Министерство образования Республики Беларусь
Учреждение образования
Белорусский государственный университет информатики и радиоэлектроники
Кафедра информационных технологий автоматизированных систем

В. С. Муха

Вычислительные методы и компьютерная алгебра

Лабораторный практикум для студентов специальности I-53 01 02
"Автоматизированные системы обработки информации"

Минск 2012

ОГЛАВЛЕНИЕ

ЛАБОРАТОРНАЯ РАБОТА № 1. Работа в системе Matlab

1.1. Цель работы

1.1.1. Ознакомление с системой Matlab, приобретение навыков работы.

1.1.2. Ознакомление с языком программирования Matlab.

1.1.3. Приобретение навыков программирования на языке Matlab.

1.2. Порядок выполнения работы

1.2.1. Ознакомиться с системой Matlab, ее запуском и работой в ней по методическому пособию "Введение в Matlab" [1].

1.2.2. Разработать m-файл-сценарий для вывода в графическое окно графика функции одной переменной с помощью программы **plot**. Функцию взять из табл. 1.1 в соответствии с номером своей бригады и кодом подгруппы (а или б). Функцию оформить в виде m-файла-функции.

1.2.3. Разработать m-файл-сценарий для вывода в одно графическое окно контурных графиков двух функций двух переменных на уровне $z_1 = 0$, $z_2 = 0$ с помощью программ **meshgrid** и **contour**. Функции взять из табл. 1.2 в соответствии с номером своей бригады и кодом подгруппы. Функции оформить в виде m-файлов-функций.

1.2.4. Разработать m-файл-сценарий для вывода в графическое окно графика функции двух переменных с помощью программ **meshgrid**, **mesh** и **meshc** для одной из функций табл. 1.2 в соответствии с номером своей бригады и кодом подгруппы. Функцию оформить в виде m-файла-функции.

Таблица 1.1

Функции одной переменной для индивидуальных заданий

№ варианта	Функция	№ варианта	Функция
1а	$y = x^3 e^{-x} + 6x - 5$	1б	$y = x^4 - 5x + 1$
2а	$y = x^6 - 2x - 7$	2б	$y = x^3 \cos x - 2x^2 + x + 1$
3а	$y = x^3 \sin x - 0{,}985x - 0{,}991$	3б	$y = x^5 + 1{,}025x - 3{,}116$
4а	$y = 2x^2 \ln x - 1$	4б	$y = x^2 \lg x - 1$
5а	$y = 2x \ln x - 1$	5б	$y = 2 \lg x - (x-2)^2$
6а	$y = e^x - 2(x-1)^2$	6б	$y = (x-1)^2 - \sin 2x$
7а	$y = e^x + x^2 - 2$	7б	$y = x^3 - 12x - 8$
8а	$y = 2^x + x^2 - 1{,}15$	8б	$y = e^{-x} + x^2 - 2$
9а	$y = \sqrt{x} - \cos 0{,}387x$	9б	$y = tg\frac{\pi}{4}x - x - 3$
10а	$y = 2x \lg x - x + 2$	10б	$y = 3^{-x} - x^2 + 1$
11а	$y = x^3 - 3x + 3$	11б	$y = \ln x - \sqrt{4 - 2x}$
12а	$y = \ln x \sin x$	12б	$y = x^3 - \cos \pi x$
13а	$y = e^x \cos 2x + x - 3$	13б	$y = x \sin x + x - 7$
14а	$y = e^{-x} \cos 2x + x - 3$	14б	$y = e^{-x \sin x} - 3$
15а	$y = e^{-x} \sin 2x + x^2 - 3$	15б	$y = x \sin x - 5$

Таблица 1.2

Функции двух переменных для индивидуальных заданий

№ варианта	Функции	№ варианта	Функции
1	2	3	4
1а	$z_1 = \sin(x+1) - y - 1{,}2$ $z_2 = 2x + \cos y - 2$	1б	$z_1 = \sin y + 2x - 2$ $z_2 = y + \cos(x-1) - 0{,}7$
2а	$z_1 = \cos(x-1) + y - 0{,}5$ $z_2 = x - \cos y - 3$	2б	$z_1 = \cos y + x - 1{,}5$ $z_2 = 2y - \sin(x-0{,}5) - 1$
3а	$z_1 = \sin x + 2y - 2$ $z_2 = \cos(y-1) + x - 0{,}7$	3б	$z_1 = \sin(y+0{,}5) - x - 1$ $z_2 = y + \cos(x-2)$
4а	$z_1 = \cos x + y - 1{,}5$ $z_2 = 2x - \sin(y-0{,}5) - 1$	4б	$z_1 = \cos(y+0{,}5) + x - 0{,}8$ $z_2 = -2y + \sin x - 1{,}6$
5а	$z_1 = \sin(x+0{,}5) - y - 1$ $z_2 = x + \cos(y-2)$	5б	$z_1 = \sin(y-1) + x - 1{,}3$ $z_2 = y - \sin(x+1) - 0{,}8$
6а	$z_1 = \cos(x+0{,}5) + y - 0{,}8$ $z_2 = -2x + \sin y - 1{,}6$	6б	$z_1 = 2x - \cos(y+1)$ $z_2 = y + \sin x + 0{,}4$
7а	$z_1 = \sin(x-1) + y - 1{,}3$ $z_2 = x - \sin(y+1) - 0{,}8$	7б	$z_1 = \cos(y+0{,}5) - x - 2$ $z_2 = \sin x - 2y - 1$
8а	$z_1 = -\cos(x+1) + 2y$ $z_2 = x + \sin y + 0{,}4$	8б	$z_1 = \sin(y+2) - x - 1{,}5$ $z_2 = y + \cos(x-2) - 0{,}5$
9а	$z_1 = \cos(x+0{,}5) - y - 2$ $z_2 = -2x + \sin y - 1$	9б	$z_1 = \sin(x+1) - y - 1$ $z_2 = 2x + \cos y - 2$
10а	$z_1 = \sin(x+2) - y - 1{,}5$	10б	$z_1 = \cos(x-1) + y - 0{,}8$

	$z_2 = x + \cos(y-2) - 0{,}5$		$z_2 = x - \cos y - 2$
11а	$z_1 = \sin(y+1) - x - 1{,}2$ $z_2 = 2y + \cos x - 2$	11б	$z_1 = \sin x + 2y - 1{,}6$ $z_2 = x + \cos(y-1) - 1$
12а	$z_1 = \cos(y-1) + x - 0{,}5$ $z_2 = y - \cos x - 3$	12б	$z_1 = \cos x + y - 1{,}2$ $z_2 = 2x - \sin(y-0{,}5) - 2$
13а	$z_1 = \cos(y-1) + x - 0{,}5$ $z_2 = \ln y - \cos x - 3$	13б	$z_1 = \sin(x+2) - y - 1{,}5$ $z_2 = e^{0{,}1x} + \cos(y-2) - 0{,}5$
14а	$z_1 = \cos 2x + y - 0{,}8$ $z_2 = -2x + \sin y - 1{,}6$	14б	$z_1 = x\cos 2x + y - 0{,}8$ $z_2 = -2x + y\sin y - 1{,}6$
15а	$z_1 = x\sin x + 2y - 1{,}6$ $z_2 = x + \cos(y-1) - 1$	15б	$z_1 = \sin(y+1)y - x - 1$ $z_2 = 2y + \cos x - 2$

ЛАБОРАТОРНАЯ РАБОТА № 2. Решение систем линейных алгебраических уравнений

2.1. Цель работы

2.1.1. Изучение методов решения систем линейных алгебраических уравнений (СЛАУ).

2.1.2. Приобретение навыков программирования методов Гаусса и Гаусса–Зейделя.

2.1.3. Приобретение навыков использования стандартных средств системы Matlab для решения СЛАУ.

2.2. Теоретические положения

2.2.1. Постановка задачи

Системой линейных алгебраических уравнений (СЛАУ) называется следующая система равенств

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2, \\ \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n, \end{cases} \tag{2.1}$$

которая при некоторых значениях переменных $x_2, x_2, \ldots, x_n$ превращается в систему тождеств. Решить данную систему – это значит по известным коэффициентам системы $a_{i,j}$, $i, j = \overline{1,n}$, и правым частям b_i, $i = \overline{1,n}$, найти значения переменных $x_2, x_2, \ldots, x_n$, при которых эти равенства превращаются в тождества.

Часто систему (2.1) записывают в векторно-матричной форме. Для этого вводят в рассмотрение квадратную $(n \times n)$-матрицу коэффициентов системы

$A=(a_{i,j}),\ i,j=\overline{1,n}$, и векторы-столбцы неизвестных $X=(x_j),\ j=\overline{1,n}$, и правой части системы $B=(b_i),\ i=\overline{1,n}$,

$$A=\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots\vdots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix},\quad X=(x_j)=\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix},\quad B=(b_i)=\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

Тогда система уравнений (2.1) записывается в виде

$$AX=B. \tag{2.2}$$

Эта запись совпадает по форме с линейным уравнением $ax=b$, решением которого является $x=\frac{b}{a}=a^{-1}b$. Аналогичную простую формулу можно записать и для решения векторно-матричного уравнения (2.2). Если определитель $\det(A)=|A|\neq 0$, то система имеет единственное решение

$$X=A^{-1}B, \tag{2.3}$$

где A^{-1} – матрица, обратная матрице A. Известно также правило Крамера для решения СЛАУ (2.1), в соответствии, с которым неизвестные определяются по формуле:

$$x_i=\frac{\Delta_i}{\Delta},\quad i=\overline{1,n}, \tag{2.4}$$

где $\Delta=\det(A)$ – определитель матрицы A, а Δ_i – определитель матрицы A, в которую вместо коэффициентов $a_{i,j}$ при x_j подставлены свободные члены b_i.

Однако решение СЛАУ с помощью обратной матрицы (2.3) или с помощью правила Крамера (2.4) является достаточно трудоемким. Известны более рациональные численные методы решения СЛАУ, рассмотренные ниже. Вообще все методы решения СЛАУ можно разделить на конечные и итерационные. Конечные методы позволяют получить решение с определенной точностью за известное заранее конечное число операций. В итерационных методах число операций заранее не определено. Оно зависит от точности, с которой необходимо по-

лучить решение. К конечным методам решения СЛАУ относится метод исключения Гаусса, а к итерационным – метод Гаусса–Зейделя.

2.2.2. Метод Гаусса для решения СЛАУ

Метод Гаусса в решении СЛАУ (2.1) состоит из двух этапов: исключение переменных (прямой ход) и нахождение решения (обратный ход).

Прямой ход состоит из $n-1$ шагов. На первом шаге исключается неизвестная x_1 из всех уравнений, начиная со второго. На втором шаге исключается x_2 из всех уравнений, начиная с третьего. На k-м шаге исключается x_k, из всех уравнений, начиная $k+1$ уравнения. На последнем $(n-1)$-м шаге исключается x_{n-1} из последнего уравнения. В результате выполнения прямого хода мы получаем систему уравнений с так называемой верхней треугольной матрицей коэффициентов.

Обратный ход позволяет последовательно получить неизвестные системы уравнений. Сначала определяют x_n из последнего n-го уравнения. Затем это значение подставляют в $(n-1)$-е уравнение и определяют x_{n-1}, и т. д., до определения x_1 из первого уравнения.

Опишем более подробно шаги прямого хода. На первом шаге i-е уравнение начиная с $i=2$ преобразуется следующим образом. Вводится коэффициент

$$m_i = \frac{a_{i1}}{a_{11}}, \quad i = \overline{2,n},$$

и из i-го уравнения вычитается 1-е уравнение, умноженное на этот коэффициент. Результирующее уравнение записывается на место i-го. В результате из i-го уравнения исключается переменная x_1. После этого шага система уравнений примет следующий вид:

$$
\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + \ldots + a_{1,n}x_n &= b_1, \\
a_{2,2}^{(1)}x_2 + a_{2,3}^{(1)}x_3 + \ldots + a_{2,n}^{(1)}x_n &= b_2^{(1)}, \\
a_{3,2}^{(1)}x_2 + a_{3,3}^{(1)}x_3 + \ldots + a_{3,n}^{(1)}x_n &= b_3^{(1)}, \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
a_{n,2}^{(1)}x_2 + a_{n,3}^{(1)}x_3 + \ldots + a_{n,n}^{(1)}x_n &= b_n^{(1)},
\end{aligned}
$$

где $a_{i,j}^{(1)}, b_i^{(1)}$ – коэффициенты, полученные на первом шаге прямого хода. Они определяются следующими выражениями:

$$a_{i,j}^{(1)} = a_{i,j} - m_i a_{1,j},$$

$$b_i^{(1)} = b_i - m_i b_1.$$

На втором шаге i-е уравнение начиная с $i = 3$ преобразуется следующим образом. Вводится коэффициент

$$m_i^{(1)} = \frac{a_{i,2}^{(1)}}{a_{2,2}^{(1)}}, \quad i = \overline{3,n},$$

и из i-го уравнения вычитается 2-е уравнение, умноженное на этот коэффициент. Результирующее уравнение записывается на место i-го. В результате из i-го уравнения исключается переменная x_2. После второго шага система уравнений примет следующий вид:

$$
\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + \ldots + a_{1,n}x_n &= b_1, \\
a_{2,2}^{(1)}x_2 + a_{2,3}^{(1)}x_3 + \ldots + a_{2,n}^{(1)}x_n &= b_2^{(1)}, \\
a_{3,3}^{(2)}x_3 + \ldots + a_{3,n}^{(2)}x_n &= b_3^{(2)}, \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
a_{n,3}^{(2)}x_3 + \ldots + a_{2,n}^{(2)}x_n &= b_n^{(2)},
\end{aligned}
$$

где $a_{i,j}^{(2)}, b_i^{(2)}$ – коэффициенты, полученные на втором шаге прямого хода. Они определяются выражениями

$$a_{i,j}^{(2)} = a_{i,j}^{(1)} - m_i^{(1)} a_{2,j}^{(1)},$$

$$b_i^{(2)} = b_i^{(1)} - m_i^{(1)} b_2^{(1)}.$$

Вообще, на k-м шаге i-е уравнение начиная с $i = k+1$ преобразуется следующим образом. Вводится коэффициент

$$m_i^{(k-1)} = \frac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}}, \quad i = \overline{k+1, n}, \tag{2.5}$$

и из i-го уравнения вычитается k-е уравнение, умноженное на этот коэффициент. Результирующее уравнение записывается на место i-го. В результате из i-го уравнения исключается переменная x_k. Коэффициенты системы уравнений на k-м шаге пересчитываются по формулам:

$$a_{i,j}^{(k)} = a_{i,j}^{(k-1)} - m_i^{(k-1)} a_{k,j}^{(k-1)}; \tag{2.6}$$

$$b_i^{(k)} = b_i^{(k-1)} - m_i^{(k-1)} b_k^{(k-1)}, \tag{2.7}$$

$$i = \overline{k+1, n}, \quad j = \overline{k, n}, \quad k = \overline{1, n-1}.$$

При $k = n-1$ происходит исключение x_{n-1} из последнего уравнения и окончательная верхняя треугольная система записывается следующим образом:

$$\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + \ldots + a_{1,n}x_n &= b_1, \\
a_{2,2}^{(1)}x_2 + a_{2,3}^{(1)}x_3 + \ldots + a_{2,n}^{(1)}x_n &= b_2^{(1)}, \\
a_{3,3}^{(2)}x_3 + \ldots + a_{3,n}^{(2)}x_n &= b_3^{(2)}, \\
\ldots\ldots\ldots\ldots\ldots & \\
a_{n,n}^{(n-1)}x_n &= b_n^{(n-1)},
\end{aligned}$$

Теперь выполняется *обратный ход*. Видно, что из последнего уравнения можно сразу определить x_n,

$$x_n = \frac{b_n^{(n-1)}}{a_{n,n}^{(n-1)}}.$$

Подставляя это значение в предпоследнее уравнение, находим x_{n-1},

$$x_{n-1} = \frac{b_{n-1}^{(n-2)} - a_{n-1,n}^{(n-2)} x_n}{a_{n-1,n-1}^{(n-2)}}.$$

Для нахождения любой переменной x_j применяется формула

$$x_j = \frac{b_j^{(j-1)} - a_{j,n}^{(j-1)} x_n - \ldots - a_{j,j+1}^{(j-1)} x_{j+1}}{a_{j,j}^{(j-1)}}, \quad j = n-1, n-2, \ldots, 1.$$

Замечание. В процессе решения СЛАУ легко может быть получен определитель системы $\det(A)$. Он равен произведению диагональных элементов матрицы верхней треугольной системы:

$$\det(A) = a_{1,1} a_{2,2}^{(1)} a_{3,3}^{(2)} \ldots a_{n,n}^{(n-1)}.$$

Метод исключения Гаусса требует приблизительно n^2 ячеек памяти и выполнения приблизительно $\frac{2}{3}n^3$ арифметических операций.

Метод Гаусса реализуется по схеме, приведенной на рис. 2.1., в случае, когда блок № 5 “Выбор главного элемента” пропускается.

2.2.3. Метод Гаусса с выбором главного элемента

Основные вычисления в методе исключений Гаусса выполняются по формулам (2.6), (2.7). Эти формулы позволяют проследить накопление погрешностей в процессе вычислений.

Обозначим $\delta_{i,j}$ относительную погрешность, содержащуюся в коэффициенте $a_{i,j}^{(k-1)}$, α, β, γ – относительные погрешности округления соответственно при делении, умножении и вычитании. Тогда для относительной погрешности $\delta_{i,j}^{(k)}$ вычисления коэффициента $a_{i,j}^{(k)}$ можно получить следующее выражение:

$$\delta_{i,j}^{(k)} = (\delta_{i,k} - \delta_{k,k} + \alpha + \delta_{k,j} + \beta) \frac{m_i^{(k-1)} a_{k,j}^{(k-1)}}{a_{i,j}^{(k)}} + \delta_{i,j} \frac{a_{i,j}^{(k-1)}}{a_{i,j}^{(k)}} + \gamma = \frac{\Delta_{i,j}^{(k)}}{a_{i,j}^{(k)}},$$

откуда получаем формулу для оценки абсолютной погрешности $\Delta_{i,j}^{(k)}$:

$$|\Delta_{i,j}^{(k)}| \le (|\delta_{i,k}|+|\delta_{k,k}|+|\alpha|+|\delta_{k,j}|+|\beta|)\cdot|m_i^{(k-1)}|\cdot|a_{k,j}^{(k-1)}|+$$
$$+|\delta_{i,j}|\cdot|a_{i,j}^{(k-1)}|+|\gamma|\cdot|a_{i,j}^{(k)}|.$$

Если предположить, что погрешности $\delta_{i,j}$, α,β,γ не превышают некоторой величины C, то получим следующую формулу для оценки погрешности:

$$|\Delta_{i,j}^{(k)}| \le (5\cdot|m_i^{(k-1)}|\cdot|a_{k,j}^{(k-1)}|+|a_{i,j}^{(k-1)}|+|a_{i,j}^{(k)}|)C.$$

Из последнего выражения видно, что погрешность вычисления коэффициента $a_{i,j}^{(k)}$ в основном определяется первым слагаемым в скобках этого выражения и уменьшается с уменьшением $|m_i^{(k-1)}|$. Чтобы $|m_i^{(k-1)}|$ было по возможности меньшим, необходимо чтобы $|a_{k,k}^{(k-1)}|$ было по возможности большим. Поэтому перед выполнением шага исключения каждой переменной желательно переставить уравнения системы так, чтобы

$$|a_{k,k}^{(k-1)}| \ge |a_{i,k}^{(k-1)}|,$$

потому что тогда

$$|m_i^{(k-1)}| \le 1.$$

Если в методе Гаусса выполняется такая перестановка, то метод называется методом Гаусса с выбором главного элемента. Этот метод имеет меньшую погрешность при определении решения СЛАУ. Метод Гаусса с выбором главного элемента реализуется с помощью схемы, приведенной на рис. 2.1. Схема блока № 5 “Выбор главного элемента” приведена на рис. 2.2.

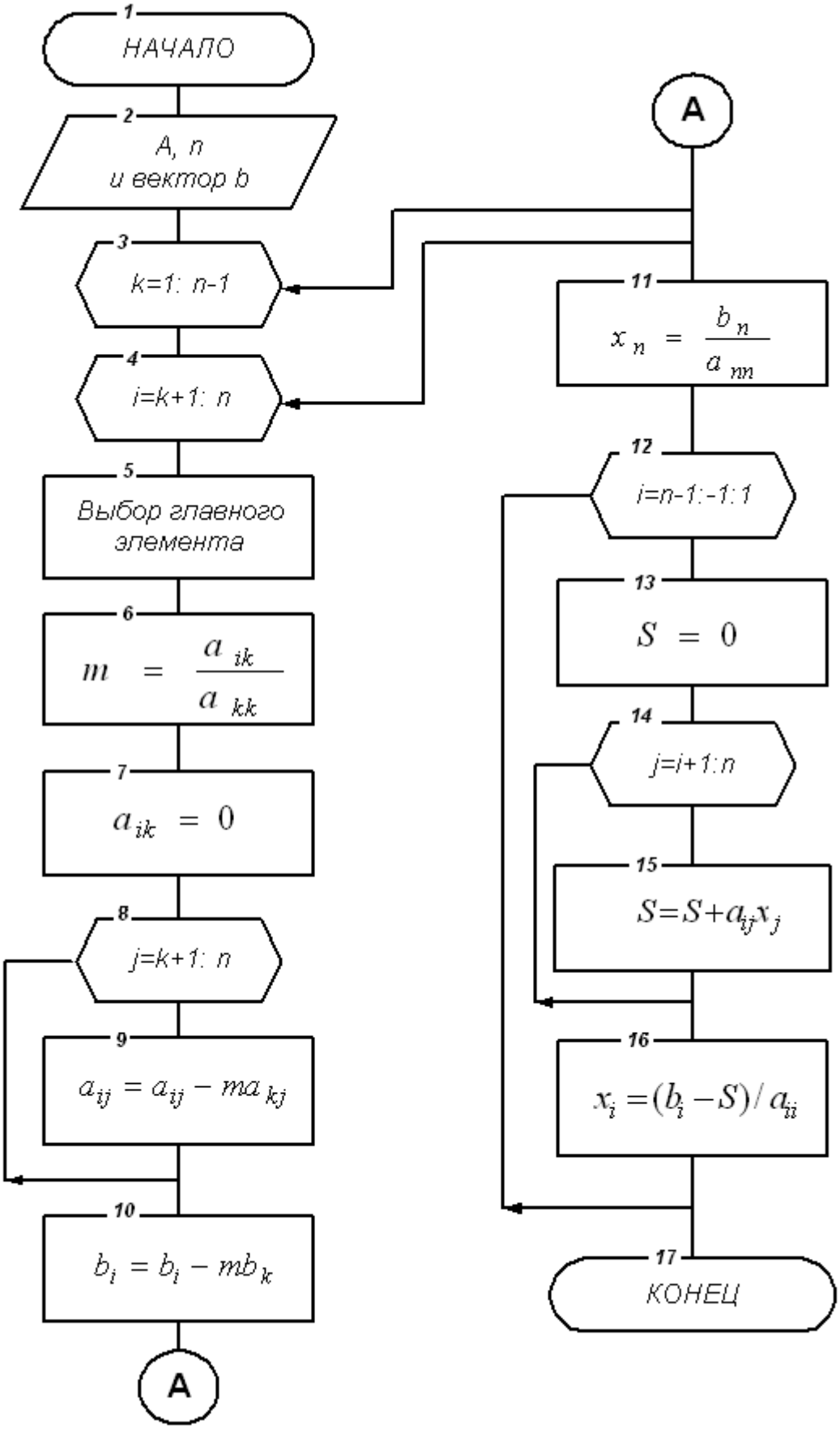
1
НАЧАЛО
2
A, n
и вектор b
3
k=1: n-1
4
i=k+1: n
5
Выбор главного элемента
6
$m = \frac{a_{ik}}{a_{kk}}$
7
$a_{ik} = 0$
8
j=k+1: n
9
$a_{ij} = a_{ij} - ma_{kj}$
10
$b_i = b_i - mb_k$
A
A
11
$x_n = \frac{b_n}{a_{nn}}$
12
i=n-1:-1:1
13
$S = 0$
14
j=i+1:n
15
$S=S+a_{ij}x_j$
16
$x_i = (b_i - S)/a_{ii}$
17
КОНЕЦ

Рис. 2.1.

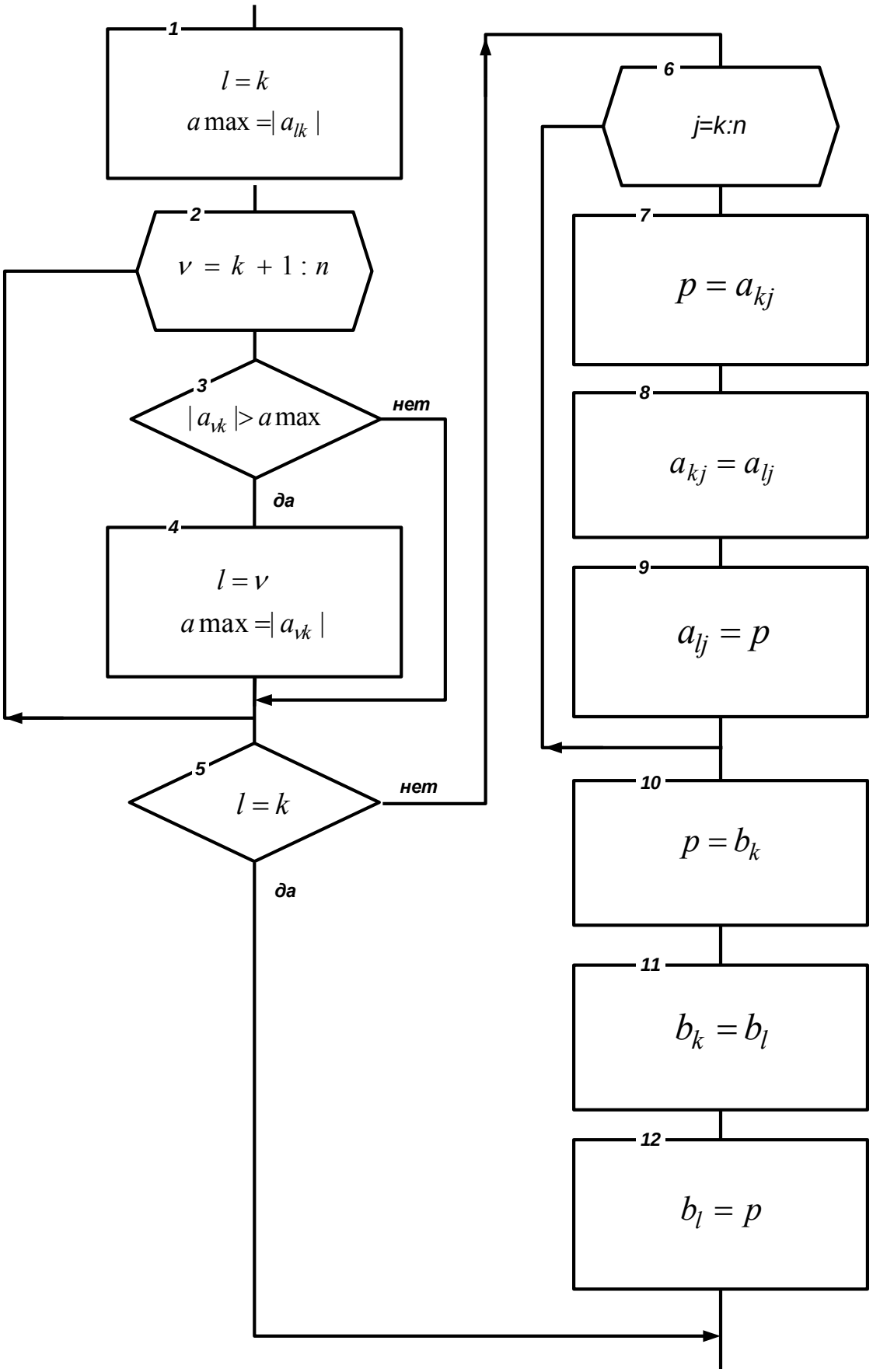
1
$l = k$
$a\max = |a_{lk}|$
2
$v = k + 1 : n$
3
$|a_{vk}| > a\max$
нет
да
4
$l = v$
$a\max = |a_{vk}|$
5
$l = k$
нет
да
6
j=k:n
7
$p = a_{kj}$
8
$a_{kj} = a_{lj}$
9
$a_{lj} = p$
10
$p = b_k$
11
$b_k = b_l$
12
$b_l = p$

Рис. 2.2.

2.2.4. Метод Гаусса–Зейделя

Метод Гаусса–Зейделя – это итерационный метод решения задачи, или метод последовательных приближений. Дадим описание этого метода. Пусть решается система уравнений (2.1). Выразим из 1-го уравнения x_1, из 2-го уравнения x_2 и т.д. В результате получим

$$x_1 = \frac{1}{a_{1,1}}(b_1 - a_{1,2}x_2 - a_{1,3}x_3 - \ldots - a_{1,n}x_n),$$

$$x_2 = \frac{1}{a_{2,2}}(b_2 - a_{2,1}x_1 - a_{2,3}x_3 - \ldots - a_{2,n}x_n),$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$x_n = \frac{1}{a_{n,n}}(b_n - a_{n,1}x_1 - a_{n,2}x_2 - \ldots - a_{n,n-1}x_{n-1}),$$

или вообще для любого i

$$x_i = \frac{1}{a_{i,i}}(b_i - a_{i,1}x_1 - a_{i,2}x_2 - \ldots - a_{i,i-1}x_{i-1} - a_{i,i+1}x_{i+1} - \ldots - a_{i,n}x_n), \quad i = \overline{1,n}. \qquad (2.8)$$

Предположим, что на некоторой k-й итерации мы получили решение $x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}$. Используем известные к моменту расчета x_i значения других переменных в правой части уравнения (2.8) для расчета значения x_i в левой части. В результате мы получим следующую рекуррентную формулу, которая и составляет метод Гаусса–Зейделя:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}}(b_i - a_{i,1}x_1^{(k+1)} - \ldots - a_{i,i-1}x_{i-1}^{(k+1)} - a_{i,i+1}x_{i+1}^{(k)} - \ldots - a_{i,n}x_n^{(k)}), \quad i = \overline{1,n}.$$

Расчеты по последней формуле продолжаются при $k = 1,2,3,\ldots$ до тех пор, пока не будет выполняться условие

$$\max_i |x_i^{(k+1)} - x_i^{(k)}| < \varepsilon,$$

или условие

$$\max_i \left| \frac{x_i^{(k+1)} - x_i^{(k)}}{x_i^{(k+1)}} \right| < \delta,$$

где $\varepsilon > 0$, $\delta > 0$, причем ε – требуемая абсолютная погрешность нахождения решения СЛАУ, а δ – требуемая относительная погрешность.

Итерационные методы могут сходиться к решению или не сходиться. То же самое относится и к методу Гаусса–Зейделя. Метод Гаусса–Зейделя сходится, если выполняются условия

$$|a_{i,i}| \geq |a_{i,1}| + |a_{i,2}| + \ldots + |a_{i,i-1}| + |a_{i,i+1}| + \ldots + |a_{i,n}|$$

для всех $i = \overline{1,n}$, кроме одного, для которого выполняется менее жесткое условие

$$|a_{i,i}| > |a_{i,1}| + |a_{i,2}| + \ldots + |a_{i,i-1}| + |a_{i,i+1}| + \ldots + |a_{i,n}|.$$

Данное условие является достаточным, но не необходимым. Возможны случаи невыполнения данного условия при сходимости метода.

Содержательный смысл приведенных условий сходимости поясним на примере системы двух уравнений с двумя неизвестными

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 = b_2 \end{cases}.$$

Достаточные условия сходимости для этой системы имеют вид

$$|a_{1,1}| \geq |a_{12}| \quad \text{и} \quad |a_{2,2}| > |a_{11}|,$$

или

$$|a_{1,1}| > |a_{12}| \quad \text{и} \quad |a_{2,2}| \geq |a_{11}|.$$

Данные условия означают, что диагональные элементы системы уравнений по абсолютной величине превышают недиагональные. Такой же смысл имеют и условия сходимости для системы из n уравнений. В связи с этим часто для обеспечения сходимости метода Гаусса–Зейделя бывает достаточно поменять местами уравнения системы. Схема алгоритма для решения системы линейных уравнений итерационным методом Гаусса–Зейделя представлена на рис. 2.3.

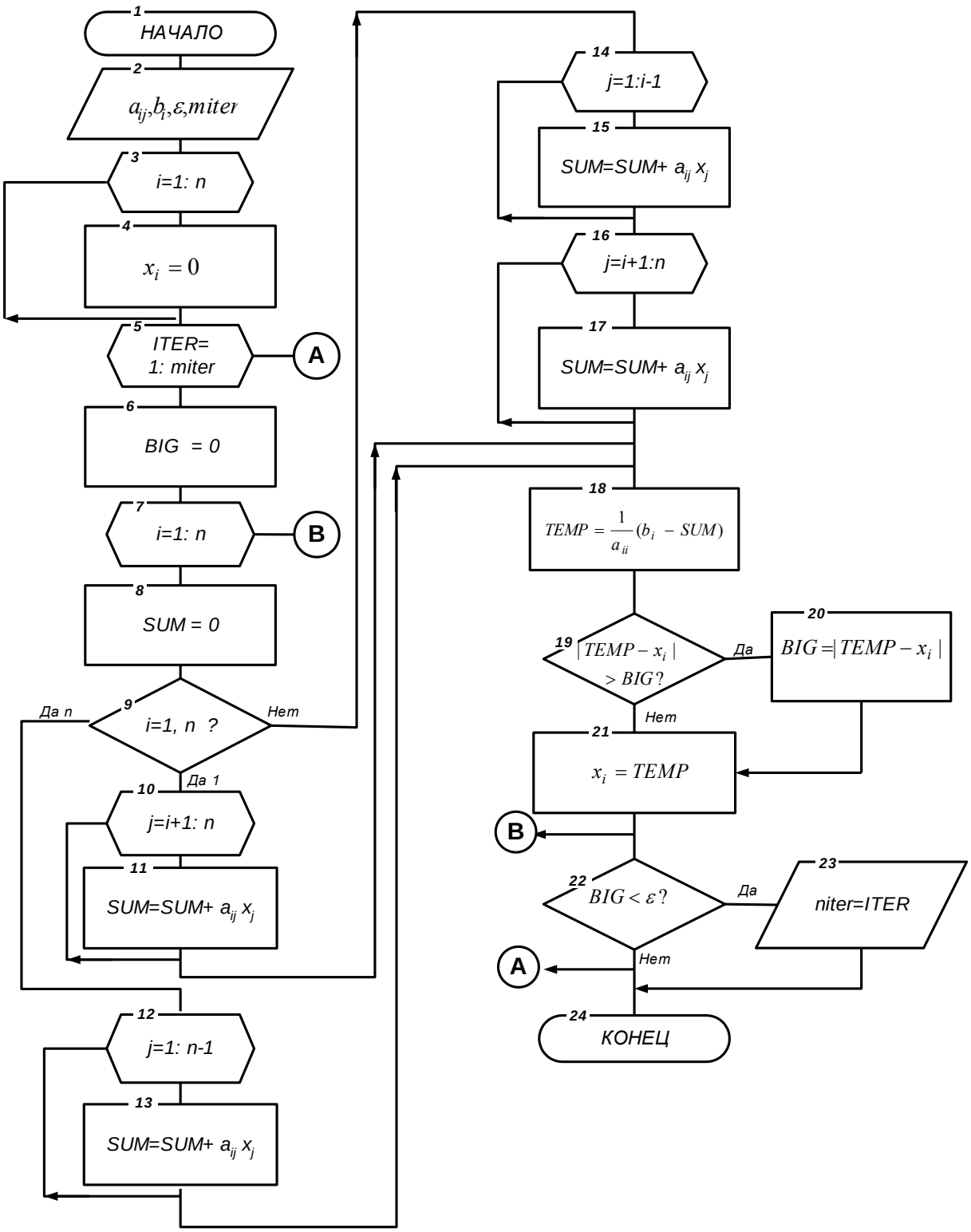
1
НАЧАЛО
2
$a_{ij}, b_i, \varepsilon, miter$
3
i=1: n
4
$x_i = 0$
5
ITER= 1: miter
A
6
BIG = 0
7
i=1: n
B
8
SUM = 0
9
i=1, n ?
Да n
Нет
Да 1
10
j=i+1: n
11
SUM=SUM+ a_{ij} x_j
12
j=1: n-1
13
SUM=SUM+ a_{ij} x_j
14
j=1:i-1
15
SUM=SUM+ a_{ij} x_j
16
j=i+1:n
17
SUM=SUM+ a_{ij} x_j
18
$TEMP = \frac{1}{a_{ii}}(b_i - SUM)$
19
$|TEMP - x_i| > BIG$?
Да
20
$BIG = |TEMP - x_i|$
Нет
21
$x_i = TEMP$
B
22
$BIG < \varepsilon$?
Да
23
niter=ITER
A
Нет
24
КОНЕЦ

Рис. 2.3.

2.2.5. Средства Matlab для решения СЛАУ

Для работы с матрицами в Matlab применяются общепринятые для матричной алгебры символы: +(плюс) или -(минус) для сложения или вычитания матриц, * (звездочка) для умножения матриц. Для возведения матрицы в степень используется символ ^, для транспонирования матрицы – символ ' (кавычка).

Например, для получения обратной матрицы можно использовать запись

$$A1 = A\text{^}{-1}. \qquad (2.9)$$

Это дает возможность получить решение СЛАУ с помощью обратной матрицы в виде (2.3). Для этого достаточно записать команду

$$X = (A\text{^}{-1}) * B. \qquad (2.10)$$

Кроме того, в Matlab имеется функция $INV(A)$ для обращения квадратной матрицы A. С помощью этой функции можно найти обратную матрицу, записав вместо (2.9) команду

$$A1 = INV(A),$$

или решить СЛАУ, записав вместо (2.10) команду

$$X = INV(A) * B.$$

Однако решение СЛАУ с помощью обратной матрицы связано с большим объемом вычислений. Более рациональным является использование операций матричного деления:

/ (наклонная черта или slash) для правого деления,

\ (обратная наклонная черта или backslash) для левого деления.

Запись $A \backslash B$ означает левое деление матрицы B на матрицу A. По смыслу это то же, что и $INV(A) * B$, однако расчеты выполняются по-другому. Запись

$$X = A \backslash B$$

означает решение СЛАУ (2.2) методом исключения Гаусса.

Запись B/A означает правое деление матрицы B на матрицу A. По смыслу это то же, что и $B*INV(A)$, однако расчеты выполняются по-другому. Более точно, $B/A=(A'\backslash B')'$ (см. левое деление). Запись

$$X=B/A$$

дает решение уравнения $XA=B$ методом исключения Гаусса.

2.3. Порядок выполнения работы

2.3.1. Написать m-файл-сценарий для решения СЛАУ n-го порядка (2.1) методом Гаусса без выбора главного элемента. Работу программы продемонстрировать на системе уравнений, выбранной из табл. 2.1 в соответствии с номером своей бригады и кодом подгруппы (а или б). Правильность решения подтвердить путем использования средств Matlab.

2.3.2. Написать m-файл-сценарий для решения СЛАУ n-го порядка методом Гаусса с выбором главного элемента. Работу программы продемонстрировать на той же системе уравнений. Правильность решения подтвердить путем использования средств Matlab.

2.3.3. Написать m-файл-сценарий для решения СЛАУ n-го порядка методом Гаусса–Зейделя. Работу программы продемонстрировать на той же системе уравнений. Правильность решения подтвердить путем использования средств Matlab.

Таблица 2.1

Системы уравнений для индивидуальных заданий

№ варианта	Матрица системы	Вектор правой части	№ варианта	Матрица системы	Вектор правой части
1	2	3	4	5	6
1а	-3 4 -5 -4 -1 -5 1 5 1	-9 -1 -9	1б	4 -2 0 -2 -4 3 4 -5 -5	2 -12 9
2а	1 4 -1 4 5 -3 1 2 -1	-14 -24 -8	2б	4 1 0 0 -1 -4 -4 1 -5	9 3 -15
3а	1 5 -1 0 0 -5 -2 1 -1	0 -10 5	3б	-5 2 -1 -1 -2 0 -4 5 2	-10 1 -3
4а	4 -5 -5 -3 -3 4 -5 2 -5	5 10 -9	4б	4 -3 -5 -5 4 3 0 3 -5	3 3 13
5а	-5 -5 5 -4 1 0 5 1 -5	-5 -1 9	5б	-5 -5 5 0 1 1 1 2 -1	0 2 2
6а	-3 1 2 -1 -1 4 -3 -3 -1	-4 -4 -12	6б	1 4 -5 5 3 -4 -2 2 1	3 6 6
7а	2 1 1 -1 3 -3 -4 3 3	4 -3 -18	7б	-5 3 3 1 -4 3 -5 -4 2	24 -1 15

8а	-2 -5 5	1	8б	3 5 -2	-18
	5 -1 5	7		3 4 2	-12
	-3 2 -5	-5		-3 1 -4	0
9а	-5 -3 -2	22	9б	4 2 3	-16
	5 -1 5	-24		2 -3 2	0
	0 -3 4	-5		-1 2 1	-1
10а	-3 5 1	-8	10б	-4 0 1	-11
	-2 4 -1	-11		-2 -5 -2	-3
	4 -3 -3	2		3 2 -1	11
11а	-3 3 2	0	11б	-2 3 4	8
	-2 1 0	-5		4 5 5	23
	-3 0 3	0		2 2 -1	1
12а	-5 3 -1	10	13б	-4 -3 5	8
	1 5 4	-17		-5 5 1	3
	1 -4 2	3		-5 4 2	3
13а	2 -1 4	11	13б	-5 5 -2	-28
	5 -4 -1	7		-2 3 -3	-12
	-1 -5 -5	-6		0 2 2	-8
14а	-3 -1 -1	11	14б	-5 4 4	-26
	-2 0 2	-2		2 -1 2	5
	1 -2 -4	14		1 -1 2	3
15а	-1 -3 2	-8	15б	5 -1 -3	-14
	2 3 -4	16		-4 -5 2	0
	1 1 -4	14		-3 -3 1	2

ЛАБОРАТОРНАЯ РАБОТА № 3. Аппроксимация функций

3.1. Цель работы

3.1.1. Ознакомление с задачей аппроксимации функций одной переменной, изучение задачи интерполирования функций.

3.1.2. Приобретение навыков программирования интерполяционных формул.

3.1.3. Приобретение навыков использования стандартных средств системы Matlab для интерполирования функций.

3.2. Теоретические положения

3.2.1. Понятие аппроксимации функций

Аппроксимация функции $y = f(x)$ – это замена этой функции другой более простой функцией $y = \varphi(x)$, близкой к $f(x)$ в некотором смысле. Критерий близости функций $f(x)$ и $\varphi(x)$ определяет способ аппроксимации.

Если расстояние ρ между функциями $f(x)$ и $\varphi(x)$ на некотором отрезке $[a,b]$ действительной прямой определить выражением

$$\rho(f(x),\varphi(x)) = \int_a^b (f(x) - \varphi(x))^2 \, dx,$$

то аппроксимация функции $f(x)$ по критерию минимума такого расстояния ρ будет называться аппроксимацией с минимальной интегральной квадратичной погрешностью.

Если критерий близости функций $f(x)$ и $\varphi(x)$ состоит в том, чтобы $f(x)$ и $\varphi(x)$ совпадали в дискретном ряде точек $x_0, x_1, \ldots, x_n$ отрезка $[a,b]$, то такой

способ аппроксимации функции $f(x)$ называется *интерполированием* функции $f(x)$.

Если расстояние ρ между функциями $f(x)$ и $\varphi(x)$ на некотором отрезке $[a,b]$ действительной прямой определить выражением

$$\rho(f(x),\varphi(x)) = \sum_{i=0}^{n}(f(x_i)-\varphi(x_i))^2,$$

то аппроксимация функции $f(x)$ по критерию минимума такого расстояния ρ будет называться аппроксимацией по методу наименьших квадратов.

3.2.2. Постановка задачи интерполирования функций

Задача интерполирования функции $f(x)$ на некотором отрезке $[a,b]$ формулируется следующим образом. На отрезке $[a,b]$ задано $n+1$ точек $x_0,x_1,\ldots,x_n \in [a,b]$, которые называют узлами. Обычно считают, что первая и последняя точки совпадают с концами отрезка $[a,b]$: $x_0 = a$, $x_n = b$. Известны значения $y_i = f(x_i)$ функции $f(x)$ в этих точках, $i = \overline{0,n}$. Требуется заменить эту функцию некоторой другой функцией $\varphi(x)$ таким образом, чтобы значения обеих функций совпадали в узлах, т.е. чтобы выполнялись равенства

$$\varphi(x_i) = f(x_i) = y_i, \quad i = \overline{0,n}.$$

Искомой неизвестной в данной задаче является функция $\varphi(x)$.

Сформулированную задачу иногда интерпретируют следующим образом. Некоторая функция $f(x)$ задана на отрезке $[a,b]$ таблицей своих значений

x_i	x_0	x_1	x_2	…	x_n
$y_i = f(x_i)$	y_0	y_1	y_2	…	y_n

и требуется найти способ определения значений этой функции в любых других точках отрезка $[a,b]$.

При формулировке задачи интерполирования обычно предполагают, что аппроксимирующая функция $\varphi(x)$ задана с точностью до $(n+1)$ параметров $a_0, a_1, \ldots, a_{n+1}$, т.е. в виде $\varphi(x, a_0, a_1, \ldots, a_n)$. Тогда нам необходимо отыскать неизвестные параметры $a_0, a_1, \ldots, a_n$ исходя из заданных равенств

$$\varphi(x_i, a_0, a_1, \ldots, a_n) = y_i, \quad i = \overline{0,n}. \tag{3.1}$$

Эти равенства можно рассматривать как систему $n+1$ уравнений относительно неизвестных коэффициентов $a_0, a_1, \ldots, a_n$.

Чаще всего функцию $\varphi(x, a_0, a_1, \ldots, a_n)$ представляют в виде полинома n-й степени

$$\varphi(x, a_0, a_1, \ldots, a_n) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n. \tag{3.2}$$

Тогда система уравнений (3.1) принимает следующий вид:

$$\begin{aligned} &a_0 + a_1 x_0 + a_2 x_0^2 + \ldots + a_n x_0^n = y_0, \\ &a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_n x_1^n = y_1, \\ &\quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ &a_0 + a_1 x_n + a_2 x_n^2 + \ldots + a_n x_n^n = y_n. \end{aligned} \tag{3.3}$$

Определитель этой системы имеет вид

$$\begin{vmatrix} 1 & x_0 & x_0^2 & \ldots x_0^n \\ 1 & x_1 & x_1^2 & \ldots x_1^n \\ \ldots & \ldots & \ldots & \ldots\ldots \\ 1 & x_n & x_n^2 & \ldots x_n^n \end{vmatrix}$$

и называется определителем Вандермонда на системе точек $x_0, x_1, \ldots, x_n$. Доказано, что если точки $x_0, x_1, \ldots, x_n$ попарно различны, что предполагается при постановке задачи, то определитель Вандермонда не равен нулю. Тогда система уравнений (3.3) имеет единственное решение, т.е. существует единственный полином (3.2) степени n, коэффициенты которого удовлетворяют системе уравнений (3.3). Этот полином называется интерполяционным полиномом для функции $f(x)$.

3.2.3. Интерполяционный полином Лагранжа

Интерполяционный полином может быть представлен в различных формах. Одной из них является форма Лагранжа. Полином Лагранжа имеет следующий вид:

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

или в компактной форме

$$P_n(x) = \sum_{i=0}^{n} y_i \cdot \prod_{\substack{k=0 \\ k\neq i}}^{n} \frac{(x-x_k)}{(x_i-x_k)}. \qquad (3.4)$$

Этот полином легко может быть получен путем следующих рассуждений. Рассмотрим так называемый полином влияния i-го узла:

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

$$= \prod_{\substack{k=0 \\ k\neq i}}^{n} \frac{(x-x_k)}{(x_i-x_k)}, \quad i = \overline{0,n}. \qquad (3.5)$$

Понятно, что это полином n-й степени. Из выражения (3.5) видно, что он равен нулю в любых точках, кроме x_i, а в точке x_i равен 1. Вид этого полинома приведен на рис. 3.1. Произведение $y_i L_i(x)$ есть полином n-й степени, который во всех узлах, кроме x_i, равен нулю, а в узле x_i равен y_i. Просуммировав произведения $y_i L_i(x)$ по всем i (по всем узлам), мы получим полином (3.4), удовлетворяющий постановке задачи интерполирования. Следовательно, построенный полином будет интерполяционным.

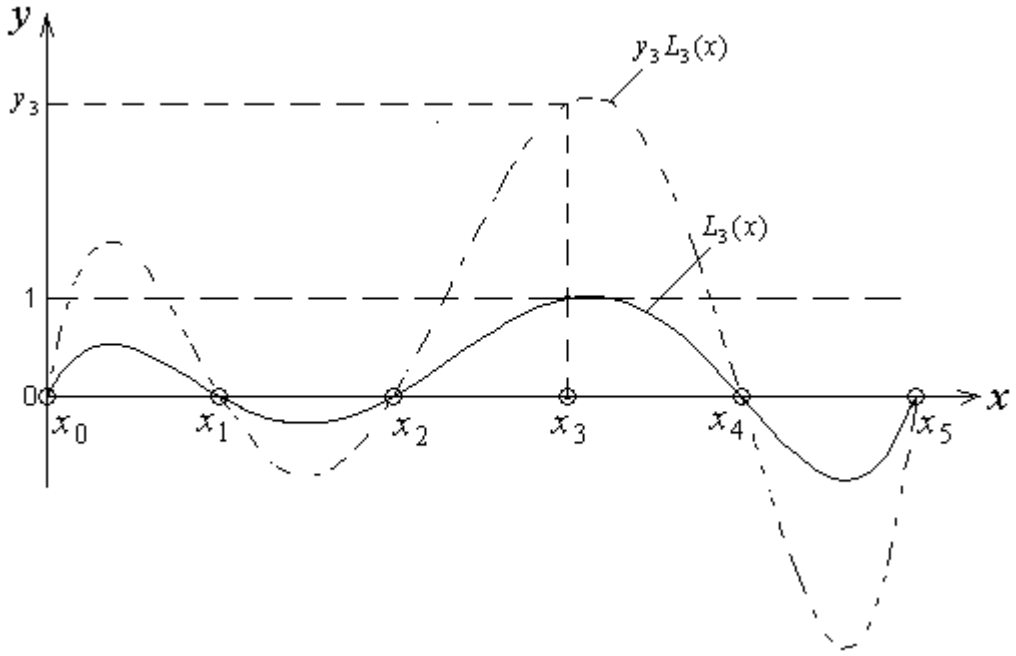

Рис. 3.1.

Полином Лагранжа можно записать в другом виде. Для этого рассмотрим полином

$$A(x) = (x - x_0)(x - x_1)\cdots(x - x_n) = \prod_{j=0}^{n}(x - x_j). \qquad (3.6)$$

Это полином $(n+1)$-й степени со старшим коэффициентом, равным 1, и обращающийся в нуль во всех узлах. Легко показать, что производная от этого полинома в точке $x = x_i$ имеет вид

$$A'(x_i) = (x - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n) = \prod_{\substack{j=0 \\ j\neq i}}^{n}(x - x_j).$$

Тогда полином влияния i-го узла (3.5) можно записать в виде

$$L_i(x) = \frac{A(x)}{A'(x_i)(x - x_i)},$$

а сам полином Лагранжа (3.4) – в виде

$$P_n(x) = \sum_{i=0}^{n} y_i \frac{A(x)}{A(x_i)(x - x_i)}. \qquad (3.7)$$

3.2.4. Погрешность интерполирования

Погрешность интерполяционного полинома Лагранжа оценивается выражением

$$| f(x) - P_n(x) | \le \frac{M_{(n+1)}}{(n+1)!} | A(x) |,$$

где $M_{(n+1)} = \max\limits_{x \in [a,b]} | f^{(n+1)}(x) |$ – максимальное по модулю значение $(n+1)$-й производной функции $f(x)$ на отрезке интерполирования $[a,b]$, $A(x)$ – полином (3.6).

3.2.5. Наилучший выбор узлов интерполирования

Чаще всего узлы интерполирования располагают на отрезке интерполирования с равномерным шагом. Вместе с тем, выбирая неравномерную сетку узлов, можно увеличить точность интерполирования. Задача о наилучшем выборе узлов интерполирования была решена Чебышевым. Наилучшие узлы интерполирования выбираются равными корням так называемого "полинома, наименее отклоняющегося от нуля" на отрезке интерполирования $[a,b]$. Полином, наименее отклоняющийся от нуля на отрезке $[-1,1]$, был найден Чебышевым и назван его именем. Полином Чебышева $T_n(x)$ определяется следующим выражением:

$$T_n = 2^{1-n} \cos(n \arccos z).$$

Следовательно, наилучшие узлы интерполирования $z_0, z_1, \ldots, z_n$ на отрезке $[-1,1]$ выбираются из условия

$$T_{n+1}(x) = 0$$

и имеют следующие значения:

$$z_i = \cos\frac{2i+1}{2(n+1)}\pi, \quad i = \overline{0,n}. \tag{3.8}$$

Наилучшие узлы интерполирования на произвольном отрезке $[a,b]$ находятся по формуле

$$x_i = \frac{b-a}{2} z_i + \frac{b+a}{2}, \; i = \overline{0,n}, \tag{3.9}$$

где z_i – узлы (3.8).

3.2.6. Средства Matlab для интерполирования функций

Для интерполирования функций в Matlab можно использовать функции **polyfit** и **polyval**.

Функция **a=polyfit(x, y, n)** находит массив коэффициентов a длиной $n+1$ полинома степени n по массивам длиной m узлов x и значений функции в узлах y, $m \geq n+1$. Этот полином аппроксимирует функцию $y(x)$ по методу наименьших квадратов. Предполагается, что полином задается в виде

$$y = a_1 x^n + a_2 x^{n-1} + \ldots + a_n x + a_{n+1}.$$

Если $m = n+1$, то программа возвращает коэффициенты интерполяционного полинома.

Функция **y=polyval(a, x)** возвращает значение y полинома в точке x, коэффициенты которого определены в векторе a.

Таким образом, для интерполирования функции необходимо последовательно обратиться к функциям **polyfit** и **polyval**.

3.3. Порядок выполнения работы

3.3.1. Написать m-файл-функцию для интерполирования функции $y = f(x)$ на отрезке $[a,b]$ с помощью полинома Лагранжа (3.4) при равномерной сетке

узлов. Входными параметрами m-файла-функции должны быть массив узлов интерполирования, массив значений интерполируемой функции в узлах и аргумент x, при котором вычисляется значение полинома Лагранжа (контрольная точка). Выходной параметр m-файла-функции – значение интерполяционного полинома в точке x.

3.3.2. Использовать написанную m-файл-функцию для интерполирования конкретной функции, взятой из табл. 1.1 лабораторной работы № 1 в соответствии с номером своей бригады и кодом подгруппы. Результаты интерполирования представить в виде графиков интерполируемой функции и интерполяционного полинома в одном графическом окне. Число узлов выбрать равным 3–5. Шаг между узлами интерполирования выбрать равномерным, шаг по переменной x при построении графиков функций выбрать кратным шагу между узлами интерполирования с тем, чтобы можно было наблюдать значения функции и полинома в узлах. Исследовать зависимость погрешности интерполирования от количества узлов интерполирования.

3.3.3. Выполнить интерполирование заданной функции с помощью стандартных средств Matlab и сравнить с результатами, полученными по собственным программам.

3.3.4. Выполнить интерполирование заданной функции с наилучшим выбором узлов интерполирования по формуле (3.9). Результаты сравнить с результатами, полученными на равномерной сетке узлов.

ЛАБОРАТОРНАЯ РАБОТА № 4. Численное интегрирование

4.1. Цель работы

4.1.1. Изучение задачи численного интегрирования функций.

4.1.2. Приобретение навыков программирования квадратурных формул.

4.1.3. Приобретение навыков использования стандартных средств системы Matlab для численного интегрирования функций.

4.2. Теоретические положения

4.2.1. Постановка задачи численного интегрирования

Задача численного интегрирования состоит в том, чтобы найти численное значение определенного интеграла

$$\mathrm{I} = \int_a^b f(x)dx, \qquad (4.1)$$

где $f(x)$ – функция, непрерывная на отрезке интегрирования $[a,b]$. Формулы для решения этой задачи называются квадратурными. Квадратурная формула позволяет вместо точного значения интеграла (4.1) найти некоторое его приближенное значение $\widetilde{I}$. Разность точного и приближенного значений интеграла называется абсолютной погрешностью квадратурной формулы (или численного метода),

$$R = I - \widetilde{I}.$$

Квадратурные формулы используют для вычисления интеграла (4.1) значения функции $f(x)$ в ряде точек отрезка $[a,b]$. Рассмотрим различные квадратурные формулы и их погрешности.

4.2.2. Методы прямоугольников

Разобьем отрезок интегрирования $[a,b]$ на n *частей* точками $x_0, x_1, \ldots, x_n$, как это показано на рис. 4.1. Заменим площадь криволинейной трапеции суммой площадей прямоугольников, построенных на частичных отрезках $[x_i, x_{i+1}]$, $i = \overline{0, n-1}$, как на основаниях. Если высоту i-го прямоугольника взять равной значению функции $f(x)$ в левой точке основания прямоугольника, т.е. принять

$$s_i = f(x_i)(x_{i+1} - x_i) = y_i h_i,$$

то мы получим квадратурную формулу левых прямоугольников

$$\widetilde{I} = \sum_{i=0}^{n-1} s_i = \sum_{i=0}^{n-1} y_i h_i .$$

Для равноотстоящих на величину h узлов,

$$h = \frac{b-a}{n},$$

формула левых прямоугольников имеет вид

$$\widetilde{I} = h \sum_{i=0}^{n-1} y_i = h(y_0 + y_1 + \ldots + y_{n-1}). \qquad (4.2)$$

Если интеграл на i-м отрезке $[x_i, x_{i+1}]$ заменить площадью прямоугольника с высотой, равной значению функции $f(x)$ в правой точке основания прямоугольника, т.е. принять

$$s_i = f(x_{i+1})(x_{i+1} - x_i) = y_{i+1} h_i,$$

то мы получим квадратурную формулу правых прямоугольников

$$\widetilde{I} = \sum_{i=0}^{n-1} s_i = \sum_{i=0}^{n-1} y_{i+1} h_i .$$

Для равноотстоящих на величину h узлов формула правых прямоугольников имеет вид:

$$\widetilde{I} = h\sum_{i=0}^{n-1} y_{i+1} = h(y_1 + y_2 + \ldots + y_n). \qquad (4.3)$$

Абсолютная погрешность метода прямоугольников для равномерной сетки значений аргумента оценивается неравенством

$$|R| \le M_1 \frac{b-a}{2} h,$$

где $M_1 = \max\limits_{x\in[a,b]} |f'(x)|$ – максимальное по модулю значение первой производной подынтегральной функции $f(x)$ на отрезке интегрирования $[a,b]$.

4.2.3. Метод трапеций

Заменим площадь криволинейной трапеции суммой площадей трапеций, построенных на частичных отрезках $[x_i, x_{i+1}]$, $i = \overline{0, n-1}$, (см. рис. 4.1),

$$\widetilde{I} = \sum_{i=0}^{n-1} s_i,$$

где

$$s_i = \frac{(f(x_i) + f(x_{i+1}))(x_{i+1} - x_i)}{2} = \frac{y_i + y_{i+1}}{2} h_i.$$

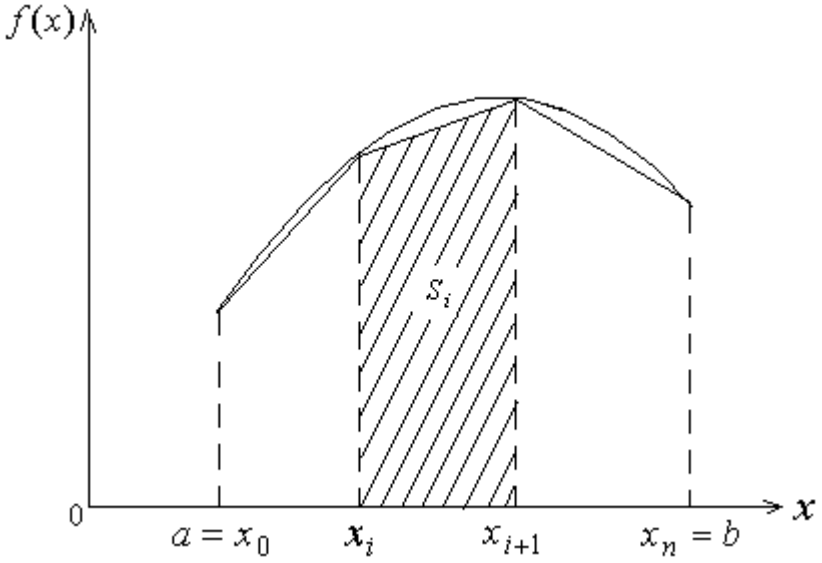

Рис. 4.1.

Получим квадратурную формулу трапеций:

$$\widetilde{I} = \sum_{i=0}^{n-1} \frac{y_i + y_{i+1}}{2} h_i \,.$$

Для равноотстоящих на величину h узлов формула трапеций имеет вид

$$\widetilde{I} = \frac{h}{2} \sum_{i=0}^{n-1} (y_i + y_{i+1}) = \frac{h}{2} (y_0 + 2y_1 + 2y_2 + \ldots + 2y_{n-1} + y_n) \,. \qquad (4.4)$$

Погрешность метода трапеций оценивается неравенством

$$|R| \le \frac{M_2 (b-a)}{12} h^2 ,$$

где $M_2 = \max\limits_{x \in [a,b]} |f''(x)|$ – максимальное по модулю значение второй производной подынтегральной функции $f(x)$ на отрезке интегрирования $[a,b]$.

4.2.4. Метод Симпсона

Разделим точки $x_0, x_1, \ldots, x_n$, разбивающие отрезок интегрирования $[a,b]$ на частичные отрезки с равномерным шагом h, на тройки точек x_0, x_1, x_2, $x_2, x_3, x_4, \ldots, x_{n-2}, x_{n-1}, x_n$. Для такого разбиения число n необходимо выбрать четным. На отрезке, определяемом i-й тройкой точек $x_{2i}, x_{2i+1}, x_{2i+2}$, $i = 0,1,2,\ldots,(n-1)/2$, заменим подынтегральную функцию параболой второго порядка $Bx^2 + Cx + D$, проходящей через точки (x_{2i}, y_{2i}), (x_{2i+1}, y_{2i+1}), (x_{2i+2}, y_{2i+2}), и заменим точное значение интеграла на этом отрезке интегралом s_i от полученной параболы. Можно показать, что

$$s_i = \frac{h}{3} (y_{2i} + 4y_{2i+1} + y_{2i+2}) .$$

Приближенное значение интеграла получим как сумму этих частичных интегралов:

$$\widetilde{I} = \sum_{i=0}^{(n-2)/2} s_i = \frac{h}{3} \sum_{i=0}^{(n-2)/2} (y_{2i} + 4y_{2i+1} + y_{2i+2}) =$$

$$= \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + \ldots + 2y_{n-2} + 4y_{n-1} + y_n). \qquad (4.5)$$

Погрешность метода Симпсона оценивается неравенством

$$|R| \le \frac{M_4(b-a)}{180} h^4,$$

где $M_4 = \max\limits_{x \in [a,b]} |f^{(4)}(x)|$ – максимальное по модулю значение четвертой производной подынтегральной функции $f(x)$ на отрезке интегрирования $[a,b]$.

4.2.5. Интерполяционные квадратурные формулы наивысшей алгебраической степени точности

Рассмотрим вычисление следующего интеграла

$$I = \int_a^b f(x)\rho(x)dx, \qquad (4.6)$$

где $\rho(x)$ – некоторая интегрируемая функция, которая называется весовой, $f(x)$ – функция, которую назовем подынтегральной. Этот интеграл является более общим по сравнению с рассматриваемым ранее интегралом (4.1). Интеграл вида (4.1) мы получим из (4.6) при весовой функции $\rho(x) = 1$.

Для вычисления интеграла (4.6) применим следующий подход: выберем на отрезке $[a,b]$ $n+1$ точек $x_0, x_1, \ldots, x_n$. В отличие от предыдущих методов, не будем вычислять интегралы на частичных отрезках, а заменим подынтегральную функцию на всем отрезке $[a,b]$ интерполяционным полиномом (3.7), построенным по узлам $x_0, x_1, \ldots, x_n$. В результате получим следующую квадратурную формулу

$$\widetilde{I} = \int_a^b \sum_{i=0}^{n} \frac{A(x)\rho(x)dx}{(x-x_i)A'(x_i)} y_i = \sum_{i=0}^{n} y_i \cdot \int_a^b \frac{A(x)\rho(x)dx}{(x-x_i)A'(x_i)} = \sum_{i=0}^{n} c_i y_i, \qquad (4.7)$$

где

$$c_i = \int_a^b \frac{A(x)\rho(x)}{(x - x_i)A'(x_i)}dx, \qquad (4.8)$$

$$y_i = f(x_i).$$

Формула (4.7), в которой коэффициенты определяются по выражению (4.8), называется *интерполяционной квадратурной формулой*. Эта формула точна для подынтегральных функций $f(x)$, представляющих собой полиномы до n-й степени включительно. В этом случае говорят, что степень точности интерполяционной квадратурной формулы (4.7) равна n.

Точность интерполяционной квадратурной формулы (4.7) можно существенно увеличить путем рационального выбора узлов $x_0, x_1, \ldots, x_n$. Рекомендуется выбирать узлы $x_0, x_1, \ldots, x_n$ равными корням полиномов, ортогональных на $[a,b]$ с весом $\rho(x)$. Интерполяционная квадратурная формула (4.7) с таким выбором узлов называется интерполяционной квадратурной формулой наивысшей алгебраической степени точности. Степень точности этой формулы равна $2n+1$. Мы видим, что рациональным выбором узлов мы увеличиваем точность интерполяционной квадратурной формулы более чем в 2 раза. Интерполяционная квадратурная формула (4.7) наивысшей алгебраической степени точности называется также квадратурной формулой Гаусса–Кристоффеля. Коэффициенты c_i (веса) (4.8) этой формулы называют числами Кристоффеля. Оптимальные узлы $x_0, x_1, \ldots, x_n$ и соответствующие им веса $c_0, c_1, \ldots, c_n$ рассчитываются заранее. Существуют таблицы узлов и весовых коэффициентов для различных весовых функций $\rho(x)$ [8,9].

Приведем примеры квадратурных формул наивысшей алгебраической степени точности.

4.2.5.1. Интегрирование функции по конечному отрезку

Интеграл с единичной весовой функцией $\rho(x)=1$ на конечном отрезке $[a,b]$, т.е. интеграл вида

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

линейной заменой переменных

$$x=\frac{b-a}{2}z+\frac{b+a}{2}$$

приводится к виду

$$I=\int_a^b f(x)dx=\frac{b-a}{2}\int_{-1}^{1} f(\frac{b-a}{2}z+\frac{b+a}{z})dz.$$

На отрезке $[-1,1]$ ортогональны с весом $\rho(x)=1$ полиномы Лежандра

$$P_n(z)=\frac{1}{2^n n!}\frac{d^n}{dz^n}(z^2-1)^n,\ n=0,1,2,\ldots.$$

Узлы $x_0,x_1,\ldots,x_n$ квадратурной формулы в этом случае выбираются равными корням полинома Лежандра $P_{n+1}(z)$. Квадратурная формула имеет вид

$$\int_a^b f(x)dx\approx\frac{b-a}{2}\sum_{i=0}^{n}c_i f(\frac{b-a}{2}z_i+\frac{b+a}{2}). \qquad (4.9)$$

В табл. 4.1 в качестве примера приведены узлы и коэффициенты для этой формулы при использовании двух, трех и четырех узлов.

Таблица 4.1.

Узлы и коэффициенты квадратурной формулы Гаусса–Лежандра

Число узлов $(n+1)$	Значения узлов z_i	Значения весовых коэффициентов c_i
2	$\pm 0{,}577350$	1
3	0 $\pm 0{,}774597$	$8/9$ $5/9$
4	$\pm 0{,}339981$ $\pm 0{,}861136$	0,652145 0,347855

4.2.5.2. Интегрирование функции по положительной полуоси

Пусть нужно вычислить интеграл вида

$$I = \int_0^\infty f(x)e^{-x}dx,$$

т.е. интеграл с весовой функцией $\rho(x) = e^{-x}$. На полуоси $(0,\infty)$ ортогональны с весом $\rho(x) = e^{-x}$ полиномы Лагерра

$$L_n(x) = e^x \frac{d^n}{dx^n}(x^n e^{-x}),\ n = 0,1,2,\ldots .$$

Узлы $x_0, x_1, \ldots, x_n$ квадратурной формулы в этом случае выбираются равными корням полинома Лагерра $L_{n+1}(x)$. Квадратурная формула имеет вид

$$\int_0^\infty f(x)e^{-x}dx \approx \sum_{i=0}^{n} c_i f(x_i). \qquad (4.10)$$

В табл. 4.2 приведены узлы и коэффициенты для этой формулы при использовании двух, трех и четырех узлов.

Таблица 4.2.

Узлы и коэффициенты квадратурной формулы Гаусса–Лагерра

Число узлов $(n+1)$	Значения узлов x_i	Значения весовых коэффициентов c_i
2	0,585786 3,414214	0,853553 0,146447
3	0,415775 2,294280 6,289945	0,711093 0,278518 0,0103893
4	0,322548 1,745761 4,536620 9,395071	0,603154 0,357419 0,0388879 0.000539295

4.2.5.3. Интегрирование функции по всей действительной прямой

Пусть нужно вычислить интеграл

$$I = \int_{-\infty}^{\infty} f(x) e^{-x^2} dx,$$

т.е. интеграл с весовой функцией $\rho(x) = e^{-x^2}$. На действительной прямой ортогональны с весом $\rho(x) = e^{-x^2}$ полиномы Эрмита

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n}(e^{-x^2}),\ n = 0,1,2,\ldots .$$

Узлы $x_0, x_1, \ldots, x_n$ квадратурной формулы в этом случае выбираются равными корням полинома Эрмита $H_{n+1}(x)$. Квадратурная формула имеет вид

$$\int_0^{\infty} f(x)e^{-x^2}dx \approx \sum_{i=0}^{n} c_i f(x_i). \qquad (4.11)$$

В табл. 4.3 приведены узлы и коэффициенты для этой формулы при использовании двух, трех и четырех узлов.

Таблица 4.3.

Узлы и коэффициенты квадратурной формулы Гаусса–Эрмита

Число узлов $(n+1)$	Значения узлов x_i	Значения весовых коэффициентов c_i
2	$\pm 0{,}707107$	0,886227
3	0 $\pm 1{,}224745$	1,181636 0,295409
4	$\pm 0{,}524648$ $\pm 1{,}650680$	0,804914 0,0813128

4.2.6. Средства Matlab для численного интегрирования

Для численного интегрирования в Matlab можно использовать функции **trapz, cumtrapz, quad, quad8**.

z = trapz(x,y) вычисляет значение интеграла от функции $y(x)$ методом трапеций по массиву **x** узлов и массиву **y** значений функции $y(x)$ в узлах.

z1 = trapz(y) вычисляет значение интеграла от функции $y(x)$ методом трапеций по массиву **y** значений функции $y(x)$ в узлах, используя единичный шаг по

аргументу x. Для получения значения интеграла при другом шаге необходимо полученное значение интеграла **z1** умножить на шаг.

zc = cumtrapz(x,y) вычисляет накапливающиеся (текущие) значения интеграла от функции $y(x)$ методом трапеций по массиву **x** узлов и массиву **y** значений функции $y(x)$ в узлах.

zc1 = cumtrapz(y) вычисляет накапливающиеся (текущие) значения интеграла от функции $y(x)$ методом трапеций по массиву **y** значений функции $y(x)$ в узлах, используя единичный шаг по аргументу x. Для получения значений интеграла при другом шаге необходимо полученные значения интеграла (массив **zc1)** умножить на шаг.

q = quad('fun',a,b) возвращает значение **q** определенного интеграла от заданной функции **'fun'** на отрезке **[a,b]**, полученное по адаптивному рекурсивному методу Симпсона с относительной погрешностью 10^{-3}. Подынтегральная функция должна оформляться в виде m-файла-функции **y=fun(x)**. Функция **fun** должна быть построена таким образом, чтобы она могла возвращать массив **y** значений выходной величины, если **x** – массив значений аргумента. Например, если подынтегральная функция имеет вид x^2, то можно оформить следующую m-файл-функцию:

```
function y=fun(x)
y=x.^2;
```

q = quad('fun',a,b,tol) возвращает значение **q** определенного интеграла, полученное с относительной погрешностью **tol**. Можно также использовать вектор,

состоящий из двух элементов **tol=[rel_tol, abs_tol]** для определения комбинации относительной и абсолютной погрешностей.

q = quad('fun',a,b,tol, trace) возвращает значение **q** определенного интеграла, полученное с относительной погрешностью **tol**, и при значении **tol**, не равном нулю, строит точечный график интегрируемой функции.

q = quad('fun',a,b,tol, trace, p1,p2,…) возвращает значение **q** определенного интеграла с использованием указанных выше возможностей, а также параметров **p1,p2,…**, которые напрямую передаются в определенную подынтегральную функцию **y = fun(x,p1,p2,…)**.

Если достигается чрезмерный уровень рекурсии, то функция **quad** возвращает q = Inf, что может означать сингулярность интеграла.

Функция **quad8** выполняет интегрирование по формулам Ньютона–Котеса 8-го порядка. Она имеет те же параметры, что и функция **quad:**

q = quad8('fun',a,b,tol,trace,p1,p2,…)

4.3. Порядок выполнения работы

4.3.1. Написать m-файлы-функции для интегрирования функции $y = f(x)$ на отрезке $[a,b]$ методами левых прямоугольников, правых прямоугольников, трапеций, Симпсона при равномерной сетке узлов. Входными параметрами m-файла-функции должны быть массив узлов интегрирования и массив значений интегрируемой функции в узлах. Выходной параметр m-файл-функции – значение интеграла.

4.3.2. Использовать написанные m-файл-функции для интегрирования конкретной функции, взятой из табл. 1.1 лабораторной работы № 1 в соответствии

с номером своей бригады и кодом подгруппы. Сравнить результаты интегрирования различными методами при одном и том же шаге интегрирования.

4.3.3. Выполнить интегрирование заданной функции с помощью стандартных средств Matlab и сравнить с результатами, полученными по собственным программам.

4.3.4. Написать m-файлы-функции для вычисления интегралов вида (4.6) по квадратурным формулам наивысшей алгебраической степени точности Гаусса–Лежандра (4.9), Гаусса–Лагерра (4.10) и Гаусса–Эрмита (4.11), используя для этого 3 или 4 узла. С их помощью вычислить эти интегралы для своей функции $y = f(x)$.

ЛАБОРАТОРНАЯ РАБОТА № 5. Решение нелинейных уравнений

5.1. Цель работы

5.1.1. Изучение задачи численного решения нелинейных уравнений.

5.1.2. Приобретение навыков программирования методов численного решения нелинейных уравнений.

5.1.3. Приобретение навыков использования стандартных средств системы Matlab для численного решения нелинейных уравнений.

5.2. Теоретические положения

5.2.1. Постановка задачи численного решения нелинейных уравнений.

Требуется решить уравнение

$$f(x) = 0, \tag{5.1}$$

где $f(x)$ – нелинейная непрерывная функция. Аналитическое решение данного уравнения можно получить лишь в простейших случаях. В большинстве случаев такие уравнения приходится решать численными методами. Решение этой задачи распадается на следующие этапы.

1. Установить количество, характер и расположение корней уравнения.

2. Найти приближенные значения корней, т.е. указать промежутки, в которых находятся корни (отделить корни).

3. Найти значения корней с требуемой точностью (уточнить корни).

Предположим, что первые два этапа выполнены, и рассмотрим численные методы уточнения корней.

5.2.2. Метод деления отрезка пополам

Предполагается известным отрезок $[a,b]$, на котором расположен искомый корень. В этом случае $f(a)\cdot f(b)<0$. Метод состоит в последовательном делении отрезка пополам и отбрасывании той половины, на которой нет корня, до тех пор, пока длина отрезка не станет малой (см. рис. 5.1). Этот алгоритм можно описать следующим образом:

1) если $\dfrac{|a-b|}{|b|}>\varepsilon$, то перейти к п.2), иначе перейти к п.5);

2) найти $x=\dfrac{a+b}{2}$;

3) если $f(a)\cdot f(x)<0$, то положить $b=x$, иначе положить $a=x$;

4) перейти к п.1);

5) найти значение корня $x_0=\dfrac{a+b}{2}$ и прекратить вычисления.

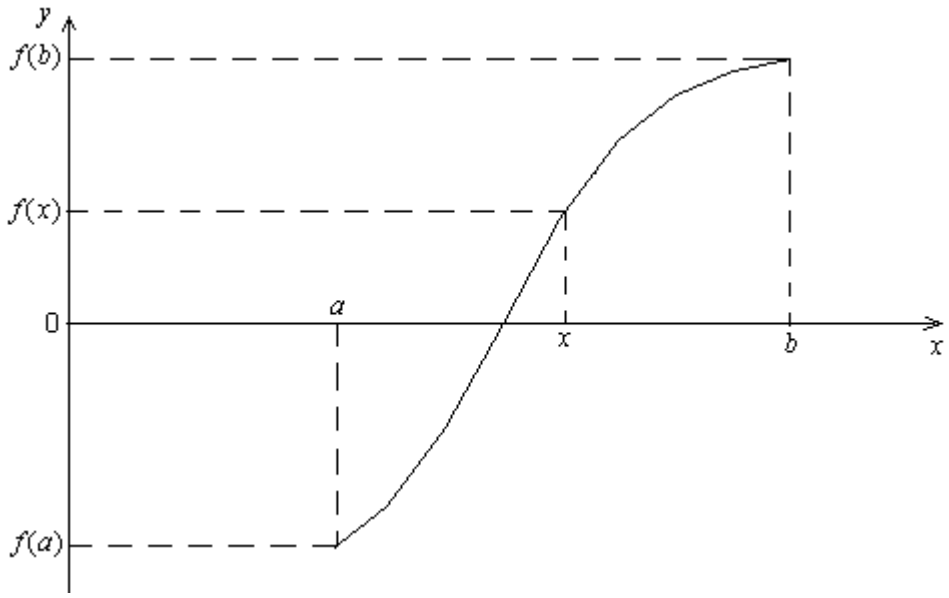

Рис. 5.1.

В описанном алгоритме число $\varepsilon>0$ определяет допустимую относительную погрешность нахождения корня.

Выполнение п.п. 1–4 называется одной итерацией.

Метод сходится для всех непрерывных функций, в том числе и для недифференцируемых. Однако для нахождения корня требуется выполнить большое число итераций.

5.2.3. Метод простой итерации

Преобразуем уравнение (5.1) следующим образом. Умножим обе части уравнения на некоторую функцию $\psi(x) \neq 0$. Получим эквивалентное уравнение

$$-\psi(x) f(x) = 0.$$

Прибавив к обеим частям последнего уравнения x, мы получим уравнение

$$x = \varphi(x), \tag{5.2}$$

где

$$\varphi(x) = x - \psi(x) f(x). \tag{5.3}$$

Организуем итерационный вычислительный процесс по формуле

$$x_{n+1} = \varphi(x_n), \;\; n = 1,2,\ldots, \tag{5.4}$$

где x_n – начальное значение корня уравнения, x_{n+1} – последующее уточненное значение корня. Формула (5.4) и есть алгоритм метода простой итерации. Он иллюстрируется на рис. 5.2. Корень a уравнения (5.2) определяется как точка пересечения кривой $y = \varphi(x)$ и прямой $y = x$. В обозначениях исходного уравнения (5.1) алгоритм простой итерации записывается в виде

$$x_{n+1} = x_n - \psi(x_n) f(x_n). \tag{5.5}$$

Выполнение расчетов по формуле (5.4) называется одной итерацией. Итерации прекращают, когда

$$\frac{|x_{n+1} - x_n|}{|x_{n+1}|} < \varepsilon, \tag{5.6}$$

где число $\varepsilon > 0$ определяет допустимую относительную погрешность нахождения корня.

Метод простой итерации сходится, если

$$|\varphi'(x)|<1$$

для значений x на всех итерациях. Условие сходимости можно записать в виде

$$|(x-\psi(x)\cdot f(x))'|<1.$$

Выбором функции $\psi(x)$ в формуле для итераций (5.5) можно влиять на сходимость метода. В простейшем случае $\psi(x)=const$ со знаком плюс или минус.

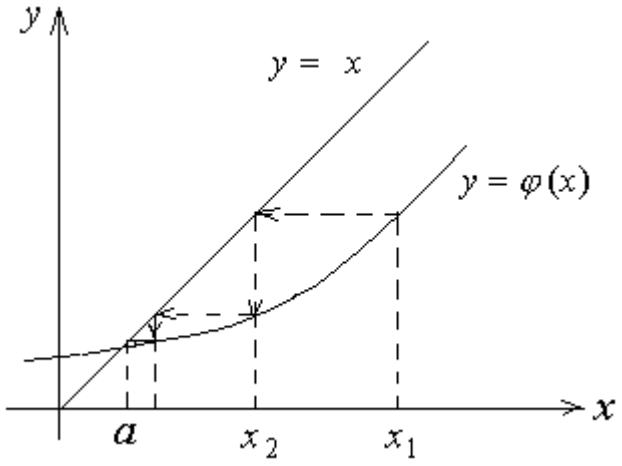

Рис. 5.2.

5.2.4. Метод Ньютона

Итерации по методу Ньютона осуществляются по формуле

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)},\ \ n=1,2,\dots\ . \tag{5.7}$$

Итерации прекращаются, если выполняется условие (5.6).

Сравнивая формулу (5.7) с формулой итераций (5.5) для метода простой итерации, приходим к выводу, что метод Ньютона – это метод простой итерации с функцией

$$\psi(x)=\frac{1}{f'(x)}.$$

Отсюда получаем условие сходимости метода:

$$|\varphi'(x)|<1, \tag{5.8}$$

где

$$\varphi(x) = x - \frac{f(x)}{f'(x)}. \tag{5.9}$$

Условие (5.8) с учетом (5.9) можно записать в виде

$$|\varphi'(x)| = \left| \frac{f''(x) \cdot f(x)}{[f'(x)]^2} \right| < 1. \tag{5.10}$$

Последнее неравенство должно выполняться для значений x на всех возможных итерациях. Детальный анализ условия сходимости (5.10) позволяет сделать вывод, что метод Ньютона сходится, если первоначальное приближение выбрано достаточно близко к корню.

На рис. 5.3 приведена графическая иллюстрация метода Ньютона. В точке x_n к кривой $f(x)$ проводится касательная, точка пересечения которой с осью абсцисс дает нам новое приближение x_{n+1}. Затем в точке x_{n+1} опять проводится касательная, и т.д.

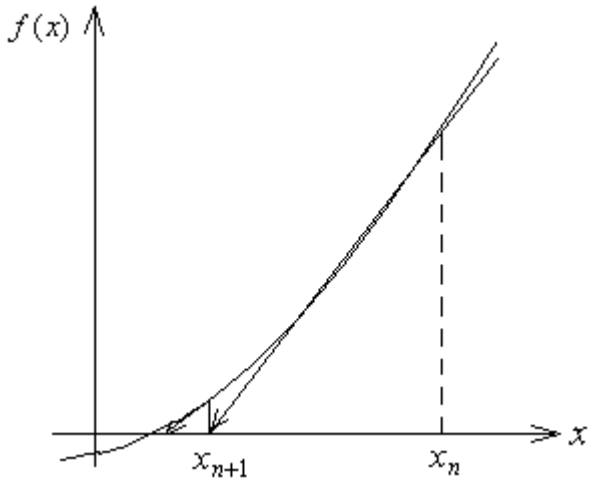

Рис. 5.3.

5.2.5. Метод секущих

В методе Ньютона необходимо знать не только функцию $f(x)$, нуль которой мы ищем, но и производную этой функции $f'(x)$, что является недостатком

данного метода. Если в методе Ньютона заменить производную на отношение приращений функции и аргумента, т.е. принять

$$f'(x_n) = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}},$$

то получим следующую формулу итераций:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})},\ n = 1,2,\ldots . \qquad (5.11)$$

Итерации прекращаются, если выполняется условие (5.6). Формула (5.11) называется методом секущих для нахождения корня уравнения (5.1). Графическая иллюстрация метода секущих приведена на рис. 5.4. Через две точки, определяемые приближениями x_{n-1} и x_n, проводится секущая и по ней определяется новое приближение x_{n+1}. Затем проводится следующая секущая через точки, определяемые приближениями x_n и x_{n+1}, и т.д. В отличие от предыдущих методов этот метод является двухшаговым. Это значит, что для определения нового приближения необходимо иметь два предыдущих приближения.

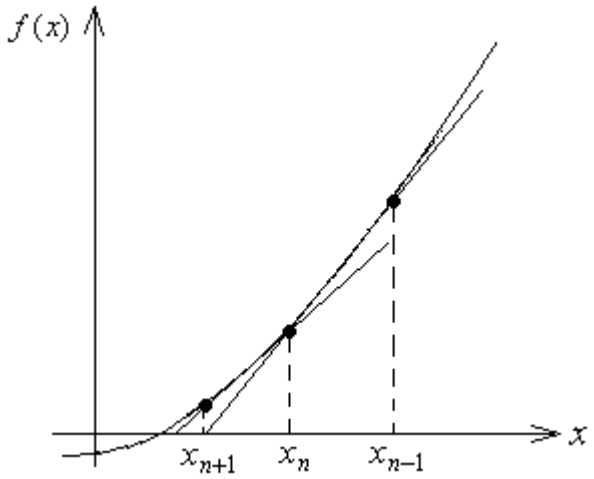

Рис. 5.4.

5.2.6. Средства Matlab для вычисления нулей функции одной переменной

Поставленную задачу в Matlab решает функция **fzero.**

b=fzero('fun',x) возвращает уточненное значение **b**, при котором достигается нуль функции **fun** при начальном значении аргумента **x**. Возвращенное значение близко к точке, где функция меняет знак, или равно **NaN**, если такая точка не найдена. Функция **fun** должна оформляться в виде m-файла-функции **y=fun(x)**.

b=fzero('fun',x) возвращает значение, при котором функция равна нулю, с возможностью задания интервала поиска с помощью вектора **x=[x1,x2]** длиной 2 такого, что **fun(x1)** и **fun(x2)** имеют разные знаки. Если это не так, то выдается сообщение об ошибке.

b=fzero('fun',x,tol) возвращает значение, при котором функция равна нулю, с заданной относительной погрешностью **tol**.

b=fzero('fun',x,tol,trace) выдает на экран информацию по каждой итерации, когда **tol** не нулевой.

b=fzero('fun',x,tol,trace,p1,p2,…) предусматривает дополнительные аргументы, передаваемые в функцию **y=fun(x,p1,p2,…)**.

Чтобы использовать значения **tol** или **trace** по умолчанию, необходимо ввести пустые матрицы, например, **b=fzero('fun',x,[],[],p1,p2,…)**.

В функции **fzero** ноль рассматривается как точка, где график функции **fun** пересекает ось x, а не касается ее. В зависимости от формы задания функции **fzero** реализуются следующие методы поиска нуля функции: деления отрезка пополам, секущих, обратного квадратичного интерполирования.

5.3. Порядок выполнения работы

5.3.1. Написать m-файлы-функции для нахождения нуля функции $f(x)$ методами деления отрезка пополам (п. 2.2), простой итерации (4.5), Ньютона (4.7)

и секущих (4.11) с заданной относительной погрешностью ε. Предусмотреть вывод числа итераций.

5.3.2. Использовать написанные m-файлы-функции для поиска нуля функции, взятой из табл. 1.1 лабораторной работы № 1 в соответствии с номером своей бригады и кодом подгруппы. Сравнить результаты поиска различными методами по числу использованных итераций.

5.3.3. Выполнить поиск нуля заданной функции с помощью функции Matlab **fzero**.

ЛАБОРАТОРНАЯ РАБОТА № 6. Решение обыкновенных дифференциальных уравнений

6.1. Цель работы

6.1.1. Изучение методов численного решения задачи Коши для обыкновенного дифференциального уравнения первого порядка.

6.1.2. Приобретение навыков программирования методов численного решения задачи Коши для обыкновенного дифференциального уравнения первого порядка.

6.1.3. Приобретение навыков использования стандартных средств системы Matlab для численного решения обыкновенного дифференциального уравнения первого порядка.

6.2. Теоретические положения

6.2.1. Постановка задачи

Соотношение вида

$$F(x, y, y', y'', \ldots, y^{(n)}) = 0, \qquad (6.1)$$

где F – некоторая функция независимой переменной x, функции $y = y(x)$ и ее производных $y' = y'(x) = \frac{dy(x)}{dx}$, $y'' = y''(x) = \frac{dy^2(x)}{dx^2}$, …, $y^{(n)} = \frac{d^n y(x)}{dx^n}$, называется обыкновенным дифференциальным уравнением n-го порядка. Решить уравнение – значит найти функцию $y(x)$, превращающую равенство (6.1) в тождество. Существует понятие *общего* и *частного* решения этого дифференциального уравнения. *Общее* решение (общий интеграл) – это формула, дающая все решения данного уравнения. Обычно общее решение обыкновенного диф-

ференциального уравнения n-го порядка (6.1) зависит от n постоянных $C_1, C_2, \ldots, C_n$, которые могут выбираться произвольно. Решение, которое не зависит от произвольных постоянных, называется *частным* решением дифференциального уравнения (частным интегралом). График каждого частного решения называется *интегральной кривой*. Чаще всего для обыкновенного дифференциального уравнения (6.1) формулируется так называемая задача Коши, когда дополнительно к уравнению (6.1) задают значения функции и ее производных до $(n-1)$-го порядка в некоторой точке x_0. Эти дополнительные данные называются начальными условиями. Наличие начальных условий позволяет получить частное решение дифференциального уравнения.

Процесс решения дифференциального уравнения называется его *интегрированием*. Интегрирование дифференциального уравнения вовсе не означает, что этот процесс сводится к вычислению интеграла. Если же решение дифференциального уравнения действительно свелось к вычислению интеграла, то говорят, что уравнение решено *в квадратурах*.

Методы решения дифференциальных уравнений бывают точные, приближенные и численные. Точные методы дают решение, которое можно выразить через элементарные функции. Получить точное решение дифференциального уравнения можно не всегда. Например, решение уравнения $y' = x^2 + y^2$ не выражается через элементарные функции. Приближенные методы дают решение в виде некоторой последовательности функций $y_m(x)$, сходящейся к решению $y(x)$ при $m \to \infty$. Численные методы дают решение в виде таблицы значений функции $y(x)$. Мы будем заниматься численными методами решения дифференциальных уравнений. В данной работе рассмотрим методы решения дифференциального уравнения первого порядка

$$y' = f(x, y) \tag{6.2}$$

с начальным условием $y(x_0) = y_0$.

6.2.2. Метод Эйлера

Этот метод решения уравнения (6.2) состоит в последовательных расчетах по формуле

$$y_{m+1} = y_m + h \cdot f(x_m, y_m), \tag{6.3}$$

начиная с точки (x_0, y_0), заданной начальными условиями x_0, $y(x_0) = y_0$. Здесь h – шаг интегрирования по независимой переменной x.

Формула Эйлера (6.3) иллюстрируется рис. 6.1. В точке (x_m, y_m) проводится касательная к интегральной кривой $y(x)$, новое значение функции y_{m+1} определяется как точка пересечения касательной с вертикальной прямой $x = x_{m+1} = x_m + h$. Следующая касательная проводится в точке (x_{m+1}, y_{m+1}). Из рисунка видно, что при возрастании переменной x погрешность в определении функции накапливается. Погрешность формулы Эйлера имеет порядок h^2.

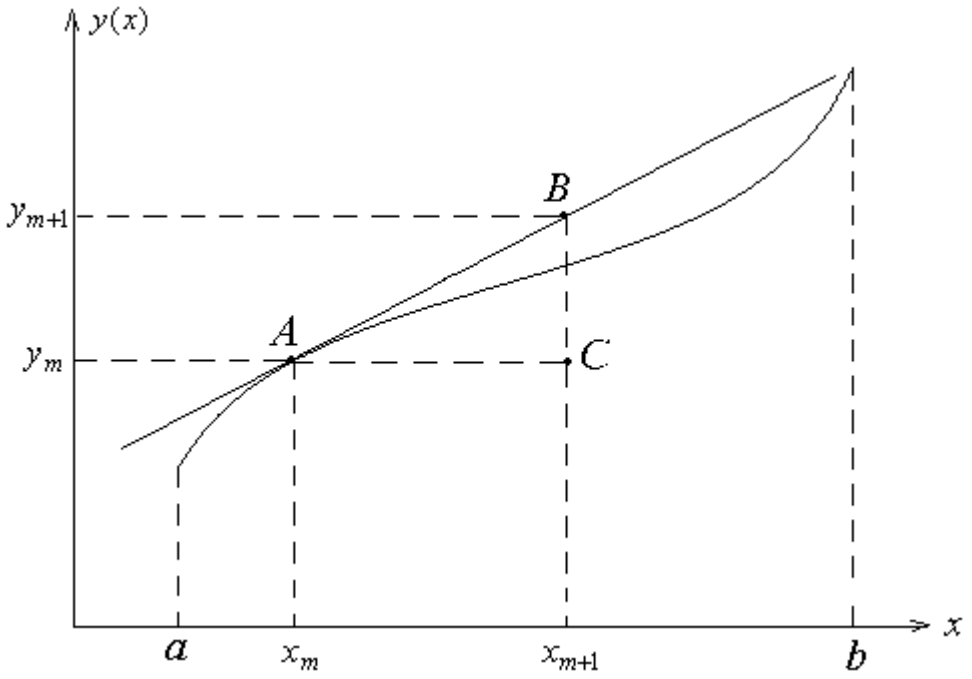

Рис. 6.1.

6.2.3. Метод Рунге–Кутта 2-го порядка

Этот метод состоит в последовательных расчетах по формулам

$$k_1 = f(x_m, y_m),$$
$$k_2 = f(x_m + h, y_m + hk_1), \qquad (6.4)$$
$$y_{m+1} = y_m + \frac{h}{2}(k_1 + k_2),$$

начиная с точки (x_0, y_0).

Формулы Рунге–Кутта 2-го порядка (6.4) иллюстрируются рис. 6.2.

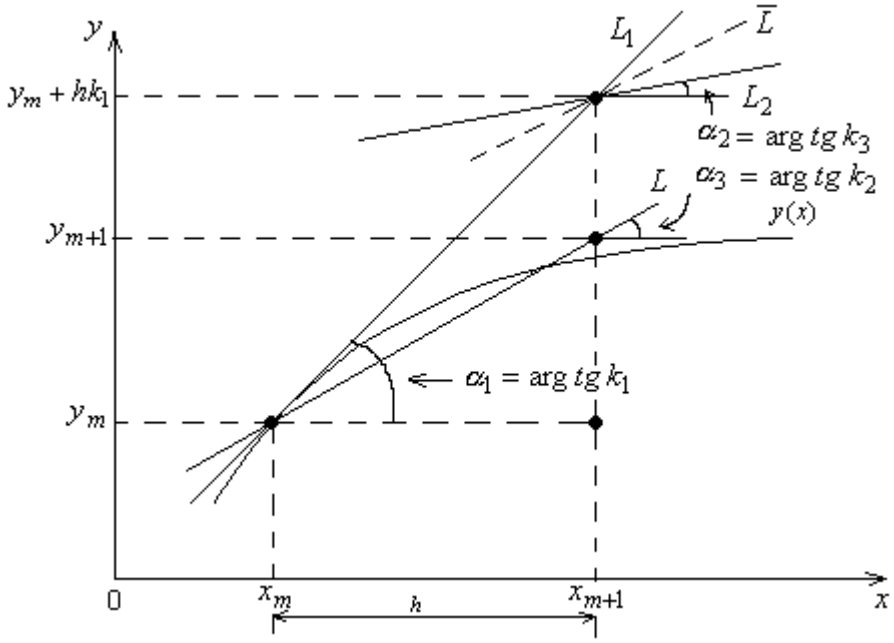

Рис. 6.2.

В точке (x_m, y_m) проводится касательная к интегральной кривой $y = y(x)$ (прямая L_1) и определяется тангенс угла наклона (угловой коэффициент) этой касательной k_1. Аналогично методу Эйлера определяется новая точка $(x_m + h, y_m + hk_1)$. В этой точке проводится касательная с угловым коэффициентом k_2 (прямая L_2). Новое значение функции y_{m+1} определяется как точка пересечения касательной с усредненным угловым коэффициентом

$$k_3 = \frac{k_1 + k_2}{2}$$

(прямая L, параллельная прямой $\overline{L}$) и вертикальной прямой $x = x_{m+1} = x_m + h$.

Метод Рунге–Кутта 2-го порядка (6.4) имеет погрешность порядка kh^3.

6.2.4. Метод Рунге–Кутта 4-го порядка

Это метод состоит в последовательных расчетах по формулам

$$k_1 = f(x_m, y_m),$$

$$k_2 = f(x_m + \frac{h}{2}, y_m + \frac{h}{2}k_1),$$

$$k_3 = f(x_m + \frac{h}{2}, y_m + \frac{h}{2}k_2), \qquad (6.5)$$

$$k_4 = f(x_m + h, y_m + hk_3),$$

$$y_{m+1} = y_m + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

начиная с точки (x_0, y_0).

Метод Рунге–Кутта 4-го порядка (6.5) имеет погрешность порядка kh^5.

Рассмотренные методы называются одношаговыми, так как для получения нового значения интегральной кривой достаточно знать лишь одно ее предыдущее значение.

6.2.5. Средства Matlab для решения обыкновенных дифференциальных уравнений

Для решения обыкновенного дифференциального уравнения первого порядка (6.2) в Matlab реализованы различные методы. Укажем имена m-файлов-функций, реализующих лишь два из них:

ode34 – одношаговые явные методы Рунге–Кутта 2-го и 4-го порядков;

ode45 – одношаговые явные методы Рунге–Кутта 4-го и 5-го порядков.

Приведем описание m-файла-функции для решения обыкновенного дифференциального уравнения (6.2). Поскольку все методы реализованы в Matlab в одном стиле, то вместо конкретного имени m-файла-функции будем использовать обобщенное имя **solver** (решатель), понимая под ним **ode34** или **ode45**.

[T,y]=solver('f', tspan, y0) интегрирует уравнение вида

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

от **t0** до **tfinal** с начальными условиями **y0**. Здесь **'f'** – строка, содержащая имя m-файла-функции, реализующей правую часть уравнения – функцию $f(t, y)$, оформленную в виде

function v=f(t,y),

tspan – массив из двух элементов, **tspan=[t0, tfinal]**. **T** – вектор значений аргумента t, при которых вычисляются значения функции $y(t)$, **y** – вектор значений функции $y(t)$. Для получения решения в конкретных точках **t0, t1,…, tfinal** (расположенных в порядке увеличения или уменьшения) можно использовать **tspan=[t0, t1, …, tfinal]**.

[T,y]=solver('f', tspan, y0, options) дает решение, подобное описанному выше, но с параметрами, определяемыми значениями аргумента **options**, созданного функцией **odeset**. Обычно используемые опции включают допустимое значение относительной погрешности **RelTol** (10^{-3} по умолчанию) и допустимое значение абсолютной погрешности **AbsTol** (10^{-6} по умолчанию).

В данном пособии использование функции **odeset** не предполагается и она не описывается.

[T,y]=solver('f', tspan, y0, options, p1, p2,…) дает решение, подобное описанному выше, помещая дополнительные параметры **p1, p2,…** в m-файл **f** всякий раз, когда он используется. При этом файл **f** должен быть оформлен в виде

function v=f(t, y, flag, p1, p2,…).

Если никакие опции не установлены, то необходимо использовать **options=[]**.

При обращении к функции **solver** без указания выходных параметров по умолчанию вызывается функция **odeplot** для построения графика вычисленного решения.

6.3. Порядок выполнения работы

Написать m-файл-сценарий для решения дифференциального уравнения первого порядка изложенными выше методами. Дифференциальное уравнение взять из табл. 6.1 в соответствии с номером своей бригады и кодом подгруппы. Получить также решение уравнения с помощью функций **ode34** и **ode45**. Все решения вывести в виде графиков в одно графическое окно.

Таблица 6.1

Дифференциальные уравнения первого порядка

№ варианта	Уравнение	№ варианта	Уравнение
1	2	3	4
1а	$y' = x + 2y,\ y(0) = 1$	1б	$y' = e^{-x},\ y(0) = 5$
2а	$y' = \frac{xy - y^2}{x^2 - 2xy},\ y(1) = 1$	2б	$y' = \frac{x + y}{x - y},\ y(1) = 0$
3а	$y' = \frac{2y}{x},\ y(1) = 0$	3б	$y' = \frac{x - y}{x + y},\ y(1) = 0$
4а	$y' = -\frac{x}{y},\ y(0) = 5$	4б	$y' = 2y + 3,\ y(0) = 3$
5а	$y' = 2y^2 + y,\ y(0) = 3$	5б	$y' = e^x + 1,\ y(0) = 0$
6а	$y' = x + 2y^2,\ y(0) = 0$	6б	$y' = x^2 y + x^3,\ y(1) = 0$

7а	$y' = x + \frac{xy}{x^2+1},\ y(0)=1$	7б	$y' = \frac{y}{x-1} + \frac{y^2}{x-1},\ y(0)=1$
8а	$y' = \frac{y}{y^2+x},\ y(1)=1$	8б	$y' = \frac{\cos x}{x},\ y(1)=1$
9а	$y' = x^2 + y^2,\ y(0)=-1$	9б	$y' = x^3 + y^2,\ y(0)=1$
10а	$y' = x^3 - y^2,\ y(0)=-1$	10б	$y' = x^2 + y^3,\ y(0)=0$
11а	$y' = x^3 + y^3,\ y(0)=0$	11б	$y' = x^3 - y^3,\ y(0)=1$
12а	$y' = x + \frac{y}{x},\ y(1)=0$	12б	$y' = 1 + x^2 + \frac{2xy}{x^2+1},\ y(0)=1$
13а	$y' = \frac{1}{\ln x},\ y(2)=1$	13б	$y' = \frac{1}{x+y},\ y(0)=-1$
14а	$y' = e^{-x^2},\ y(0)=1$	14б	$y' = y - x^4,\ y(0)=1$
15а	$y' = 3x^2 - y^2,\ y(1)=1$	15б	$y' = x^3 + 2y^2,\ y(0)=1$

ЛАБОРАТОРНАЯ РАБОТА № 7. Решение систем обыкновенных дифференциальных уравнений

7.1. Цель работы

7.1.1. Изучение методов численного решения задачи Коши для системы обыкновенных дифференциальных уравнений первого порядка.

7.1.2. Приобретение навыков программирования методов численного решения задачи Коши для системы обыкновенных дифференциальных уравнений первого порядка.

7.1.3. Приобретение навыков использования стандартных средств Matlab для численного решения системы обыкновенных дифференциальных уравнений первого порядка.

7.2. Теоретические положения

7.2.1. Постановка задачи

Системой обыкновенных дифференциальных уравнений первого порядка называется совокупность дифференциальных уравнений следующего вида:

$$\begin{cases} y_1'(x) = f_1(x, y_1, y_2, \ldots, y_n), \\ y_2'(x) = f_2(x, y_1, y_2, \ldots, y_n), \\ \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ y_n'(x) = f_n(x, y_1, y_2, \ldots, y_n). \end{cases} \tag{7.1}$$

Неизвестными здесь являются функции $y_1(x)$, $y_2(x)$, …, $y_n(x)$ независимой переменной x, а $y_1'(x)$, $y_2'(x)$, …, $y_n'(x)$ – их производные. Задача Коши для данной системы дифференциальных уравнений формулируется следующим об-

разом: найти функции $y_1(x)$, $y_2(x)$, …, $y_n(x)$, удовлетворяющие равенствам (7.1) и начальным условиям

$$\begin{aligned} y_1(x_0) &= y_{1,0}, \\ y_2(x_0) &= y_{2,0}, \\ &\cdots\ \cdots \\ y_n(x_0) &= y_{n,0}. \end{aligned} \tag{7.2}$$

Обычно для записи системы дифференциальных уравнений (7.1) используется векторная форма, для чего данные организуются в виде векторов. Введем в рассмотрение векторную функцию – вектор-столбец

$$\bar{y}(x) = \begin{bmatrix} y_1(x) \\ y_2(x) \\ \cdots \\ y_n(x) \end{bmatrix}.$$

Тогда можно рассматривать также вектор-столбец производной

$$\bar{y}'(x) = \begin{bmatrix} y_1'(x) \\ y_2'(x) \\ \cdots \\ y_n'(x) \end{bmatrix}$$

и вектор-столбец функций правой части системы (7.1)

$$\bar{f}(x, \bar{y}) = \begin{bmatrix} f_1(x, \bar{y}) \\ f_2(x, \bar{y}) \\ \cdots \\ f_n(x, \bar{y}) \end{bmatrix}.$$

С использованием этих векторных обозначений система дифференциальных уравнений (7.1) запишется в виде

$$\bar{y}'(x) = \bar{f}(x, \bar{y}), \tag{7.3}$$

а начальные условия (7.2) – в виде

$$\bar{y}(x_0) = \bar{y}_0, \tag{7.4}$$

где

$$\bar{y}_0 = \begin{bmatrix} y_{1,0} \\ y_{2,0} \\ \dots \\ y_{n,0} \end{bmatrix}.$$

Мы видим, что векторная запись (7.3), (7.4) системы дифференциальных уравнений первого порядка (7.1), (7.2) имеет тот же вид, что и дифференциальное уравнение 1-го порядка (6.1), (6.2). Это внешнее сходство позволяет предположить, что методы решения дифференциального уравнения 1-го порядка, (см. лабораторную работу №6), можно распространить (обобщить) и на систему дифференциальных уравнений 1-го порядка вида (7.1), (7.2). Это предположение оказывается справедливым.

7.2.2. Приведение дифференциального уравнения n-го порядка к системе дифференциальных уравнений 1-го порядка

Пусть требуется найти решение дифференциального уравнения n-го порядка

$$\Phi(x, u, u', u'', \dots, u^{(n)}) = 0, \tag{7.5}$$

удовлетворяющее начальным условиям

$$u(x_0) = u_0,\ u'(x_0) = u_0',\ \dots,\ u^{(n)}(x_0) = u_0^{(n)}, \tag{7.6}$$

где $u_0, u_0', \dots, u_0^{(n)}$ – некоторые числа. Если уравнение (7.5) можно разрешить относительно старшей производной $u^{(n)}(x)$, то его можно представить в виде системы n дифференциальных уравнений 1-го порядка. Покажем, как это сделать. Пусть уравнение (7.5) представлено в виде

$$u^{(n)}(x) = F(x, u(x), u'(x), u''(x), \dots, u^{(n-1)}(x)). \tag{7.7}$$

Для функции $u(x)$ и ее производных до $(n-1)$-го порядка введем обозначения

$$y_1(x) = u(x),$$

$$y_2(x) = u'(x),$$
$$y_3(x) = u''(x),$$
$$\ldots\ldots\ldots\ldots$$
$$y_n(x) = u^{(n-1)}(x).$$

Дифференцирование этих равенств с учетом выражения (7.7) дает нам следующую систему дифференциальных уравнений первого порядка

$$\begin{aligned} y_1'(x) &= y_2(x), \\ y_2'(x) &= y_3(x), \\ y_3'(x) &= y_4(x), \\ &\ldots\ldots\ldots\ldots \\ y_n'(x) &= F(x, y_1(x), y_2(x), y_3(x), \ldots, y_n(x)). \end{aligned} \tag{7.8}$$

Начальные условия (7.6) приобретают теперь следующий вид:

$$\begin{aligned} y_1(x_0) &= u_0 = y_{1,0}, \\ y_2(x_0) &= u_0' = y_{2,0}, \\ &\ldots\ldots\ldots\ldots \\ y_n(x_0) &= u_0^{(n)} = y_{n,0}. \end{aligned} \tag{7.9}$$

7.2.3. Метод Эйлера

Этот метод решения векторного дифференциального уравнения (7.3) состоит в последовательных расчетах по формуле

$$\bar{y}_{m+1} = \bar{y}_m + h \cdot \bar{f}(x_m, \bar{y}_m), \tag{7.10}$$

начиная с точки $(x_0, \bar{y}_0)$, заданной начальными условиями x_0, $\bar{y}(x_0) = \bar{y}_0$. Здесь h – шаг интегрирования по независимой переменной x.

Для системы из двух уравнений векторная формула (7.10) представляется в виде двух следующих скалярных формул:

$$y_{1,m+1} = y_{1,m} + h \cdot f_1(x_m, y_{1,m}, y_{2,m}),$$

$$y_{2,m+1} = y_{2,m} + h \cdot f_2(x_m, y_{1,m}, y_{2,m}).$$

7.2.4. Метод Рунге–Кутта 2-го порядка

Этот метод состоит в последовательных расчетах по формулам

$$\begin{aligned} \bar{k}_1 &= \bar{f}(x_m, \bar{y}_m), \\ \bar{k}_2 &= \bar{f}(x_m + h, \bar{y}_m + h\bar{k}_1), \\ \bar{y}_{m+1} &= \bar{y}_m + \frac{h}{2}(\bar{k}_1 + \bar{k}_2), \end{aligned} \qquad (7.11)$$

начиная с точки $(x_0, \bar{y}_0)$. Необходимо заметить, что здесь $\bar{k}_1$ и $\bar{k}_2$ – векторы.

Для системы из двух уравнений векторные формулы (7.11) представляются в виде следующих скалярных формул:

$$\begin{aligned} k_{1,1} &= f_1(x_m, y_{1,m}, y_{2,m}), \\ k_{1,2} &= f_2(x_m, y_{1,m}, y_{2,m}), \\ k_{2,1} &= f_1(x_m + h, y_{1,m} + hk_{1,1}, y_{2,m} + hk_{1,2}), \\ k_{2,2} &= f_2(x_m + h, y_{1,m} + hk_{1,1}, y_{2,m} + hk_{1,2}), \\ y_{1,m+1} &= y_{1,m} + \frac{h}{2}(k_{1,1} + k_{2,1}), \\ y_{2,m+1} &= y_{2,m} + \frac{h}{2}(k_{1,2} + k_{2,2}). \end{aligned}$$

7.2.5. Метод Рунге–Кутта 4-го порядка

Этот метод состоит в последовательных расчетах по формулам

$$\begin{aligned} \bar{k}_1 &= \bar{f}(x_m, \bar{y}_m), \\ \bar{k}_2 &= \bar{f}(x_m + \frac{h}{2}, \bar{y}_m + \frac{h}{2}\bar{k}_1), \end{aligned}$$

$$\bar{k}_3 = \bar{f}(x_m + \frac{h}{2}, \bar{y}_m + \frac{h}{2}\bar{k}_2), \qquad (7.12)$$

$$\bar{k}_4 = \bar{f}(x_m + h, \bar{y}_m + h\bar{k}_3),$$

$$\bar{y}_{m+1} = \bar{y}_m + \frac{h}{6}(\bar{k}_1 + 2\bar{k}_2 + 2\bar{k}_3 + \bar{k}_4),$$

начиная с точки $(x_0, \bar{y}_0)$.

Для системы из двух уравнений каждая из векторных формул (7.12) представляется в виде двух скалярных формул, так что вместо (7.12) будем иметь

$$k_{1,1} = f_1(x_m, y_{1,m}, y_{2,m}),$$

$$k_{1,2} = f_2(x_m, y_{1,m}, y_{2,m}),$$

$$k_{2,1} = f_1(x_m + \frac{h}{2}, y_{1,m} + \frac{h}{2}k_{1,1}, y_{2,m} + \frac{h}{2}k_{1,2}),$$

$$k_{2,2} = f_2(x_m + \frac{h}{2}, y_{1,m} + \frac{h}{2}k_{1,1}, y_{2,m} + \frac{h}{2}k_{1,2}),$$

$$k_{3,1} = f_1(x_m + \frac{h}{2}, y_{1,m} + \frac{h}{2}k_{2,1}, y_{2,m} + \frac{h}{2}k_{2,2}),$$

$$k_{3,2} = f_2(x_m + \frac{h}{2}, y_{1,m} + \frac{h}{2}k_{2,1}, y_{2,m} + \frac{h}{2}k_{2,2}),$$

$$k_{4,1} = f_1(x_m + h, y_{1,m} + hk_{3,1}, y_{2,m} + hk_{3,2}),$$

$$k_{4,2} = f_2(x_m + h, y_{1,m} + hk_{3,1}, y_{2,m} + hk_{3,2}),$$

$$y_{1,m+1} = y_{1,m} + \frac{h}{6}(k_{1,1} + 2k_{2,1} + 2k_{3,1} + k_{4,1}),$$

$$y_{2,m+1} = y_{2,m} + \frac{h}{6}(k_{1,2} + 2k_{2,2} + 2k_{3,2} + k_{4,2}).$$

7.2.6. Средства Matlab для решения систем обыкновенных дифференциальных уравнений

Рассмотренные в работе 6 программы **ode34** и **ode45** имеют векторную реализацию, т.е. предназначены для решения систем обыкновенных дифференциальных уравнений 1-го порядка (7.1). Уточним описания этих программ с учетом их векторного использования. При этом будем использовать обобщенное имя **solver** (решатель), понимая под ним **ode34** или **ode45**.

[T,y]=solver('f', tspan, y0) интегрирует векторное уравнение вида

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

от **t0** до **tfinal** с начальными условиями **y0**. Здесь **'f'** – строка, содержащая имя m-файла-функции, реализующий правую часть уравнения – функцию $\bar{f}(t, \bar{y})$, оформленную в виде

function v=f(t,y).

Функция **f(t,y)** должна возвращать вектор-столбец **v**. **tspan** – массив из двух элементов, **tspan=[t0, tfinal]**. **T** – вектор значений аргумента t, при которых вычисляются значения функции $\bar{y}(t)$. **y** – матрица значений функции $\bar{y}(t)$, каждая строка которой содержит значения функций $y_1(t)$, $y_2(t)$, …, $y_n(t)$ в фиксированный момент времени. Для получения решения в конкретных точках **t0, t1,…, tfinal** (расположенных в порядке увеличения или уменьшения значений) можно использовать **tspan=[t0, t1, …, tfinal]**.

[T,y]=solver('f', tspan, y0, options) дает решение, подобное описанному выше, но с параметрами, определяемыми значениями аргумента **options**, созданного функцией **odeset**. Обычно используемые опции включают допустимое значение относительной погрешности **RelTol** (10^{-3} по умолчанию) и допустимое значение абсолютной погрешности **AbsTol** (10^{-6} по умолчанию).

В данном пособии использование функции **odeset** не рассматривается, и она не описывается.

[T,y]=solver('f', tspan, y0, options, p1, p2,…) дает решение, подобное описанному выше, помещая дополнительные параметры **p1, p2,…** в m-файл **f** всякий раз, когда он используется. При этом файл **f** должен быть оформлен в виде

function v=f(t, y, flag, p1, p2,…).

Если никакие опции не установлены, то необходимо использовать **options=[]**.

При обращении к функции **solver** без указания выходных параметров по умолчанию вызывается выходная функция **odeplot** для построения графиков полученного решения.

Пример. Пусть требуется найти решение дифференциального уравнения

$$T^2u''(t)+2\xi Tu'(t)+u(t)=0,$$

удовлетворяющее начальным условиям $u(0)=5,\ u'(0)=0$. Обозначив $u(t)=y_1(t)$, $u'(t)=y_2(t)$, вместо этого уравнения 2-го порядка мы получим следующую систему из двух уравнений 1-го порядка:

$$y_1'(t)=y_2(t),$$

$$y_2'(t)=-2\frac{\xi}{T}y_2(t)-\frac{1}{T^2}y_1(t)$$

с начальными условиями $y_1(0)=5,\ y_2(0)=0$.

Для численного решения этой системы дифференциальных уравнений составим m-файл-функцию, реализующую правую часть системы:

```
function v=sys_de(t,y,flag,T,ksi)
v=[0;0];
v(1)=y(2);
v(2)=-2*ksi*y(2)/T-y(1)/T^2;
```

и m-файл-сценарий

```
clc
```

```
clear
t0=0;
tfinal=6;
y0=[5, 0];
T=0.3;
ksi=0.1;
tspan=[t0, tfinal];
%[T,y]=ode45('sys_de', tspan, y0,[],T,ksi)
ode45('sys_de', tspan, y0,[],T,ksi)
```

7.3. Порядок выполнения работы

7.3.1. Из табл. 7.1 в соответствии с номером своей бригады взять дифференциальное уравнение и представить его в виде системы дифференциальных уравнений 1-го порядка.

7.3.2. Написать m-файл-сценарий для решения данного дифференциального уравнения изложенными выше методами. Получить также решение с помощью функций **ode34** и **ode45**. Вывести графики полученных решений.

Таблица 7.1

Дифференциальные уравнения 2-го порядка для индивидуальных заданий

№ варианта	Уравнение	№ варианта	Уравнение
1	2	3	4
1	$T^2u''(t)+2\xi Tu'(t)+u(t)-k=0$ $u(0)=1, u'(0)=0$ $\xi=0{,}5,\ T=2,\ k=1$	8	$u''(x)-3u'(x)-x^2=0$ $u(0)=0,\ u'(0)=1$

2	$4x^2u''(x)+u(x)=0$ $u(1)=1,\ u'(1)=0{,}5$	9	$T^2u''(t)+2\xi Tu'(t)+u(t)-k=0$ $u(0)=1,\ u'(0)=0$ $\xi=-1{,}5,\ T=2,\ k=1$
3	$u''(x)-xu'(x)+x^2u(x)+3x=0$ $u(0)=0,\ u'(0)=0$	10	$u''(x)-\sqrt{1+(u'(x))^2}=0$ $u(0)=1,\ u'(0)=0$
4	$\frac{1}{\omega^2}u''(x)+u(x)-k=0$ $u(0)=0,\ u'(0)=0,\ \omega=2,\ k=5$	11	$T^2u''(t)+2\xi Tu'(t)+u(t)-k=0$ $u(0)=1,\ u'(0)=0$ $\xi=1{,}5,\ T=2,\ k=1$
5	$u''(x)-2u'(x)+u(x)-x-1=0$ $u(0)=2,\ u'(0)=-3$	12	$u''(x)-4u'(x)+13u(x)=0$ $u(0)=1,\ u'(0)=0$
6	$T^2u''(t)+2\xi Tu'(t)+u(t)-k=0$ $u(0)=1,\ u'(0)=0$ $\xi=0,\ T=2,\ k=1$	13	$T^2u''(t)+2\xi Tu'(t)+u(t)-k=0$ $u(0)=0,\ u'(0)=0$ $\xi=-0{,}5,\ T=2,\ k=1$
7	$u''(x)-\omega^2u(x)=0$ $u(0)=1,\ u'(0)=0,\ \omega=2$	14	$u''(x)-4u'(x)+3u(x)-x+1=0$ $u(0)=0,\ u'(0)=1$

ЛАБОРАТОРНАЯ РАБОТА № 8. Выполнение символьных операций

8.1. Цель работы

8.1.1. Приобретение навыков выполнения символьных вычислений в среде Matlab.

8.2. Теоретические сведения

8.2.1. Понятие символьных операций

Символьными (или аналитическими) операциями называются такие операции, когда задания на вычисления задаются в виде символьных (формульных) выражений и результаты вычислений также получаются в символьном виде. В настоящее время имеется возможность выполнять символьные операции на компьютере. Для этого разработаны различные программные системы, такие как Reduce, Maple, Mathematica. Эти системы способны преобразовывать алгебраические выражения, находить аналитические решения систем линейных, нелинейных и дифференциальных уравнений, манипулировать полиномами, вычислять производные и интегралы, анализировать функции и находить их пределы и т.д. К символьным вычислениям относят также численные расчеты с произвольным числом цифр результатов и с отсутствующей погрешностью, поскольку это требует символьного представления чисел и особых алгоритмов выполнения операций с ними. Появление возможности выполнения символьных операций на компьютере привело к развитию нового научного направления – компьютерной математики (или компьютерной алгебры).

8.2.2. Выполнение символьных операций Matlab

В систему Matlab 5.2.1 входит обновленная версия пакета расширения Symbolic Math Toolbox (Symbolic), которая базируется на ядре символьной математической системы Maple V R4, лидирующей а области автоматизации аналитических решений. Последняя реализация системы символьной математики Maple V R5 в своем ядре и в расширениях имеет около 2700 функций. Система Matlab с пакетом Symbolic, включающим в себя чуть больше сотни символьных команд и функций, намного уступает Maple V по количеству таких команд и функций. В данный пакет включены лишь наиболее важные и широко распространенные функции, так что возможности выполнения символьных операций в системе Matlab остаются весьма широкими. Помимо типовых аналитических вычислений (дифференцирование и интегрирование, упрощение математических выражений, подстановка и т.д.) пакет Symbolic позволяет реализовать арифметические операции с произвольной точностью.

8.2.3. Создание символьных переменных

Поскольку переменные системы Matlab по умолчанию не определены и задаются как векторные, матричные, числовые и т.д., т.е. не имеющие отношения к символьной математике, то для реализации символьных вычислений нужно, прежде всего, позаботиться о создании специальных символьных переменных. В простейшем случае их можно определить как строковые переменные, заключив имена в апострофы. Например, при вводе в окне управления команды

$$\sin('x')\text{^}2 + \cos('x')\text{^}2$$

и нажатии клавиши Enter мы получим результат

$$ans = 1.$$

Для создания символьных переменных или объектов используется функция **sym**

x=sym('x') – возвращает символьную переменную с именем **'x'** и записывает результат в **x**.

x=sym('x','real') – возвращает символьную переменную вещественного типа с именем **'x'** и записывает результат в **x** (в общем случае символьные переменные рассматриваются как комплексные).

x=sym('x','unreal') – возвращает символьную переменную мнимого типа с именем **'x'** и записывает результат в **x**.

Возможно создание числа или матрицы в символьном виде с помощью записи вида **eps=sym('0.001')**.

8.2.4. Создание группы символьных переменных

Для создания группы символьных переменных или объектов используется функция **syms**.

syms x1 x2 … – создает группу символьных объектов, подобную выражениям **x1=sym('x1'); x2=sym('x2'); …**

syms x1 x2 … real и **syms x1 x2 … unreal** – создают группы символьных объектов с вещественными (real) и не вещественными (unreal) значениями. Последнюю функцию можно использовать для отмены задания вещественных объектов.

8.2.5. Создание списка символьных переменных

В математических выражениях могут использоваться как обычные, так и символьные переменные. Для выделения символьных переменных в некотором выражении используется функция **findsym**.

findsym(s) – возвращает в алфавитном порядке список всех символьных переменных выражения **s**. При отсутствии таковых возвращается пустая строка.

findsym(s,n) – возвращает список **n** символьных переменных, ближайших к 'x' в алфавитном порядке в выражении **s.**

Пример. В результате выполнения m-файла сценария

```
syms alpha x1 y b
a=1;
findsym(alpha+a+b)
findsym(cos(alpha)*b*x1 + 14*y,2)
findsym(y*(4+3*i) + 6*j)
```

будут выведены следующие результаты:

```
ans =
alpha, b
ans =
x1,y
ans =
y
```

Функция **findsym** позволяет упростить запись многих функций, поскольку она автоматически находит используемую в этих функциях символьную переменную.

8.2.6. Вывод символьного выражения

Система Matlab пока не способна выводить выражения и результаты их преобразований в естественной математической форме с использованием общепринятых спецзнаков для отображения интегралов, сумм, произведений и т.д. Тем не менее, некоторые возможности близкого к математическому виду вывода обеспечивает функция **pretty**.

pretty(s) – дает вывод выражения **s** в формате, приближенном к математическому.

pretty(s,n) – аналогична предшествующей функции, но задает вывод выражения **s** в **n** позициях строки (по умолчанию n=79).

Функция **latex(s)** возвращает выражение **s** в форме текстового редактора LaTeX. Это позволяет использовать это выражение в LaTeX для получения выражения в его обычной математической форме.

Пример. В результате выполнения m-файла сценария

```
syms x y
pretty(x^2/y^2)
z=latex(x^2/y^2)
```

будут выведены следующие результаты:

```
   2
  x
 ----
   2
  y


z =
{\frac {{x}^{2}}{{y}^{2}}}
```

8.2.7. Упрощение выражений

Функция **z = simplify(s)** поэлементно упрощает символьные выражения массива **s**. Если упрощение невозможно, то возвращается исходное выражение.

Пример. Программа

```
syms x
z=simplify(sin(x)^2+cos(x)^2)
```

возвращает результат z = 1.

Команды **simplify** в Symbolic не обладает в полной мере возможностями системы Maple V по упрощению выражений. Дополнительные возможности

обеспечивает функция **simple(s)**, которая выполняет различные упрощения для элементов массива *s* и выводит как промежуточные результаты, так и самый короткий конечный результат. При обращении к функции **simple** в форме

[R,HOW]=simple(s)

промежуточные результаты не выводятся. Конечные результаты упрощений содержатся в векторе **R**, а в строковом векторе **HOW** указывается выполняемое преобразование.

Пример. Программа

```
syms x
[R1,HOW1]=simple(cos(x)^2-sin(x)^2)
[R2,HOW2]=simple(2*cos(x)^2-sin(x)^2)
```

возвращает следующие результаты:

```
 R1 =
 cos(2*x)
 HOW1 =
combine(trig)
R2 =
3*cos(x)^2-1
HOW2=
simplify
```

8.2.8. Вычисление производных

Для вычисления в символьном виде производных от выражения **s** служит функция **diff**.

diff(s) – возвращает символьное значение первой производной от символьного выражения или массива символьных выражений **s** по независимой переменной, определенной функцией **findsym**.

diff(s,n) – возвращает символьное значение **n**-й производной от символьного выражения или массива символьных выражений **s** по независимой переменной, определенной функцией **findsym**.

diff(s,'v') или **diff(s,sym('v'))** – возвращает символьное значение первой производной от символьного выражения или массива символьных выражений **s** по переменной **v**.

diff(s,'v',n) или **diff(s,n,'v')** – возвращает символьное значение **n**-й производной от символьного выражения или массива символьных выражений **s** по переменной **v**.

Пример. Программа

```
syms x t
y1=diff(sin(x^2))
y2=diff(t^6,6)
```

возвращает следующие результаты:

```
y1 =
2*cos(x^2)*x
y2 =
720
```

8.2.9. Вычисление интегралов

Для вычисления определенных и неопределенных интегралов в символьном виде служит функция **int**.

int(s) – возвращает символьное значение неопределенного интеграла от символьного выражения или массива символьных выражений **s** по переменной, которая автоматически определяется функцией **findsum**. Если **s** – константа, то вычисляется интеграл по переменной **'x'**.

int(s,a,b) – возвращает символьное значение определенного интеграла на отрезке интегрирования **[a,b]** от символьного выражения или массива символь-

ных выражений **s** по переменной, которая автоматически определяется функцией **findsum**. Пределы интегрирования **a, b** могут быть как символьными, так и числовыми.

int(s,v) – возвращает символьное значение неопределенного интеграла от символьного выражения или массива символьных выражений **s** по переменной **v**.

int(s,v,a,b) – возвращает символьное значение определенного интеграла от символьного выражения или массива символьных выражений **s** по переменной **v** с пределами интегрирования **[a,b]**.

Пример. Программа

```
syms x alpha u x1
y1=int(1/(1+x^2))
y2=int(sin(alpha*u),alpha)
y3=int(x1*log(1+x1),0,1)
```

возвращает следующие результаты:

```
y1 =
atan(x)
y2 =
-cos(alpha*u)/u
y3 =
1/4
```

8.2.10. Вычисление сумм рядов

Для аналитического вычисления суммы

$$s = \sum_{i=a}^{b} f(i),$$

где i – переменная суммирования, служит функция **symsum**. Параметр b может быть конечным или бесконечном (inf).

symsum(s) – возвращает символьное значение суммы бесконечного ряда по переменной суммирования, найденной автоматически с помощью функции **findsum**.

symsum(s,a,b) – возвращает символьное значение суммы ряда по переменной суммирования, найденной автоматически с помощью функции **findsum**, при изменении этой переменной от **a** до **b**.

symsum(s,v) – возвращает символьное значение суммы бесконечного ряда по переменной суммирования **v**.

symsum(s,v,a,b) – возвращает символьное значение суммы бесконечного ряда по переменной суммирования **v** при изменении этой переменной от **a** до **b**.

Пример. Программа

```
syms k n
y1=simple(symsum(k))
y2=simple(symsum(k,0,n-1))
y3=simple(symsum(k,0,n))
y4=simple(symsum(k^2,0,n))
```

возвращает следующие результаты:

```
y2 =
1/2*n*(n-1)
y3 =
1/2*n*(n+1)
y4 =
1/6*n*(n+1)*(2*n+1)
```

8.2.11. Вычисление пределов

Для вычисления предела функции $f(x)$ служит функция **limit**.

limit(f,a) – возвращает предел в точке **a** символьного выражения **f** по независимой переменной, найденной с помощью функции **findsum**.

limit(f) – возвращает предел в точке **a=0** символьного выражения **f** по независимой переменной, найденной с помощью функции **findsum**.

limit(f,x,a) – возвращает предел символьного выражения **f** в точке **x=a**.

limit(f,x,a,'right') – возвращает предел символьного выражения **f** в точке **x=a+0** (справа).

limit(f,x,a,'left') – возвращает предел символьного выражения **f** в точке **x=a-0** (слева).

Пример. В результате выполнения программы

```
syms x t a
y1=limit(sin(x)/x)
y2=limit((x-2)/(x^2-4),2)
y3=limit((1+2*t/x)^(3*x),x,inf)
y4=limit(1/x,x,0,'right')
v=[(1 + a/x)^x, exp(-x)];
y5=limit(v,x,inf,'left')
```

будет получен следующий результат:

```
y1 =
1
y2 =
1/4
y3 =
exp(6*t)
y4 =
inf
y5 =
[ exp(a),      0]
```

8.2.12. Разложение функции в ряд Тейлора

Ряд Тейлора для функции $f(x)$ имеет вид

$$f(x)=\sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}(x-a)^n.$$

При $a=0$ этот ряд называется рядом Маклорена. Для получения разложений аналитических функций в ряд Тейлора служит функция **taylor**.

taylor(f) – возвращает 6 членов ряда Маклорена.

taylor(f,n) – возвращает члены ряда Маклорена до (n-1)-го порядка.

taylor(f,a) – возвращает 6 членов ряда Тейлора в окрестности точки **a.**

taylor(f,a,n) – возвращает члены ряда Тейлора до (n-1)-го порядка в окрестности точки **a.**

Пример. В результате выполнения программы

```
syms x t
y1=taylor(exp(-x))
y2=taylor(log(x),6,1)
y3=taylor(sin(x),pi/2,6)
y4=taylor(x^t,3,t)
```

будет получен следующий результат:

```
y1 =
1-x+1/2*x^2-1/6*x^3+1/24*x^4-1/120*x^5
y2 =
x-1-1/2*(x-1)^2+1/3*(x-1)^3-1/4*(x-1)^4+1/5*(x-1)^5
y3 =
1-1/2*(x-1/2*pi)^2+1/24*(x-1/2*pi)^4
y4 =
1+log(x)*t+1/2*log(x)^2*t^2
```

8.2.13. Вычисление определителя матрицы, обращение матрицы

Функция **det(X)** вычисляет определитель (детерминант) квадратной матрицы X в символьном виде.

Для обращения (инвертирования) матрицы в символьном виде используется функция **inv(X)**.

Пример. В результате выполнения программы

```
syms a b c d t
A=[a b;c d];
y1=det(A)
B=[1/(2-t), 1/(3-t)
   1/(3-t), 1/(4-t)];
y2=inv(B)
```

будет получен следующий результат:

```
y1 =
a*d-b*c
y2 =
[    -(-3+t)^2*(-2+t), (-3+t)*(-2+t)*(-4+t)]
[ (-3+t)*(-2+t)*(-4+t),    -(-3+t)^2*(-4+t)]
```

8.3. Порядок выполнения работы

8.3.1. Вычислить производную функции, выбранной из таб. 1.1 работы № 1 в соответствии с номером своей бригады и кодом подгруппы.

8.3.2. Вычислить определенный интеграл от функции, выбранной из табл.1.1 работы № 1 в соответствии с номером своей бригады и кодом подгруппы. Пределы интегрирования a, b выбрать самостоятельно.

8.3.3. Найти сумму ряда, предел функции и разложение функции в ряд Тейлора, указанные в табл. 8.1. Номера заданий выбрать равными номеру бригады и коду подгруппы (а или б).

Таблица 8.1.

Индивидуальные задания

Найти сумму ряда	Найти предел функции	Найти разложение функции в ряд Тейлора
1	2	3
1а, 15б: $\sum_{k=1}^{n} k$	1а, 8б: $\lim_{x\to 2} \frac{x+4}{3x+1}$	1а, 9б: e^x
2а, 12б: $\sum_{k=1}^{n} k^2$	2а, 14б: $\lim_{x\to 3} \frac{\sqrt{6+x}-3}{x-3}$	2а, 13б: a^x
3а, 14б: $\sum_{k=1}^{n} k^3$	3а, 9б: $\lim_{x\to\infty} \frac{2x^3+x^2-1}{5x^3-x}$	3а, 10б: $\sin x$
4а, 10б: $\sum_{k=1}^{n} k^4$	4а, 15б: $\lim_{x\to 0} \frac{\sqrt{1+x}-1}{x}$	4а, 7б: $\cos x$
5а, 1б: $\sum_{k=1}^{n} k^5$	5а, 13б: $\lim_{x\to 0} \frac{\ln(1+x)}{x}$	5а, 11б: shx
6а, 8б: $\sum_{k=1}^{n} (2k+1)$	6а, 10б: $\lim_{x\to 6} \frac{x-6}{\sqrt[3]{2+x}-2}$	6а, 8б: chx
7а, 13б: $\sum_{k=1}^{n} (2k+1)^2$	7а, 11б: $\lim_{x\to 0} \frac{\sin x}{x}$	7а, 12б: $\frac{1}{1-x}$
8а, 2б: $\sum_{k=1}^{n} (2k+1)^3$	8а, 2б: $\lim_{x\to 0} (1+x)^{\frac{1}{2x}}$	8а, 14б: $\ln(1-x)$
9а, 11б: $\sum_{k=1}^{n} k(k+1)$	9а, 12б: $\lim_{x\to 0} \frac{e^{2x}-1}{\sin x}$	9а, 15б: $arctgx$
10а, 3б: $\sum_{k=1}^{n} k(k+3)(k+6)$	10а, 6б: $\lim_{x\to 1} (1+2\ln x)^{\frac{1}{\ln x}}$	10а, 3б: $shx+\sin x$

11а, 4б: $\sum_{k=1}^{n} k(k+4)(k+8)$	11а, 3б: $\lim_{x\to 0} \frac{\ln(1+5x)}{x}$	11а, 2б: $shx - \sin x$
12а, 6б: $\sum_{k=1}^{\infty} \frac{1}{k(k+1)}$	12а, 1б: $\lim_{x\to 0} \frac{\sin 3x}{x}$	12а, 4б: $shx + \cos x$
13а, 9б: $\sum_{k=1}^{\infty} \frac{1}{k(k+2)}$	13а, 5б: $\lim_{x\to 0} \frac{\sin 7x}{tg5x}$	13а, 5б: $shx - \cos x$
14а, 5б: $\sum_{k=1}^{\infty} \frac{1}{k(k+3)}$	14а, 7б: $\lim_{x\to 0} (1+\sin x)^{\frac{2}{x}}$	14а, 1б: $shx \sin x$
15а, 7б: $\sum_{k=1}^{\infty} \frac{1}{(3k+1)(3k+2)}$	15а, 4б: $\lim_{x\to 0} (1-x^2)^{\frac{1}{\sin^2 x}}$	15а, 6б: $shx \cos x$

ЛИТЕРАТУРА

1. Муха В.С., Птичкин В.А. Введение в MATLAB: Метод. пособие для выполнения лаб. работ по курсам "Статистические методы обработки данных" и "Теория автоматического управления" для спец. 53 01 02 "Автоматизированные системы обработки информации". – Мн.: БГУИР, 2002. – 40 с.

2. Дьяконов В.П., Абраменкова И.В. MATLAB 5.0/5.3. Система символьной математики. – М.: Нолидж. – 1999. – 740 с.

3. Мак–Кракен Д., Дорн У. Численные методы и программирование на Фортране. – М.: Мир, 1978. – 584 с.

4. Гусак А.А. Элементы методов вычислений. – Мн.: Университетское, 1982. – 519 с.

5. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. – М.: Наука, 1988. – 700 с.

6. Дьяконов В.П. Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ. – М.: Наука, 1988. – 239 с.

7. Бахвалов Н.С. Численные методы. – М.: Наука, 1975. – 700 с.

8. Крылов В.И., Шульгина Л.Т. Справочная книга по численному интегрированию. – М.: Наука, 1966

9. Корн Г., Корн Т. Справочник по математике для научных работников и инженеров. – М.: Наука, 1973. – 832 с.

10. Дьяконов В.П. Mathematica 4 с пакетами расширений. – М.: Нолидж, 2000. – 608 с.

11. Компьютерная алгебра. Символьные и алгебраические вычисления. – М.: Мир,1986. – 391 с.

12. Грегори Р., Кришнамурти Е. Безошибочные вычисления. Методы и приложения. – М.: Мир, 1988. – 207 с.