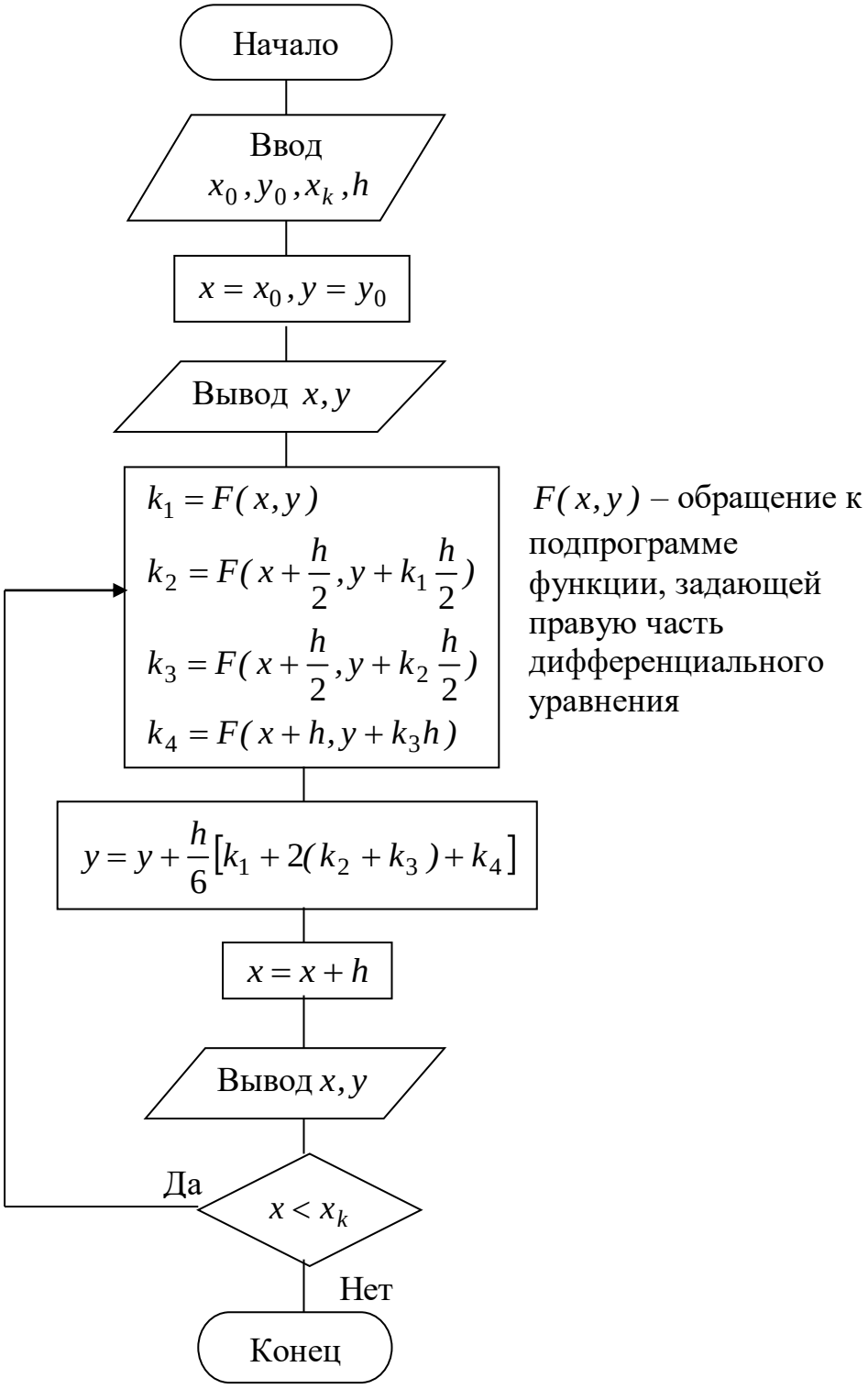


Рис. 5.3. Блок-схема метода Рунге-Кутта 4-го порядка

5.1.4. Решение дифференциальных уравнений высшего порядка

Методы, применяемые для численного интегрирования ОДУ 1-го порядка могут быть использованы для интегрирования систем ОДУ высшего порядка. Последние при этом должны быть приведены к системам 1-го порядка. Например, если мы имеем уравнение 2-го порядка: $\frac{d^2y}{dx^2} = F(x, y, \frac{dy}{dx})$ с одной

искомой функцией, легко перейти к равносильной системе из двух уравнений 1-го порядка с двумя искомыми функциями:

$$\begin{cases} \dfrac{dy}{dx} = z \\ \dfrac{dz}{dx} = F(x, y, z) \end{cases}$$

Подобным образом поступают с уравнениями любого порядка, приводя их системе дифференциальных уравнений 1-го порядка.

Пусть дана система двух дифференциальных уравнений первого порядка в постановке задачи Коши:

$$\begin{cases} \dfrac{dy}{dx} = \varphi(x, y, z) \\ \dfrac{dz}{dx} = \psi(x, y, z) \end{cases}$$

$$y(x_0) = y_0$$

$$z(x_0) = z_0.$$

Запишем формулы Рунге-Кутта для системы двух уравнений:

$$y_{i+1} = y_i + \frac{h}{6}(k_0 + 2k_1 + 2k_2 + k_3);$$

$$z_{i+1} = z_i + \frac{h}{6}(l_0 + 2l_1 + 2l_2 + l_3);$$

$$k_0 = \varphi(x_i, y_i, z_i); \qquad l_0 = \psi(x_i, y_i, z_i);$$

$$k_1 = \varphi\left(x_i + \frac{h}{2}; y_i + \frac{k_0 h}{2}; z_i + \frac{l_0 h}{2}\right); \qquad l_1 = \psi\left(x_i + \frac{h}{2}; y_i + \frac{k_0 h}{2}; z_i + \frac{l_0 h}{2}\right);$$

$$k_2 = \varphi\left(x_i + \frac{h}{2}; y_i + \frac{k_1 h}{2}; z_i + \frac{l_1 h}{2}\right); \qquad l_2 = \psi\left(x_i + \frac{h}{2}; y_i + \frac{k_1 h}{2}; z_i + \frac{l_1 h}{2}\right);$$

$$k_3 = \varphi(x_i + h; y_i + k_2 h; z_i + l_2 h); \qquad l_3 = \psi(x_i + h; y_i + k_2 h; z_i + l_2 h).$$

В результате мы получим решение в виде сеток для функции $y = y(x)$ и её производной $z = \dfrac{dy}{dx}$.

5.2. Выполнение работы

Порядок выполнения работы

1. Организовать многооконное приложение в C# следующей структуры: главная форма, содержащая интерфейс для общего управления проектом и две дочерних формы для а) управления решением о.д.у. первого порядка; б) управления решением о.д.у. высших порядков.
2. Написать процедуру решения ОДУ первого порядка в соответствии с номером варианта из таблиц 5.1.
3. Отладить программу на выбранном самостоятельно тестовом примере и полученный результат сравнить с ручным просчётом по тесту. Убедиться в соответствии ручного и машинного результатов в пределах допустимой точности.
4. Разрешить уравнение своего варианта (ОДУ высшего порядка) относительно старшей производной. В случае необходимости перейти к системе ОДУ

первого порядка. Написать и отладить процедуру для решения ОДУ высшего порядка. Произвести расчет в соответствии с вариантом задания.

5. Составить подпрограмму для представления решения ОДУ в виде графика. Получить графическое решение для контрольного примера на экране дисплея и показать решение преподавателю.
6. Организовать вывод значений сеточного решения в файл данных.
7. Оформить отчёт и защитить работу преподавателю.

Замечание: Каждая составленная студентом процедура должна быть структурирована, т.е. в виде отдельных блоков (подпрограмм) необходимо оформить:

1) метод решения ОДУ;
2) функцию, задающую правую часть ОДУ, разрешённого относительно производной;
3) вычерчивание графика функции по табличному её представлению;
4) запись решений в файл данных.

Исходные данные для выполнения работы

Таблица 5.1

Вариант	Метод	Нач. условия	Шаг (h) или кол-во узлов (n); конечное значение аргумента (x_k)	Уравнение	Точность решения
1	2	3	4	5	6
1	Эйлера	$x=1$ $y=1$	$h=0.1$ $x_k=2$	$xy'+2x^2-e^{-x}=0$	___
2	Эйлера с пересчётом	$x=0$ $y=1$	$h=0.1$ $x_k=1$	$xy'+2x^2-e^{-x}=0$	$\varepsilon=10^{-2}$
3	Рунге-Кутта	$x=0$ $y=1$	$h=0.1$ $x_k=1$	$xy'+2x^2-e^{-x}=0$	___
4	Эйлера	$x=0$ $y=1$	$n=10$ $x_k=1$	$\cos x-y'=xy$	___

Таблица 5.1 (продолжение)

1	2	3	4	5	6
5	Эйлера с пересчётом	$x=1$ $y=1.5$	$h=0.2$ $x_k=3$	$cos(y+0.6)-y'=2.5x$	$\varepsilon=10^{-1}$
6	Рунге-Кутта	$x=0$ $y=1$	$h=0.1$ $x_k=1$	$\cos x - y' = xy$	___
7	Эйлера с пересчётом	$x=0$ $y=1$	$n=10$ $x_k=1$	$y'y + y^2e^{-x} - x = 0$	$\varepsilon=10^{-2}$
8	Рунге-Кутта	$x=0$ $y=1$	$h=0.1$ $x_k=1$	$y'y + y^2e^{-x} - x = 0$	___
9	Рунге-Кутта	$x=0$ $y=0$	$h=0.1$ $x_k=2$	$(y'+0.5y^2)(1.25+x)=\cos y$	___
10	Эйлера	$x=0$ $y=1$	$h=0.1$ $x_k=1$	$y'y + y^2e^{-x} - x = 0$	___
11	Рунге-Кутта	$y(0.4)=2$ $y'(0.4)=2.5$	$h=0.1$ $x_k=2$	$y'' - 3y' + \frac{y}{x} = 1$	___
12	Рунге-Кутта	$y(1)=0.4$ $y'(1)=0.2$	$h=0.1$ $x_k=3$	$x^2y'' + xy' + (x^2-1)y = 0$	___

5.3. Контрольные вопросы

1. Как организовать связь между модулями в многооконном приложении в Delphi?
2. Объясните разницу между методами *Show* и *ShowModal*.
3. Как получена формула метода Эйлера решения ОДУ 1-го порядка?
4. В чём состоит модификация метода Эйлера с пересчётом. Оцените точность полученного метода.
5. В чём суть метода Рунге-Кутта? Почему у метода более высокая точность по сравнению с методами Эйлера?
6. Каким образом использовать формулы описанных методов для решения систем дифференциальных уравнений и дифференциальных уравнений высших порядков?

Список использованной литературы

Основные источники:

1. Численные методы и программирование: Учебное пособие / В.Д. Колдаев; Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: НИЦ Инфра-М, 2013. - 336 с.

Дополнительные источники:

2. Зенков, А. В. Численные методы : учебное пособие для СПО / А. В. Зенков. — М. :
3. Издательство Юрайт, 2017.
4. Лапчик М.П. Численные методы: / М.П.Лапчик, М.И.Рагулина, Е.К.Хеннер; под ред
5. М.П.Лапчика. – М.:Издательский центр «Академия», 2015, 224 с

Интернет-ресурсы :

1. http://www.uchites.ru/chislennye_metody/posobie
2. http://www.intuit.ru/department/calculate/vnmdiffeq/
3. http://www.intuit.ru/department/calculate/calcmathbase/