Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Е.В. Гайдукова, Н.В. Викторова

ЧИСЛЕННЫЕ МЕТОДЫ В ГИДРОЛОГИИ

Учебное пособие

Направление подготовки 05.03.05 – Прикладная гидрометеорология
Профиль подготовки – Прикладная гидрология
Квалификация – бакалавр

Санкт-Петербург
2019

УДК [556.048:519.6](075.8)
ББК 26.222я73

Гайдукова Е.В., Викторова Н.В. Численные методы в гидрологии. Учебное пособие. – СПб.: РГГМУ, 2019. – 112 с.

ISBN 978-5-86813-487-6

Рецензент: д-р физ.-мат. наук С. А. Кондратьев (зам. директора Института озероведения РАН).

В учебном пособии рассмотрены классификация дифференциальных уравнений, принципы построения конечно-разностных схем и оценка погрешностей вычислений, понятия сходимости, корректности и устойчивости. Приведены примеры реализации численных методов в компьютерном приложении *MatLab*.

Учебное пособие предназначено студентам-гидрологам.

Gaidukova E.V., Victorova N.V. Numerical methods in hydrology. Tutorial. – St. Petersburg, RSHU Publishers, 2019. – 112 pp.

The manual discusses the classification of differential equations, the principles of constructing finite-difference schemes and the estimation of calculation errors, the concepts of convergence, correctness and stability. Examples of the implementation of numerical methods in the computer application MatLab are given.

The manual is intended for students-hydrologists.

ISBN 978-5-86813-487-6

Введение

Дисциплина «Численные методы в гидрологии» направлена на формирование у студентов навыков решения дифференциальных уравнений численными методами.

Основные задачи дисциплины заключаются в систематизированном изложении численных методов решения уравнений, описывающих процессы, протекающие в гидросфере, а также возможностей их реализации с применением программных приложений.

В результате освоения дисциплины «Численные методы в гидрологии» студент должен знать:

– классификацию обыкновенных дифференциальных уравнений;

– классификацию дифференциальных уравнений в частных производных;

– принципы построения конечно-разностных схем;

– понятия сходимости, корректности и устойчивости;

– принципы оценки погрешности вычислений;

– конечно-разностные схемы решения моделей с сосредоточенными и распределенными параметрами, представляющие собой дифференциальные уравнения.

Примеры реализации численных методов рассматриваются в компьютерном приложении *MatLab*, которому посвящен отдельный раздел учебного пособия.

Введение в численные методы.

В настоящее время численные методы реализуются на компьютере, решение задач на котором можно разбить на следующие этапы: 1) постановка задачи; 2) построение математической модели (математическая формулировка задачи); 3) разработка численного метода 4) разработка алгоритма; 5) программирование; 6) отладка программы; 7) проведение расчетов; 8) анализ результатов.

В данном пособии подробно рассмотрены этапы, связанные с выбором численного метода и разработкой программного решения для поставленной задачи, причем этапы с пятого по седьмой предлагается решать с помощью компьютерного приложения *MatLab*, в котором предусмотрены следующие средства, реализующие численные задачи: *m*-файлы, командная строка, средства пакета *Simulink*.

Численные методы подразделяют на группы: – аналитические (решение в виде формул); – графические (позволяют оценить порядок искомой величины; геометрические построения); – численные (решение задачи сводится к выполнению конечного числа арифметических действий над числами; результаты представлены в виде числовых значений). На практике, в основном, используются аналитические и численные методы. Численные методы предполагают приближенное решение задач с конкретными значениями параметров и исходных данных и отличаются относительной простотой получения результата за приемлемое время без внесения значительных погрешностей в вычислительный процесс (при правильном выборе расчетного метода).

Особое значение в численных методах имеют точность (погрешность) вычислений. Напомним основные понятия теории погрешностей.

Для решения численных задач используются числа с плавающей точкой, точность которых может достигать бесконечности. Процессор же имеет ограниченную разрядную сетку и, поэтому может оперировать лишь с некоторым конечным значением вещественного числа.

Например, 4 байта памяти – это примерно число от $-2\cdot10^9$ до $2\cdot10^9$.

Вещественное число F имеет следующий вид: $F = \pm m\cdot10^n$, где m – мантисса числа; n – порядок числа ($-273.2 \Rightarrow -2732\cdot10^{-1}$, $-2.732\cdot10^2$, $-0.2732\cdot10^3$ – нормализованная форма числа с плавающей точкой).

Применяется одинарная и двойная точность представления числа в памяти компьютера:

4 байта	8 байтов
$1.2\cdot10^{-38} - 3.4\cdot10^{38}$	$2.2\cdot10^{-308} - 1.8\cdot10^{308}$

Кстати, в *MatLab* используется по умолчанию формат числа *double* – точность порядка $10^{\pm308}$.

Мерой точности вычислений является погрешность. Различают абсолютные и относительные погрешности.

Абсолютная погрешность некоторого числа равна разности между его истинным значением и приближенным значением, полученным в результате вычисления или измерения.

Относительная погрешность – это отношение абсолютной погрешности к приближенному значению числа.

Если a – приближенное значение числа x, то
– абсолютная погрешность $\Delta x = x - a$
– относительная погрешность $\delta x = \Delta x / a$.

Истинное значение величины x обычно неизвестно. Есть приближенное значение a, и нужно найти его предельную погрешность Δa, являющуюся верхней оценкой модуля абсолютной погрешности, т. е. $|\Delta x| \le \Delta a$ (Δa – абсолютная погрешность приближенного числа a). Истинное значение x находится в интервале $(a - \Delta a, a + \Delta a)$, в котором Δa принимается равной половине единицы последнего разряда числа.

Например.

a	18.6	–0.0036	13	13.00
Δa	0.05	0.00005	0.5	0.005

Предельное значение относительной погрешности – $\delta a = \Delta a / |a|$, Δa – предельная абсолютная погрешность, $|a|$ – абсолютная величина приближенного числа.

Например, $\delta(-2.3) = 0.05/2.3 \approx 0.022$ (2.2 %)

Следует отметить, что погрешность округляется всегда в сторону увеличения, поэтому $\delta(-2.3) \approx 0.03$.

При расчетах погрешностей нужно соблюдать равносильность преобразований. Например:

– правильные преобразования: $5500 = 0.5500 \cdot 10^4$ и $0.110 \cdot 10^2 = 11.0$.

– неправильные преобразования: $5500 = 0.55 \cdot 10^4$ и $0.110 \cdot 10^2 = 11$.

При сложении или вычитании чисел их абсолютные погрешности складываются: $\Delta(a \pm b) = \Delta a + \Delta b$. При умножении и делении относительные погрешности также складываются: $\delta(a \cdot b) = \delta a + \delta b$, $\delta(a / b) = \delta a + \delta b$. При возведении в степень приближенного числа его относительная погрешность умножается на показатель степени: $\delta(a^k) = k\delta a$.

Рассмотрим источники погрешностей, которыми могут быть:

1) математическая модель, если в ней не учтены какие-либо важные черты рассматриваемой задачи (надо учитывать область применимости модели); 2) исходные данные; 3) численный метод; 4) погрешности округлений.

При численных расчетах первый и второй источники погрешностей относятся к неустранимым видам погрешностей. Третий источ-

ник погрешностей может появиться, например, при замене интегрирования суммой, при интерполировании табличных данных и т. д. Поэтому можно уменьшить погрешность, если, например, уменьшить шаг интегрирования до величины в несколько раз меньшей неустранимой погрешности. Избежать четвертого вида погрешностей можно при правильной организации порядка вычислений, например, сложение чисел разных знаков приводит к компенсации погрешностей.

При вычислениях необходимо соблюдать устойчивость, корректность и сходимость задачи.

1) Задача называется устойчивой по исходному параметру x, если решение y непрерывно от него зависит, т. е. малое приращение исходной величины Δx приводит к малому приращению искомой величины Δy. Другими словами: малые погрешности в исходной величине приводят к малым погрешностям в решении. Неустойчивые задачи чувствительны к погрешностям исходных данных.

2) Задача называется поставленной корректно, если для любых значений исходных данных из некоторого класса ее решение существует, единственно и устойчиво по исходным данным.

3) Сходимость – близость получаемого численного решения задачи к истинному решению.

1. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

1.1. Решение обыкновенных дифференциальных уравнений

В этой главе рассмотрим наиболее простые, с практической точки зрения, методы решения обыкновенных дифференциальных уравнений (ОДУ): метод Эйлера и метод Рунге–Кутты.

1.1.1. Метод Эйлера

По определению решением обыкновенного дифференциального уравнения первого порядка, разрешенного относительно производной,

$$\frac{dy}{dx} = f(x, y(x)) \tag{1.1}$$

называется дифференцируемая функция $y = \varphi(x)$, удовлетворяющая этому уравнению, т. е. такая, что $\varphi'(x) \equiv f(x, \varphi(x))$ тождественна на некотором участке изменения x. Задача нахождения решения уравнения (1.1), удовлетворяющего начальному условию $\varphi(x_0) = y_0$, называется задачей Коши.

В общем случае уравнение (1.1) аналитического решения, выражающегося через элементарные функции, не имеет. Кроме того, даже при наличии аналитического решения представление его в графическом виде существенно облегчает его анализ. Отмеченные причины являются весьма важным аргументом в пользу нахождения приближенных численных решений ОДУ.

При численном решении ОДУ методом Эйлера вместо исходного дифференциального уравнения ищется решение конечно-разностного ОДУ. Переход к конечно-разностному уравнению в уравнении (1.1) осуществляется следующим образом. Вместо точного значения производной рассматривает ее разностный аналог:

$$\frac{dy}{dx} \to \frac{y(x+\Delta x) - y(x)}{\Delta x}, \tag{1.2}$$

где Δx достаточно малая величина. Тогда в конечных разностях уравнение (1.1) принимает следующий вид:

$$\frac{y(x+\Delta x) - y(x)}{\Delta x} = f(x, y(x)). \tag{1.3}$$

Откуда сразу получаем формулу для нахождения значения функции $y(x)$ в точке $x+\Delta x$

$$y(x+\Delta x) = y(x) + \Delta x f(x, y(x)). \tag{1.4}$$

Из (1.4) видно, как построить алгоритм для решения задачи Коши.

1. Задать начальные условия (x_0, y_0).

2. Вычислить значение функции $y(x)$ в точке $x_1 = x_0+\Delta x$, определяемое выражением (1.4)

$$y_1 = y_0 + \Delta x f(x_0, y_0). \tag{1.5}$$

3. Повторить процедуру, описанную в п. 2, и найти значение функции $y(x)$ в точке $x_2 = x_1+\Delta x$ и т. д. для всех значений переменной $x_n = x_i + i\Delta x$.

Наиболее удобно записать данный алгоритм в виде итерационной формулы:

$$y_i = y_{i-1} + \Delta x f(x_{i-1}, y(x_{i-1})), \quad i = 1, 2, ... \tag{1.6}$$

Из (1.3) видно, что метод будет давать хорошее приближение к «истинному» значению функции $y(x)$, если приращение аргумента Δx достаточно мало. Величина шага Δx зависит от конкретной задачи. Значение в n-й точке оценивается по значению функции и ее производной в $n-1$-й точке, при этом принимается, что значение производной (т. е. угол наклона касательной при геометрической интерпретации) на отрезке $[x_{i-1}; x_i]$ остается неизменным. Не выполнение

данного условия приводит к отклонению численного решения от точного, которое может быть уменьшено уменьшением Δx.

Приступая к разработке программы вне зависимости от использованного языка программирования, необходимо разбить всю задачу на последовательность независимых заданий, соответствующих алгоритму, описанному выше. Программа должна состоять из следующих блоков:

1. Задание начальных условий.
2. Задание функции $f(x, y(x))$.
3. Задание отрезка, на котором ищется решение, и шага интегрирования. На практике более удобно задавать не шаг интегрирования, а количество интервалов, на которые разбивается отрезок интегрирования, а затем вычислять значение шага.
4. Вычисление координат точек, в которых ищется решение дифференциального уравнения.
5. Решение уравнения (1.6) методом Эйлера.
6. Вывод результатов.

Несколько слов о точности решения, получаемого методом Эйлера. Так как дифференциальное уравнение заменяется разностным аналогом, то естественно можно ожидать некоторое отличие численного решения от «истинного» решения ОДУ. Далее показана оценка погрешности метода Эйлера на i-м шаге интегрирования Δ_i, равная разности между точным решением уравнения $y(x_i)$ и соответствующим значением численного решения $y_i = y_{i-1} + \Delta x f(x_{i-1}, y(x_{i-1}))$:

$$\Delta_i = y(x_i) - [y_{i-1} + \Delta x f(x_{i-1}, y(x_{i-1}))]. \qquad (1.7)$$

Вспоминая, что $x_i = x_{i-1} + \Delta x$, и раскладывая $y(x_{i-1} + h)$ в ряд Тейлора, приводим (1.7) к следующему виду

$$\Delta_i = y_{i-1} + y'(x_{i-1})\Delta x + \frac{1}{2} y^n(x_{i-1})(\Delta x)^2 - \ldots - [y_{i-1} + \Delta x f(x_{i-1}, y(x_{i-1}))] + O(\Delta x^3). \qquad (1.8)$$

Но в соответствии с уравнением (1.2) $y'(x_{i-1}) = f(x_{i-1}, y(x_{i-1}))$, поэтому

$$\Delta_i = \frac{1}{2} y^n(x_i)(\Delta x)^2 + \mathrm{O}(\Delta x^3). \qquad (1.9)$$

Из (1.9) видно, что главный член погрешности интегрирования ОДУ первого порядка по методу Эйлера на одном шаге пропорционален $(\Delta x)^2$. После N шагов погрешность составит N $(\Delta x)^2$. Так как при заданном интервале интегрирования $N \sim \frac{1}{\Delta x}$, полная погрешность численного решения $\Delta \sim \Delta x$, поэтому говорят, что метод Эйлера является методом первого порядка точности.

В общем случае отклонение численного решения от точного обусловлено двумя причинами. Во-первых, компьютеры не оперируют с вещественными числами (например, десятичными числами с дробной частью) бесконечной точности, но представляют вещественные числа в виде конечного числа десятичных цифр, определяемого аппаратными средствами компьютера и его программным обеспечением. Это приводит к тому, что арифметические операции, выполняемые с действительными числами, будут выполняться с некоторой погрешностью, называемой погрешностью округления. Погрешности округлений по мере роста объема вычислений имеют свойство накапливаться.

Второй причиной отклонения численного решения от точного является вычислительный алгоритм, применяемый в конкретной задаче. Указанное обстоятельство определяет необходимость проводить специальные исследования применяемых вычислительных алгоритмов и оценивать их точностные характеристики. Но не существует правил для выбора «наилучшего» метода решения ОДУ. У каждого метода имеются свои достоинства и недостатки, а конкретный выбор определяется требованиями и квалификацией исследователя, а также характером конкретного решения, который заранее неизвестен.

На практике точность численного решения определяют, уменьшая шаг интегрирования ОДУ до тех пор, пока численное решение не перестанет зависеть от шага при заданном уровне точности. Выбирая величину шага, важно помнить, что выбор слишком малого шага приводит к увеличению объема вычислений и, соответственно, погрешности округлений.

Другой не менее важной характеристикой алгоритма является его устойчивость. В ряде задач возникают ситуации, когда численные результаты находятся в хорошем соответствии с «истинным решением» на коротких интервалах, а на больших интервалах отклоняются от него. Это обусловлено тем, что малые погрешности алгоритма, многократно перемножаясь, приводят к геометрическому росту погрешности. О таком алгоритме применительно к данной задаче говорят как о неустойчивом, использование которого приведет к неверным численным результатам. Поэтому при решении конкретной задачи проводят специальные исследования, позволяющие оценить точность и устойчивость выбранного вычислительного алгоритма.

1.1.2. Метод Рунге–Кутты

Не следует отдавать предпочтение одному какому-нибудь алгоритму, однако для конкретных целей достаточно ограничиться одним вычислительным алгоритмом, который позволяет получать устойчивые решения, отвечающие требованиям по точности по сравнению с методом Эйлера. Указанными свойствами обладает метод Рунге–Кутта 4-го порядка. Данный метод реализуется следующей итерационной формулой:

$$y_{n+1} = y_n + \frac{\Delta x}{6}(k_1(y_n) + 2k_2(y_n) + 2k_3(y_n) + k_4(y_n)), \qquad (1.10)$$

где k_1, k_2, k_3, k_4 – поправки, вычисляемые по формулам

$$k_1 = f(x_n,\ y_n),$$

$$k_2 = f(x_n + \Delta x/2,\ y_n + \Delta x k_1/2),$$

$$k_3 = f(x_n + \Delta x/2,\ y_n + \Delta x k_2/2),$$

$$k_4 = f(x_n + \Delta x,\ y_n + \Delta y).$$

О точности метода Рунге–Кутты. Предположим, что $y(t)$ – решение задачи Коши. Если $y(t) \in C^5[t_0, b]$ и $\{(t_k, y_k)\}_{k=0}^{M}$ – последовательность приближений, сгенерированная методом Рунге–Кутты порядка 4, то:

$$| e_k | = | y(t_k) - y_k | = O(h^4), \tag{1.11}$$

$$| e_{k+1} | = | y(t_{k+1}) - y_k - hT_N(t_k, y_k) | = O(h^5). \tag{1.12}$$

В частности, окончательная общая ошибка в конце интервала будет равна

$$E(y(b), h) = | y(b) - y_M | = O(h^4). \tag{1.13}$$

Если приближения вычислены с шагами h и $h/2$, то получаем

$$E(y(b), h) \approx Ch^4 \tag{1.14}$$

для шага большей длины и

$$E(y(b), h/2) \approx C\frac{h^4}{16} = \frac{1}{16}Ch^4 \approx \frac{1}{16}E(y(b), h). \tag{1.15}$$

Если длину шага в методе Рунге–Кутты порядка $N = 4$ уменьшить в 2 раза, то можно ожидать, что полная окончательная ошибка уменьшится в 16 раз.

1.2. Методы решения дифференциальных уравнений в частных производных

Методы решения зависят от вида дифференциального уравнения в частных производных (ДУЧП): линейный или нелинейный.

К методам решения линейных ДУЧП относятся метод разделения переменных (метод Фурье), метод интегральных преобразований,

метод функций Грина, метод интегральных уравнений, метод разложения по собственным функциям, метод теории функций комплексного переменного.

Методы, пригодные для решения как линейных, так и нелинейных ДУЧП, – вариационные методы, метод преобразования координат, метод преобразования переменных, метод теории возмущений (метод малого параметра), метод моментов (метод Галеркина), метод автомодельных решений, численные методы.

В данном пособии рассматриваются именно численные методы, по которым исходное дифференциальное уравнение сводится к системе алгебраических уравнений, которое можно решить с помощью компьютерного приложения, это дает возможность получения более точного решения ДУЧП. К этой группе методов относятся метод конечных разностей, метод конечных элементов, метод Монте-Карло и др.

1.2.1. Классификация дифференциальных уравнений второго порядка

В настоящей главе дана классификация дифференциальных уравнений в частных производных второго порядка и представлены некоторые важные уравнения. Информация о типе уравнения необходима, в первую очередь, для выбора методов решения, соответствующих данному типу уравнения.

Уравнение называют линейным, если оно имеет вид

$$a_{11}\frac{\partial^2 u}{\partial x^2} + 2a_{12}\frac{\partial^2 u}{\partial x \partial y} + a_{22}\frac{\partial^2 u}{\partial y^2} + b_1\frac{\partial u}{\partial x} + b_2\frac{\partial u}{\partial y} + cu = g\,, \quad (1.16)$$

где $a_{11}, a_{12}, a_{22}, b_1, b_2, c, g$ – функции x и y, но не зависят от u. Если коэффициенты уравнения (1.16) не зависят от x и y, то оно представляет собой уравнение с постоянными коэффициентами. Уравнение называется однородным, если $g(x, y) = 0$.

В виде (1.16) могут быть записаны три типа уравнений – так называемые, гиперболические, параболические и эллиптические. Такая классификация построена по аналогии с классификацией кривых

второго порядка в аналитической геометрии. Тип уравнения в частных производных определяется знаком определителя, составленного из коэффициентов уравнения.

Составим, так называемое, уравнение характеристик:

$$ady^2 - 2bdxdy + cdx^2 = 0,$$

перепишем его в виде

$$a(dy/dx)^2 - 2b(dy/dx) + c = 0.$$

Полученное уравнение – квадратное относительно величины *dy*/*dx*, выражающей тангенс угла наклона характеристических линий. Решая его, найдем два корня

$$\left(\frac{dy}{dx}\right)_{1,2} = \frac{b \pm \sqrt{b^2 - ac}}{a},$$

откуда находим уравнения, описывающие два семейства характеристик

$$ady - \left(b + \sqrt{b^2 - ac}\right)dx = 0;$$

$$ady - \left(b - \sqrt{b^2 - ac}\right)dx = 0.$$

Знак подкоренного выражения (дискриминанта) и определяет тип уравнения.

Уравнение называют:

1. *гиперболическим* в точке M, если в этой точке $D = b^2 - ac > 0$ (существует две различные действительные характеристики);

2. *эллиптическим* в точке M, если в этой точке $D = b^2 - ac < 0$ (действительных характеристик нет);

3. *параболическим* в точке М, если в этой точке $D = b^2 - ac = 0$ (характеристики действительные, но совпадают).

Отметим, что одно и то же уравнение может иметь разный тип в разных точках вычислительной области, если коэффициенты уравнения зависят от координат, например, уравнение Трикоми $y\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} = 0$, которое описывает трансзвуковые (т. е. течения до– и сверхзвуковые) течения невязкого газа.

1.2.2. Примеры уравнений в частных производных

Приведем уравнения, которые часто встречаются при рассмотрении различных физических процессов, а также используются для анализа свойств разностных схем, применяемых при решении более сложных уравнений.

1. Уравнение Лапласа – эллиптического типа:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0 . \qquad (1.17)$$

Оператор $\nabla^2 = \nabla \cdot \nabla = \operatorname{div}\operatorname{grad}$, так называемый, лапласиан (иногда используют символ Δ), обозначает дифференциальную операцию, выполняемую над функцией *u*.

Уравнение Лапласа описывает, например, стационарное поле температур в среде с постоянным коэффициентом теплопроводности. В двумерном случае уравнение (1.17) имеет вид:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 . \qquad (1.18)$$

2. Уравнение Пуассона

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = g(x, y, z) \qquad (1.19)$$

описывает распределение температуры в твердом теле, когда внутри него есть источники тепла интенсивности $g(x, y)$. Тип – эллиптический. В двумерном случае уравнение (1.19) имеет вид:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = g(x, y). \qquad (1.20)$$

3. Уравнение теплопроводности или уравнение диффузии (описывает нестационарный процесс распространения теплоты или вещества) – параболического типа:

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) = \alpha\nabla^2 u, \qquad (1.21)$$

где α – коэффициент температуропроводности или диффузии соответственно. В одномерном случае уравнение (1.21) записывается в виде

$$\frac{\partial u}{\partial t} = \alpha\frac{\partial^2 u}{\partial x^2}. \qquad (1.22)$$

4. Уравнение конвекции и диффузии

$$\frac{\partial \xi}{\partial t} + u\frac{\partial \xi}{\partial x} = \alpha\frac{\partial^2 \xi}{\partial x^2} \qquad (1.23)$$

описывает перенос скалярной величины ξ (например, концентрации, температуры) при скорости конвекции u; α – коэффициент диффузии, температуропроводности. Тип уравнения – параболический.

5. Волновое уравнение (описывает процесс распространения возмущений в струне со скоростью c) – гиперболического типа:

$$\frac{\partial^2 u}{\partial t^2} = c^2\frac{\partial^2 u}{\partial x^2}. \qquad (1.24)$$

6. Линейное волновое уравнение первого порядка (описывает волну, бегущую вправо со скоростью c):

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 . \qquad (1.25)$$

7. Уравнение Трикоми описывает трансзвуковые течения невязкого газа:

$$y\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 . \qquad (1.26)$$

Важное его свойство – изменение типа с эллиптического на гиперболический в зависимости от знака y.

8. Уравнение Гельмгольца

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + k^2 u = 0 \qquad (1.27)$$

описывает установившийся колебательный процесс с волновым числом k – эллиптического типа.

9. Невязкое уравнение Бюргерса (нелинейное волновое уравнение, или уравнение переноса) – описывает процесс распространения нелинейных волн в одномерном случае

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 . \qquad (1.28)$$

10. Уравнение Бюргерса с диффузионным членом

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \qquad (1.29)$$

похоже на уравнения газовой динамики и используется как модель для анализа их решения. Здесь ν – кинематическая вязкость.

11. Уравнение Кортевега–де Вриза

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0 \qquad (1.30)$$

описывает процесс распространения нелинейных волн при наличии дисперсии (т. е. когда фазовая скорость распространения волн зависит от их частоты).

1.2.3. Построение конечно-разностных схем

Разностные (сеточные) методы чаще всего применяют на практике, в частности даже для задач, которые редко удается решить классическими методами, например линейные уравнения с коэффициентами достаточно общего вида или линейные уравнения в областях сложной формы. Эти методы обладают универсальностью и поддержаны хорошо разработанной теорией.

Основа метода конечных разностей – дискретизация, т. е. замена непрерывной области совокупностью изолированных точек (сеткой), причем решение ищется только в узлах сетки. Производные аппроксимируются конечными разностями, и решение уравнения в частных производных сводится к решению системы алгебраических уравнений, которую называют разностной схемой.

Основные особенности получающейся разностной схемы определяются типом исходного уравнения в частных производных. *Стационарные* задачи обычно сводятся к системам алгебраических уравнений, которые приходиться решать одновременно во всей расчетной области, учитывая заданные граничные условия; после сравнения решений на текущей и предыдущей итерациях вычисления повторяют до момента достижения заданной точности. *Нестационарные* (*маршевые*, или *эволюционные*) задачи сводятся к системе алгебраических уравнений, которую решают на каждом временном слое, учитывая граничные условия.

Весь процесс решения ДУЧП методом конечных разностей состоит из следующих этапов: 1) дискретизация расчетной области; 2) замена ДУЧП надежным конечно-разностным аналогом; 3) аппрокси-

мация граничных и начальных условий; 4) проверка согласованности, устойчивости, сходимости выбранной схемы, оценка ее точности; 5) выбор метода решения полученной системы алгебраических уравнений; 6) построение и отладка программы, реализующей алгоритм решения полученной системы уравнений; оптимизация программы; 7) анализ полученного решения, сравнение с известными точными решениями, проверка интегрального баланса массы, теплоты, количества вещества и т. д.; 8) вывод результатов в виде таблиц и графиков.

Первый шаг метода конечных разностей – замена непрерывной области конечно-разностной сеткой, т. е. дискретизация (рис. 2.1). Необходимо построить конечно-разностный аналог уравнения на данной сетке, используя определенный шаблон (*см*., например, шаблоны на рис. 2.2, *б*, *в*).

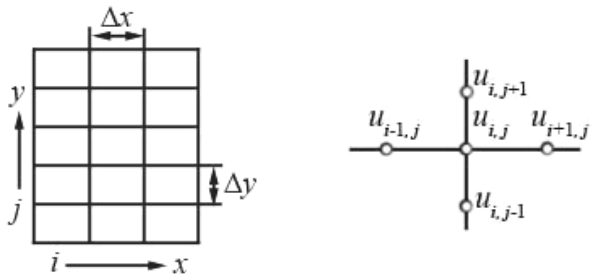


Рис. 2.1. Пример конечно-разностной сетки для прямоугольной области.

Принятые обозначения ясны из рисунка 2.1:

$$u_{i,j} = u(x_0, y_0);\ u_{i+1,j} = u(x_0 + \Delta x, y_0);\ u_{i-1,j} = u(x_0 - \Delta x, y_0);\quad (1.31)$$
$$u_{i,j+1} = u(x_0, y_0 + \Delta y);\ u_{i,j-1} = u(x_0, y_0 - \Delta y).$$

При решении нестационарных задач номер слоя принято обозначать верхним индексом (например, u_j^{n+1}).

Рассмотрим основные понятия разностных методов.

Под *разностной схемой* понимают совокупность разностных уравнений, аппроксимирующих основное дифференциальное урав-

нение и дополнительные условия исходной дифференциальной задачи в области изменения переменных $G(x, y, z, t)$.

Сеткой на отрезке $[a, b]$ называется любое конечное множество точек этого отрезка. *Равномерной сеткой* на $[a, b]$ называется множество точек

$$\omega_h = \{x_i = a + ih,\ i = 0, 1, ..., N\}, \qquad (1.32)$$

где $h = (b–a)/N$ – шаг сетки.

Понятие сетки обобщается и на большее число измерений. Например, равномерная прямоугольная сетка в прямоугольной области изменения переменных $G(x, t)$ может быть образована пересечением линий, параллельных сторонам области и отстоящих друг от друга с равным шагом (*см*. рис. 2.2, *а*).

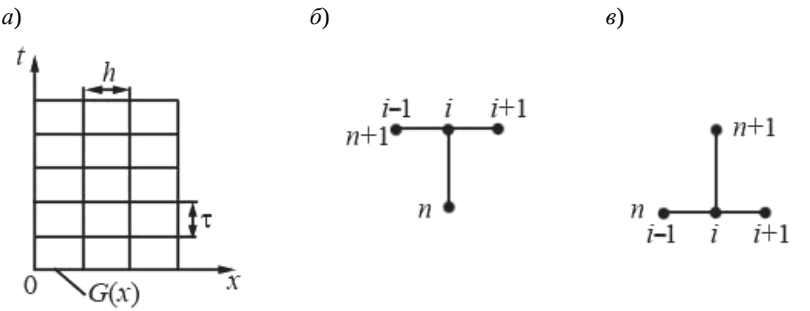


Рис. 2.2. Геометрические элементы разностных методов: *а* – конечно-разностная сетка для прямоугольной области; *б* – шаблон неявной схемы; *в* – шаблон явной схемы.

Рассмотрим задачу о приближенном вычислении производных функции $u(x)$, определенной, непрерывной и гладкой на отрезке $[a, b]$. Введем сетку ω_h и обозначим $u_i = u(x_i)$. Тогда разностные соотношения могут иметь следующий вид:

$$u_{x^-,t} = \left.\frac{\partial u}{\partial x}\right|_i = \frac{u_i - u_{i-1}}{h}, \qquad (1.33)$$

$$u_{x^+,t} = \left.\frac{\partial u}{\partial x}\right|_i = \frac{u_{i+1} - u_i}{h}, \tag{1.34}$$

$$u_{x^0,t} = \left.\frac{\partial u}{\partial x}\right|_i = \frac{u_{i+1} - u_{i-1}}{2h} \tag{1.35}$$

и называются соответственно *левой*, *правой* и *центральной разностными производными функции* $u(x)$ *в точке* $x = x_i$. Формулы (1.33) и (1.34) иногда называют *разностями назад* и *разностями вперед* соответственно.

Вторую производную можно приближенно заменить в точке x_i второй разностной производной

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_i = \frac{1}{h}(u_{x^+,t} - u_{x^-,t}) = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}. \tag{1.36}$$

Очевидно, что при предельном переходе $h \to 0$ выражения (1.33) – (1.35) будут совпадать с классическим определением первой, а выражение (1.36) – с определением второй производной функции $u(x)$ в точке $x = x_i$.

Пусть требуется составить разностные схемы для одномерной задачи теплопроводности на ограниченном отрезке

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < a, \quad t > 0$$
$$u(x,0) = \mu(x), \quad u(0,t) = \mu_1(t), \quad u(a,t) = \mu_2(t). \tag{1.37}$$

Введем в области $G = [0 \le x \le a] \times [0 \le t < \infty)$ равномерную прямоугольную сетку, образованную пересечением линий $x_i = ih$, $i = 0, \ldots, N$ и $t_n = n\tau$ $n = 0, 1, \ldots$, где h, τ – шаги сетки по переменным x, t (*см.* рис. 2.2, *а*). Величина $n_{\max}$ не задана, т. к. в нестационарных задачах время установления процесса обычно заранее не известно. Значения функции в узлах сетки будем обозначать $u_i^n = u(x_i, t_n)$. Возьмем око-

ло узла (x_i, t_n) конфигурацию узлов, изображенную на рис. 2.2, *б*. Заменим в первом из уравнений (1.37) производную u_t разностным отношением типа (1.34), т. е. выражением $(u_i^{n+1} - u_i^n)/\tau$, а производную $\partial^2 u / \partial x^2$ – отношением вида (1.36): $(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1})/h^2$. Тогда ДУЧП приближенно заменится (аппроксимируется) разностной схемой, представляющей собой систему из $N-1$ линейных алгебраических уравнений

$$\frac{u_i^{n+1} - u_i^n}{\tau} = \alpha \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{h^2}, \quad i = 1, ..., N-1. \qquad (1.38)$$

Число уравнений (1.38) меньше числа неизвестных u_i^{n+1}, $i = 0, ..., N$. Недостающие уравнения выводятся из начальных и граничных условий:

$$u_i^0 = \mu(x_i),\ i = 0, \ldots, N;\ u_0^{n+1} = \mu_1(t_{n+1}), \quad u_N^{n+1} = \mu_2(t_{n+1}). \qquad (1.39)$$

Конфигурацию узлов, используемую для составления разностной схемы, называют *шаблоном*.

Для одной и той же задачи можно составить много разностных схем. Например, если для задачи (1.37) использовать шаблон, изображенный на рис. 2.2, *в*, то вместо (1.38) получим схему вида

$$\frac{u_i^{n+1} - u_i^n}{\tau} = \alpha \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{h^2}, \quad i = 1, ..., N-1. \qquad (1.40)$$

Начальные и граничные условия для этой схемы также можно записать в форме (1.39).

В разностной схеме (1.40) содержится только одно неизвестное значение функции на новом (n + 1)-м слое, это значение легко явно выразить через известные значения функции на n-м слое. Поэтому такие схемы называют *явными*.

Схема (1.38) содержит в каждом уравнении несколько (в данном случае – три) неизвестных значения функции на (n + 1)-м слое; подобные схемы называются *неявными*.

Для большинства разностных схем узлы сетки лежат на пересечении некоторых прямых линий, проведенных либо в декартовой системе координат, либо в специально подобранной по форме области G.

Для двумерных задач в прямоугольной области G наиболее часто используют прямоугольную сетку (*см*. рис. 2.2, *а*), для трехмерных задач наиболее употребительна сетка из прямоугольных параллелепипедов.

Если одна из переменных имеет физический смысл времени t, то сетку обычно строят так, чтобы среди ее линий (плоскостей) были линии постоянного времени $t = t_n$. Совокупность узлов сетки, лежащих на такой линии (плоскости), называют *слоем*.

В многомерных задачах на каждом слое выделяют *направления* – линии, вдоль которых меняется только одна из пространственных координат.

В областях прямоугольной формы (для декартовой системы координат) часть узлов сетки естественно ложится на границу области; эти узлы называют *граничными*, а остальные узлы – *внутренними*. Начальные и граничные условия, наложенные на решение на границе $\Gamma(G)$, можно в этом случае считать заданными в граничных узлах сетки.

Если область G является кругом (кольцом), цилиндром или шаром, то часто переходят к системам координат, связанным с видом области: полярным, цилиндрическим или сферическим.

Узлы, в которых разностная схема записана на шаблоне, называют *регулярными*, а остальные узлы – *нерегулярными*. Нерегулярными являются обычно граничные узлы, а иногда также узлы, лежащие вблизи границы (такие, что взятый около этого узла шаблон выходит за границу области).

Составление разностной схемы начинается с выбора шаблона. Шаблон не всегда однозначно определяет разностную схему, но существенно влияет на ее свойства.

Введем понятие невязки. Рассмотрим операторное уравнение общего вида (оно может быть и линейное)

$$Au = f \text{ , или } Au - f = 0 . \qquad (1.41)$$

Заменяя оператор A разностным оператором A_h, правую часть f – некоторой сеточной функцией φ_h, а точное решение u – разностным решением y, запишем разностную схему

$$A_h y = \varphi_h, \text{ или } A_h y - \varphi_h = 0. \qquad (1.42)$$

Если подставить точное решение u в соотношение (1.42), то оно, вообще говоря, не будет точно удовлетворяться, т. е. $A_h u - \varphi_h \neq 0$. Величину

$$\psi = \varphi_h - A_h u \equiv (Au - f) - (A_h u - \varphi_h) \qquad (1.43)$$

называют *невязкой*.

Для каждого уравнения в частных производных существует множество его конечно-разностных аналогов, из которых обычно нельзя выбрать наилучший со всех точек зрения. В первую очередь нужно стремиться к «правильной» аппроксимации уравнений поставленной задачи, обеспечивающей сходимость, а затем следует выбрать «наилучшую» схему, т. е. оптимизировать ее, учитывая ее точность, экономичность, удобство реализации на ЭВМ и т. д.

В практической и научной гидрологии используются уравнения различных типов, но большинство относится к параболическому типу (например, уравнение Фоккера–Планка–Колмогорова), поэтому последующие разделы будут посвящены рассмотрению именно этого типа.

1.2.4. Некоторые разностные схемы для уравнения теплопроводности

В этом разделе будем рассматривать построение и характерные черты *метода конечных разностей* для уравнений в частных производных на примере неоднородного уравнения теплопроводности

$$\partial u / \partial t = a^2 \frac{\partial^2 u}{\partial x^2} + g(x,t),\ x \in [0,l],\ t \in [0,T], \qquad (1.44)$$

с заданным начальным по временной переменной t

$$u(x,0)=\varphi(x) \text{ при } x\in[0,l] \tag{1.45}$$

и граничным по пространственной переменной x

$$u(0,t)=\alpha(t),\ u(l,t)=\beta(t) \text{ при } t\in[0,T] \tag{1.46}$$

условиями.

Область Ω, на которой определена данная задача представляет собой прямоугольник $(0, l)$ $x(0, T)$ в системе координат $0xt$, а ее граница Γ состоит из отрезков прямых $x = 0$, $x = l$, $t = 0$ (отрезок прямой $t=T$ здесь к границе Γ не вносится). Разобьем этот прямоугольник на такие же прямоугольные части прямыми $x = x_i$, и $t–t_k$, параллельными осям $0t$ и $0x$ соответственно, где:

$$x_i=ih,\ h=l/n,\ i=0,1,...,n \tag{1.47}$$

$$t_k=k\tau,\ \tau=T/m,\ k=0,1,...,m \tag{1.48}$$

числа h и τ, фигурирующие в (1.47), (1.48) – шаги сетки по переменным x и t соответственно. Точки $(x_i; t_k) \in \overline{\Omega} := \Omega \cup \Gamma$ – используется сетка с постоянным шагом вдоль каждой из осей, что позволяет привлекать конечно-разностную интерполяцию (рис. 2.3).

Частные производные рассматриваются в фиксированной точке, к ним можно применить формулы численного дифференцирования функций одной переменной. А именно, для производной $\partial^2 u/\partial x^2$ наиболее естественно употребить центральную формулу, согласно которой

$$\frac{\partial^2 u}{\partial x^2}\Big|_{\substack{x=x_i\\ t=t_k}}=\frac{u(x_{i-1},t_k)-2u(x_i,t_k)+u(x_{i+1},t_k)}{h^2}+O(h^2). \tag{1.49}$$

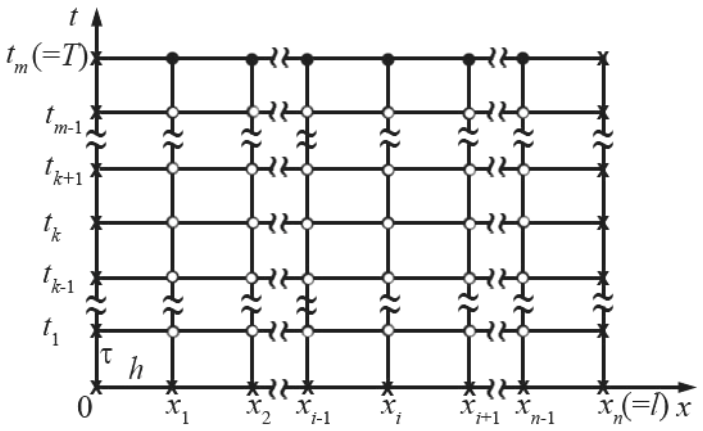


Рис. 2.3. Сетка для конечно-разностного метода решения задачи (1.44) – (1.46).

Для приближенной замены $\partial u/\partial t$ с одинаковым успехом можно привлечь формулы левой (1.33), правой (1.34) и центральной (1.35) аппроксимации первой производной. Соответственно имеем:

$$\frac{\partial u}{\partial t}\Big|_{\substack{x=x_i\\ t=t_k}} = \frac{u(x_i,t_k)-u(x_i,t_{k-1})}{\tau} + O(\tau), \qquad (1.50)$$

$$\frac{\partial u}{\partial t}\Big|_{\substack{x=x_i\\ t=t_k}} = \frac{u(x_i,t_{k+1})-u(x_i,t_k)}{\tau} + O(\tau), \qquad (1.51)$$

$$\frac{\partial u}{\partial t}\Big|_{\substack{x=x_i\\ t=t_k}} = \frac{u(x_i,t_{k+1})-u(x_i,t_{k-1})}{2\tau} + O(\tau^2). \qquad (1.52)$$

Будем считать, что $u_i^k \coloneqq u(x_i,t_k)$, т. е. чтобы не вводить другой буквы, через u_i^k обозначаем значения сеточной функции, соответствующей приближенному решению $u_h^\tau(x,t)$ на сетке Ω_h^τ, *каркаса* приближенного, а не точного решения данной задачи.

Подставив в уравнение (1.44) аппроксимации $\partial^2 u/\partial x^2$ по формуле (1.49) и $\partial u/\partial t$ по одной из формул (1.50), (1.51) или (1.52), отбро-

сив погрешности аппроксимаций, для расчетного узла (x_i, t_k) получим следующие уравнения:

$$\frac{u_i^k - u_i^{k-1}}{\tau} = a^2 \frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{h^2} + g_i^k, \qquad (1.53)$$

(где $i = 1, 2, ..., n-1$; $k = 1, 2, ..., m$);

$$\frac{u_i^{k+1} - u_i^k}{\tau} = a^2 \frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{h^2} + g_i^k, \qquad (1.54)$$

(где $i = 1, 2, ..., n-1$; $k = 0, 1, ..., m-1$);

$$\frac{u_i^{k+1} - u_i^{k-1}}{2\tau} = a^2 \frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{h^2} + g_i^k, \qquad (1.55)$$

(где $i = 1, 2, ..., n-1$; $k = 1, 2, ..., m-1$).

Будем теперь считать, что расчетный узел (x_i, t_k) смещается по сетке Ω_h^τ, т. е. полагаем в уравнениях (1.53)–(1.55) $i = 1, 2, ..., n-1$, $k = 1, 2, ..., m-1$. Учитывая, что первыми из используемых в расчетах узлами должны быть точки нулевого слоя, а последними – *m*-го слоя, дополняем множество возможных здесь значений *k* значением *k*= 0 для уравнения (1.53) и, значением *k* = *m* для уравнения (1.54).

Заметим, что начальное (1.45) и граничные (1.46) условия непременно должны быть согласованными в точках (0;0) и (*l*;0) ($u_0^0 = \alpha^0 = \varphi_0$, $u_n^0 = \beta^0 = \varphi_n$), в данном случае это условия

$$u_i^0 = \varphi_i \quad (:= \varphi(x_i)), \quad i = 0, 1, ..., n, \qquad (1.56)$$

$$u_0^k = \alpha^k, \quad u_n^k = \beta^k, \quad k = 0, 1, ..., m. \qquad (1.57)$$

На рис. 2.4 показаны шаблоны, отвечающие разностным схемам (1.53), (1.54) и (1.55) (*а*, *б* и *в* соответственно).

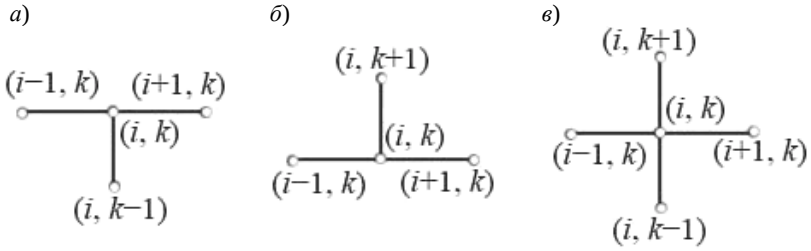


Рис. 2.4. Шаблоны для параболического уравнения: *а*) неявный двухслойный; *б*) явный двухслойный; *в*) явный трехслойный.

Введем постоянную (при заданной сетке)

$$\gamma := a^2 \frac{\tau}{h^2} \tag{1.58}$$

и перепишем разностные схемы (1.53)–(1.55) соответственно следующим образом:

$$\gamma u_{i-1}^k - (1+2\gamma)u_i^k + \gamma u_{i+1}^k = -u_i^{k-1} + \tau g_i^k, \tag{1.59}$$
$$(\text{где } i = 1, 2, ..., n-1;\ k = 1, 2, ..., m);$$

$$u_i^{k+1} = \gamma u_{i-1}^k + (1-2\gamma)u_i^k + \gamma u_{i+1}^k + \tau g_i^k, \tag{1.60}$$
$$(\text{где } i = 1, 2, ..., n-1;\ k = 0, 1, ..., m-1);$$

$$u_i^{k+1} = u_i^{k-1} + 2\gamma u_{i-1}^k - 4\gamma u_i^k + 2\gamma u_{i+1}^k + 2\tau g_i^k, \tag{1.61}$$
$$(\text{где } i = 1, 2, ..., n-1;\ k = 1, 2, ..., m-1).$$

Эти записи позволяют увидеть, как можно осуществить процесс заполнения $(n-1)\cdot m$-таблицы значениями u_i^k, определяемыми каждой из трех представленных схем.

Учитывая, что на нулевом слое значения u_i^0, согласно (1.56), известны при любом $i \in \{0, 1, ..., n\}$, формула (1.60) позволяет непосредственно вычислить все приближенные значения u_i^1, первого слоя.

Эти вычисления можно производить или последовательно, одно за другим, полагая $i = 1,2,..., n–1$, или параллельно, все сразу, при этих же значениях i, подбирая естественные возможности оптимального использования ресурсов компьютера с параллельной обработкой информации. Таким же образом, привлекая найденные значения u_i^1 первого слоя, можно подсчитать значения u_i^2 второго слоя, только при этом для вычислений u_i^2 и u_{n-1}^2 потребуется еще подставить в формулу граничные значения $u_0^1 = \alpha^1$ и $u_n^1 = \beta^1$ соответственно (*см.* (1.57)). Дальнейшие переходы от слоя к слою не отличаются от этого.

В равенстве (1.59) при $k = 1$ и любом $i \in \{1, 2, ..., n-1\}$ известна только правая часть благодаря начальным данным (1.56). Следовательно, для нахождения значений u_i^1 в узлах первого слоя нужно решить систему линейных алгебраических уравнений (СЛАУ):

$$\gamma u_{i-1}^1 - (1+2\gamma)u_i^1 + \gamma u_{i+1}^1 = -\varphi_i - \tau g_i^1,$$

где $i = 1, 2, ..., n-1$; $u_0^1 = \alpha^1$, $u_n^1 = \beta^1$.

Матрица этой системы имеет трехдиагональную структуру с диагональным преобладанием $(1+2\gamma > \gamma+\gamma)$, а это означает, что система может быть эффективно решена методом прогонки (*см.* 2 раздел). Значения искомой функции u в узлах второго слоя получаются как решение системы

$$\gamma u_{i-1}^2 - (1+2\gamma)u_i^2 + \gamma u_{i+1}^2 = -u_i^1 - \tau g_i^2,$$

где $i = 1, 2, ..., n-1$; $u_0^2 = \alpha^2$, $u_n^2 = \beta^2$,

и так далее, т. е. всего для заполнения таблицы требуется решить m однотипных СЛАУ.

Каждая из двух рассмотренных разностных схем связывает значения искомой функции на двух соседних слоях – *двухслойные схемы.* Разностная схема (1.60) представляет собой формулу для непосредственного вычисления искомых значений очередного слоя – *явная*

схема, а схема (1.59) требует при переходе от слоя к слою решения системы алгебраических уравнений – *неявная схема.*

Разностная схема (1.61) считается *явной трехслойной схемой.* Начать процесс вычислений по этой схеме можно только при $k = 1$, т. е. с формулы

$$u_i^2 = 2\gamma u_{i-1}^1 - 4\gamma u_i^1 + 2\gamma u_{i+1}^1 + \varphi_i + 2\tau g_i^1, \qquad (1.62)$$

где значения u_i^1 ($i = 1, 2,..., n–1$) еще не подсчитаны, т. е. нужна дополнительная связь между неизвестными и известными величинами. Такую дополнительную связь между значениями искомой сеточной функции трех первых слоев можно получить, например, беря в качестве расчетного узел нулевого слоя и применяя к $\partial u / \partial t$ формулу несимметричной аппроксимации второго порядка точности. Имеем такой шаблон:

$$\frac{-3u_i^0 + 4u_i^1 - u_i^2}{2\tau} = a^2 \frac{u_{i-1}^0 - 2u_i^0 + u_{i+1}^0}{h^2} + g_i^0,$$

откуда вытекает другое равенство того же вида, что и (1.62):

$$u_i^2 = 4u_i^1 - 2\gamma\varphi_{i-1} + (2\gamma - 3)\varphi_i - 2\gamma\varphi_{i+1} + 2\tau g_i^0.$$

Сравнением правых частей последнего равенства и равенства (1.62) приходим к трехточечному разностному уравнению второго порядка

$$\gamma u_{i-1}^1 - (2 + 2\gamma)u_i^1 + \gamma u_{i+1}^1 = -\gamma\varphi_{i-1} + (\gamma - 2)\varphi_i - \gamma\varphi_{i+1} - \tau g_i^0 - \tau g_i^1,$$

решая которое методом прогонки, находим значения функции u в узлах первого слоя, после чего становится возможным счет по формуле (1.62) и все последующие вычисления по общей формуле (1.61) при $k = 2, 3,..., m–1$.

1.2.5. Аппроксимация, устойчивость, сходимость разностных схем для уравнения теплопроводности

Установление факта и порядка аппроксимации задачи (1.44)–(1.46), рассмотренными выше разностными схемами в предположении о достаточной гладкости решения $u(x, t)$, не вызывает затруднений. Сравнение этих схем в виде (1.53)–(1.55) с конечномерными уравнениями, которые получаются в результате подстановки точных равенств (1.49)–(1.52) в исходное уравнение (1.44), показывает, что неявная и явная двухслойные разностные схемы (1.59) и (1.60) аппроксимируют уравнение (1.44) с погрешностью $O(\tau)+O(h^2)$, а явная трехслойная схема (1.61) – с погрешностью $O(\tau^2)+O(h^2)$. Так как при проектировании граничных и начальных условий данной задачи на сетку $\overline{\Omega}_h^\tau$ искажений (зависящих от h и от τ) не вносится, то можно считать, что с такими же погрешностями эта задача в целом аппроксимируется соответствующими дискретными задачами вместе с условиями (1.56)–(1.57). О рассмотренных двухслойных схемах, основанных на четырехточечных шаблонах (*см*. рис. 2.4, *а* и *б*), говорят, что они аппроксимируют данную параболическую задачу (1.44)–(1.46) со вторым порядком по пространственной переменной x и с первым порядком по временной переменной t, и отражает этот факт запись погрешности аппроксимации задачи вида $O(\tau+h^2)$. Для трехслойной схемы (1.61) погрешность аппроксимации составляет величину $O(\tau^2+h^2)$.

Погрешность аппроксимации данной задачи с помощью каждой из трех рассматриваемых разностных схем стремится к нулю при $h\to 0,\ \tau\to 0$. Это и называется *условием согласованности разностной схемы* и является лишь необходимым условием стремления к нулю глобальной ошибки, которая может накапливаться по тому или иному закону от слоя к слою. Способность разностной схемы при условии однозначной разрешимости не допускать неограниченного увеличения ошибки в процессе измельчения сетки и означает ее устойчивость. Поскольку источником первоначальной ошибки может служить переход к сеточным функциям при дискретизации исходного уравнения, а также неточности в начальных и граничных условиях,

изучение устойчивости разностной схемы в целом разбивают, соответственно, на изучение устойчивости по правой части, по начальным данным и по граничным условиям.

Обратимся к разностной схеме (1.61). К ней можно применить так называемую ε-*схему* изучения роста единичной ошибки. Суть ее в том, что по исследуемой разностной схеме проводят пробные расчеты, исходя из предположения о единственной ошибке ε в значении u_i^k, т. е. вместо этого значения в расчетную формулу подставляют значение $u_i^k + \varepsilon$ и следят за поведением ошибки на следующих (k + 1)-м, (k + 2)-м, … слоях. Если ошибка имеет тенденцию расти по модулю, то разностная схема должна быть признана неустойчивой. Такой ε-*анализ* трехслойной схемы (1.61) в частном случае $\gamma = 0.5, \quad g \equiv 0$ показывает, что начав процесс вычислений с k-го слоя с одним искаженным значением $u_i^k + \varepsilon$, на следующем слое получим значение u_i^{k+1} с ошибкой – 2ε далее значение u_i^{k+2} будет иметь ошибку 7ε, ..., значение u_i^{k+6} ошибку 1311ε. Ошибка катастрофически растет, что говорит о непригодности разностной схемы (1.61), по крайней мере, при γ = 0.5, хотя она имеет более высокий порядок аппроксимации по сравнению с двумя другими схемами.

Исследования явной двухслойной схемы (1.60) приводят к одному: она устойчива при условии γ < 0,5, что в соответствии с обозначением (1.58) равносильно требованию к шагу по времени

$$\tau \le \frac{h^2}{2a^2}. \qquad (1.63)$$

Разностная схема, устойчивость которой связана с некоторым ограничением на шаг, называется *условно устойчивой.*

В отличие от явной, неявная двухслойная схема (1.59) устойчива при любых γ > 0, т. е. при любом соотношении шагов по времени и по пространственной переменной, связи с чем ее называют *безусловно* или *абсолютно устойчивой разностной схемой.*

Таким образом, можно констатировать сходимость решений двухслойных разностных схем (1.59), (1.60) с совокупностью дополнительных условий (1.56)–(1.57) к решению задачи теплопроводности

(1.44)–(1.46) с погрешностью $O(\tau + h^2)$ при любых $h \to 0$, $\tau \to 0$ в случае неявной схемы и при $h \to 0$, $\tau \le \dfrac{h^2}{2a^2} \to 0$ в случае явной схемы. Очевидно, каждая из этих схем имеет свои достоинства и недостатки. Явная схема проще и требует меньше вычислительных затрат на подсчет значений одного слоя, зато таких слоев должно быть больше из-за ограничения (1.63) на шаг по времени, чем при реализации неявной схемы, допускающей любое соотношение шагов, но требующей решения СЛАУ при подсчете значений каждого слоя. Правда, при этом сопоставлении не следует забывать еще о точности аппроксимации данной задачи разностной схемой, которая для явной схемы (1.60) при ограничении (1.63) теперь есть $O(h^2)$, и чтобы иметь такую же для неявной схемы, нужно в ней брать $\tau := O(h^2)$.

1.2.6. Двухслойный шеститочечный и другие шаблоны для параболических уравнений

Возьмем за основу следующего построения неявную и явную разностные схемы для уравнения (1.44) в их исходной форме (1.53) и (1.54). Перепишем равенство (1.53) в виде

$$\frac{u_i^{k+1} - u_i^k}{\tau} = a^2 \frac{u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}}{h^2} + g_i^{k+1}, \qquad (1.64)$$

(т. е. увеличим в нем на единицу верхний индекс) и сравним результат с сеточным уравнением (1.54). Видим, что в левых частях (1.54) и (1.64) стоит одна и та же аппроксимация производной $\partial u / \partial t$, пригодная для всего отрезка $[t_k, t_{k+1}]$ прямой $x = x_i$ и имеющая наивысшую точность $O(\tau^2)$ в средней точке $t_k + 0.5\tau$ этого отрезка. Дроби в правых частях этих уравнений представляют собой аппроксимации второй производной $\partial^2 u / \partial x^2$ одного типа, но на разных слоях: на слое k в уравнении (1.54) и на слое $k + 1$ в уравнении (1.64); иначе, расчетными точками служат точки $(x_i; t_k)$ в первом и $(x_i; t_{k+1})$ во

втором случаях. Есть некий смысл в том, чтобы в качестве расчетной точки при построении новой схемы взять не узловую точку прямой $x = x_i$, а какую-то промежуточную точку отрезка $[t_k, t_{k+1}]$ этой прямой. В зависимости от того, в каком отношении эта условная расчетная точка будет делить указанный отрезок, соединяющий k-й и $(k + 1)$-й слои, производную $\partial^2 u / \partial x^2$ будем подменять соответствующей линейной комбинацией ее аппроксимаций на слоях k и $k + 1$; ту же линейную комбинацию применяем к сеточной функции $G(x_i, t_k)$. Таким образом, приходим к равенству (по логике предшествующих рассуждений относительно условной расчетной точки вместо двух последних слагаемых в нем лучше использовать значение $G(x_i, \sigma t_k + (1-\sigma)t_{k+1})$):

$$\frac{u_i^{k+1} - u_i^k}{\tau} = \frac{a^2}{h^2}[\sigma(u_{i-1}^k - 2u_i^k + u_{i+1}^k) + \\ + (1-\sigma)(u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1})] + \sigma g_i^k + (1-\sigma) g_i^{k+1}, \qquad (1.65)$$

где $\sigma \in [0,1]$ – вещественный параметр (вес); $i = 1, 2, ..., n-1$; $k = 0, 1, ..., m-1$.

Вновь построенная разностная схема (1.65), называемая *схемой с весами*, обобщает схемы (1.54) и (1.64): при $\sigma = 1$ – это явная схема (1.54), при $\sigma = 0$ – неявная схема (1.64) или, что то же, (1.53). Если $0 < \sigma < 1$, то равенство (1.65) связывает шесть точек двух слоев, т. е. отвечает шаблону, изображенному на рис. 2.5.

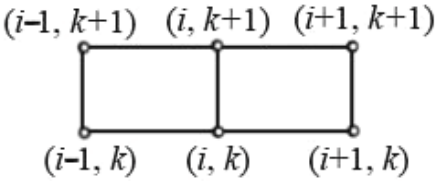


Рис. 2.5. Двухслойный шеститочечный шаблон для разностной схемы (1.65).

Заметим, что явной схема (1.65) является только при значении веса $\sigma = 1$; при всех остальных значениях $\sigma \in [0, 1]$ она – неявная. Чтобы отличить от всех неявных схем этого однопараметрического семейства рассмотренную ранее неявную схему (1.53), соответствующую четырехточечному шаблону, ее зачастую называют *чисто неявной схемой*. Для неявных схем применяют также названия *схемы с опережением* или *схемы с упреждением*.

Другим важным частным случаем семейства формул (1.65) является случай $\sigma = 0.5$. При этом значении σ с учетом обозначения постоянной γ (1.58) из (1.59) имеем равенство

$$u_i^{k+1} - u_i^k = \frac{\gamma}{2}(u_{i-1}^k - 2u_i^k + u_{i+1}^k) +$$

$$+ \frac{\gamma}{2}(u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}) + \frac{\tau}{2}(g_i^k + g_i^{k+1}),$$

которому далее придаем типичный вид трехточечного разностного уравнения второго порядка относительно неизвестных значений на $(k + 1)$-м слое:

$$\gamma u_{i-1}^{k+1} - (2 + 2\gamma)u_i^{k+1} + \gamma u_{i+1}^{k+1} = G_i^k, \qquad (1.66)$$

где $G_i^k := -\gamma u_{i-1}^k - (2 - 2\gamma)u_i^k - \gamma u_{i+1}^k - \tau g_i^k - \tau g_i^{k+1}$. (1.67)

Полученная неявная двухслойная разностная схема (1.66)– (1.67) (в которой полагаем $i = 1, 2,..., n–1$; $k = 0,1,..., m–1$) называется *схемой Кранка–Николсон*. Она аппроксимирует уравнение (1.44) с точностью $O(\tau^2 + h^2)$, и технология ее использования не отличается от описанной выше для чисто неявной схемы (1.59). Доказана абсолютная устойчивость этой схемы, что позволяет проводить расчеты по ней с произвольными шагами h и τ, обеспечивающими достаточную точность аппроксимации.

При любых других значениях $\sigma \in [0, 1]$, $\sigma \neq 0.5$, погрешность аппроксимации схемы с весами (1.65), (называемой также *обобщенной схемой Кранка–Николсон)*, составляет величину $O(\tau + h^2)$. Для $\sigma \in [0, 0.5]$ схема абсолютно устойчива, если же $\sigma \in (0.5, 1]$, то устойчивость

имеет место при выборе шага по времени, удовлетворяющего неравенству $\tau \le \dfrac{h^2}{2a^2(2\sigma - 1)}$. Схема Кранка–Николсон выделяется из семейства (1.65) как самая предпочтительная.

Наряду с рассмотренными выше применяют и другие схемы аппроксимации уравнения теплопроводности (1.44). Например, две из них.

1. *Неявная пятиточечная трехслойная схема*

$$\frac{3}{2}\frac{u_i^{k+1} - u_i^k}{\tau} - \frac{1}{2}\frac{u_i^k - u_i^{k-1}}{\tau} = a^2 \frac{u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}}{h^2} + g_i^{k+1}; \qquad (1.68)$$

абсолютно устойчива, погрешность аппроксимации $O(\tau^2 + h^2)$.

2. *Схема Дюфорта и Франкела*

$$\frac{u_i^{k+1} - u_i^{k-1}}{2\tau} = a^2 \frac{u_{i+1}^k - u_i^{k+1} - u_i^{k-1} + u_{i-1}^k}{h^2} + g_i^k \qquad (1.69)$$

– явная четырехточечная трехслойная; сходится с порядком $O(\tau + h^2) + O(\tau^2 / h^2)$ при условии, что $\tau \to 0$, $h \to 0$, $\tau / h \to 0$.

2. РЕШЕНИЕ СЛАУ

Все методы *решения линейных алгебраических задач* (наряду с задачей решения СЛАУ, это и вычисление определителей, и обращение матриц, и задачи на собственные значения) можно разбить на два класса: прямые и итерационные.

К простейшим *прямым* методам относятся метод Гаусса и, как частный случай, метод *прогонки*, пригодный для систем уравнений с так называемой трехдиагональной матрицей. Метод прогонки отличается высокой экономичностью, т. к. позволяет существенно сократить количество операций для вычисления корней уравнений.

К простейшим *итерационным* методам относятся методы Якоби, Зейделя (метод Гаусса–Зейделя) и метод верхней релаксации.

Относительно каждой системы линейных уравнений могут поставлены следующие вопросы.

1. Совместна заданная система или нет?

2. В случае, если система совместна, как определить, сколько она имеет решений – одно или несколько?

3. Как найти все решения системы?

2.1. Определения

Определители (детерминанты).

Определителем (детерминантом) n-го порядка называется число D, образованное из n^2 чисел a_{ij} (*элементов*), расположенных в квадратную табличку из n строк и n столбцов следующим образом (i – номер строки, считая сверху; j – номер столбца, считая слева):

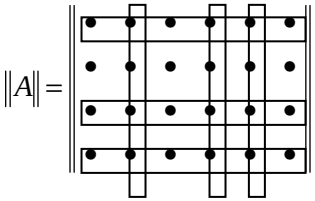
$$D = |a_{ij}| = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \sum (-1)^k a_{i\alpha} a_{2\beta} \dots a_{n\omega},$$

где α, β, ..., ω пробегают все возможные $n!$ перестановок из чисел 1, 2, ..., n; знак «+» или «–» перед каждым слагаемым определяется числом k инверсий в каждой перестановке.

Система mn чисел, расположенных в прямоугольную таблицу из m строк и n столбцов, называется *матрицей*. Обозначение:

$$\|A\| = \begin{Vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{Vmatrix}.$$

Минором k-го порядка матрицы $\|A\|$ $(k \le m, k \le n)$ называется определитель D, составленный (с сохранением порядка) из k^2 элементов матрицы, лежащих на пересечении некоторых ее k столбцов и k строк:



$$D = \begin{vmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{vmatrix}$$ – минор 3-го порядка.

Адъюнктой A_{ij} элемента a_{ij} называется его минор со знаком «плюс» или «минус» по формуле

$$A_{ij} = (-1)^{i+j} \begin{vmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & \dots & a_{nj} & \dots & a_{nn} \end{vmatrix} =$$

$$= (-1)^{i+j} \begin{vmatrix} a_{11} & \dots & a_{1,j-1} & a_{1,j+1} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{i-1,1} & \dots & a_{i-1,j-1} & a_{i-1,j+1} & \dots & a_{i-1,n} \\ a_{i+1,1} & \dots & a_{i+1,j-1} & a_{i+1,j+1} & \dots & a_{i+1,n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & \dots & a_{n,j-1} & a_{n,j+1} & \dots & a_{nn} \end{vmatrix}.$$

Линейной комбинацией нескольких строк (столбцов) определителя называется строка (столбец) $\overline{a_1}, \overline{a_2}, ..., \overline{a_n}$, элементы которой являются линейными комбинациями соответствующих элементов этих строк (столбцов); например, линейная комбинация $\overline{a_1}, \overline{a_2}, ..., \overline{a_n}$ из элементов *i*-й, *j*-й и *k*-й строк определителя образуется так:

$$\begin{aligned} \overline{a_1} &= \alpha a_{i1} + \beta a_{j1} + \gamma a_{k1}, \\ \overline{a_2} &= \alpha a_{i2} + \beta a_{j2} + \gamma a_{k2}, \\ &\dots\dots\dots\dots\dots\dots\dots\dots \\ \overline{a_n} &= \alpha a_{in} + \beta a_{jn} + \gamma a_{kn}; \end{aligned}$$

числа α, β, γ – коэффициенты линейной комбинации.

Свойства определителей:

1) Определитель не изменяет своей величины, если *заменить его строки столбцами* и обратно (поэтому все свойства, касающиеся строк определителя, относятся также к его столбцам).

2) Определитель равен нулю, если *элементы* его двух строк *равны или пропорциональны* или одна из строк является *линейной комбинацией* каких-либо остальных строк.

3)*Множитель*, общий всем элементам какой-либо строки, можно выносить за знак определителя.

4) Если определители, разнящиеся между собой только элементами какой-нибудь (*i*-й) строки, *складывать* между собой, то сумма их равна определителю, у которого элементами *i*-й строки являются суммы соответствующих элементов *i*-х строк определителей-слагаемых, а остальные элементы те же, что у слагаемых.

5) Прибавляя (вычитая) к какой-либо строке элементы другой строки или линейную комбинацию других строк, мы *не изменяем* величины определителя.

6) Определитель можно *разложить по элементам любой* (*i*-й) *строки* по формуле $D = a_{i1}A_{i1} + a_{i2}A_{i2} + ... + a_{in}A_{in}$, где A_{ij} – адъюнкты соответствующих элементов.

7) Сумма произведений всех элементов a_{ik} какой-либо (*i*-й) строки определителя на адъюнкты A_{jk} соответствующих элементов *другой* (*j*-й) строки равна нулю: $a_{i1}A_{j1} + a_{i2}A_{j2} + ... + a_{in}A_{jn} \neq 0$ ($i \neq j$).

Вычисление определителей.

Вычисление определителя матрицы 2-го *порядка* производится по формуле:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}.$$

$-$ $+$

Вычисление определителя матрицы *3-го порядка* можно произвести *по «правилу Саррюса»*:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{matrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} -$$

$- \quad - \quad - \quad + \quad + \quad +$

$$- a_{13}a_{22}a_{31} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33}.$$

Вычисление определителя матрицы *n*-го *порядка* заключается к рекуррентному сведению матрицы к (n – 1)-му порядку по свойству 6 (матрицу преобразуют, пользуясь остальными свойствами, к виду, содержащему возможно большее число элементов равных нулю).

Пример (вычисление определителя с применением свойств):

$$D = \begin{vmatrix} 2 & 3 & -4 & 5 \\ 3 & -5 & 2 & 4 \\ 5 & 4 & 3 & -2 \\ -4 & 2 & 5 & 3 \end{vmatrix} = \begin{vmatrix} 2 & 3 & -4 & 5 \\ 1 & -8 & 6 & -1 \\ 2 & 9 & 1 & -6 \\ 1 & 6 & 8 & 1 \end{vmatrix} =$$

(4 строка + 3 строка, 3 строка – 2 строка, 2 строка – 1 строка)

$$= \begin{vmatrix} 0 & -9 & -20 & 3 \\ 0 & -14 & -2 & -2 \\ 0 & -3 & -15 & -8 \\ 1 & 6 & 8 & 1 \end{vmatrix} = - \begin{vmatrix} -9 & -20 & 3 \\ -14 & -2 & -2 \\ -3 & -15 & -8 \end{vmatrix} =$$

(2 строка – 4 строка, 1 строка – 2·4 строка, 3 строка – 2·4 строка)

$$= 2 \begin{vmatrix} 9 & 20 & -3 \\ 7 & 1 & 1 \\ 3 & 15 & 8 \end{vmatrix} =$$

(из строк 1, 2, 3 выносятся множители –1, –2, –1 соответственно)

$$= 2 \begin{vmatrix} 30 & 23 & -3 \\ 0 & 0 & 1 \\ -53 & 7 & 8 \end{vmatrix} = -2 \begin{vmatrix} 30 & 23 \\ -53 & 7 \end{vmatrix} =$$

(2 строка – 3 строка, 1 строка – 7·3 строка)

$$= -2(210 + 1219) = -2858.$$

Ранг матрицы.

Рангом матрацы $\|A\|$ называется наибольший порядок, который могут иметь ее миноры, не обращающиеся в нуль. Для определения ранга матрицы следует рассмотреть все ее миноры порядка l (где l – меньшее из чисел m, n, если $m \neq n$ или $l = m = n$); если хотя бы один из них $\neq 0$, то ранг $\| A \|$ равен l; если же все они $= 0$, то следует рассмотреть все миноры порядка $l - 1$ и т. д. Практически целесообразно поступать наоборот: переходить от миноров меньшего порядка к минорам большего порядка, пользуясь следующим правилом. Если найден минор k-го порядка D_k, отличный от нуля, то остается вычислить только те миноры $(k + 1)$-го порядка, которые представляют собой «окаймление» D_k, например, $\left|\underline{D_k}\right\|$, $\left|\underline{\|D_k}\right|$, $\left|\overline{D_k}\right\|$, $\left|\overline{\|D_k}\right|$.

Если все такие миноры $(k + 1)$-го порядка равны нулю, то ранг матрицы равен k.

Пример. Определить ранг матрицы $\| A \| = \begin{Vmatrix} 1 & 3 & 2 & 0 \\ 2 & 6 & 9 & 7 \\ -2 & -5 & 2 & 4 \\ 1 & 4 & 8 & 4 \end{Vmatrix}$.

Минор 2-го порядка, стоящий в левом верхнем углу, $D_2 = \begin{vmatrix} 1 & 3 \\ 2 & 6 \end{vmatrix} =$ 0. Но в матрице $\|A\|$ есть минор 2-го порядка, не равный нулю: $D_2' = \begin{vmatrix} 2 & 4 \\ 8 & 4 \end{vmatrix} \neq 0$.

Окаймляем, например, его слева и снизу: $D_3 = \begin{vmatrix} 2 & 6 & 9 \\ -2 & -5 & 2 \\ 1 & 4 & 8 \end{vmatrix} = -15 \neq 0$.

Другие варианты окаймления можно не пробовать, так как уже понятно, что ранг $\|A\| = 3$.

Неоднородные уравнения.

Система неоднородных уравнений

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m, \end{cases} \qquad (2.1)$$

называется *совместной*, если существует хотя бы одно решение $\{x_1, x_2, \ldots, x_n\}$, обращающее все уравнения (2.1) в тождества, и *несовместной*, если ни одного такого решения не существует. *Признак совместности:* система (2.1) совместна тогда и только тогда, когда ранг матрицы

$$\|A\| = \begin{Vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{Vmatrix}$$

равен рангу «расширенной» матрицы

$$\|B\| = \begin{Vmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_m \end{Vmatrix}.$$

Совместная система уравнений называется *определенной*, если она имеет единственное решение, и *неопределенной*, если решений – бесконечное множество.

Совместная система уравнений (2.1) решается следующим образом. Определяем ранг r матрицы $||A||$, переставляем уравнения (2.1), а также и неизвестные $x_1, x_2, ..., x_n$ в них так, чтобы отличный от нуля минор матрицы $||A||$, имеющий порядок r, занял положение в верхнем левом углу матрицы (эта перестановка не нужна, если верхний левый угловой минор матрицы порядка r не равен нулю).

При этом могут быть два случая:

1) $r = n$, $r \le m$. Решая систему первых уравнений с n неизвестными, получаем единственное решение $\{\alpha_1, \alpha_2, ..., \alpha_n\}$, т. к. определитель этой системы $\neq 0$. Если $n < m$, то это же решение удовлетворяет и остальным $m - n$ уравнениям, которые являются следствиями первых. Данная система (2.1) – определенная.

2) $r < n$, $r \le m$. Решаем систему первых r уравнений относительно первых r неизвестных $x_1, x_2, ..., x_r$, выражая эти неизвестные через остальные $n - r$ неизвестных $x_{r+1}, x_{r+2}, ..., x_n$. Получаем единственное решение в виде линейных функций:

$$\begin{cases} x_1 = x_1(x_{r+1}, x_{r+2}, ..., x_n), \\ x_2 = x_2(x_{r+1}, x_{r+2}, ..., x_n), \\ ..... \\ x_r = x_r(x_{r+1}, x_{r+2}, ..., x_n), \end{cases} \qquad (2.2)$$

т. к. определитель системы уравнений $\neq 0$. Неизвестным $x_{r+1}, ..., x_n$ можно придавать любые значения, тогда неизвестные $x_1, ..., x_r$ определяются по формулам (2.2). Эти же решения удовлетворяют и остальным $m - r$ уравнениям (если $r < m$), которые являются следствиями первых. Данная система (2.1) – неопределенная.

Примеры.

1)

$a - 2b + 3c - d + 2e = 2$

$3a - b + 5c - 3d - e = 6$

$2a + b + 2c - 2d - 3e = 8$

Ранг матрицы $||A||$ равен 2, ранг матрицы $||B||$ равен 3. Система несовместная, решений нет.

2)

$a - b + 2c = 1$

$a - 2b - c = 2$

$3a - b + 5c = 3$

$-2a + 2b + 3c = -4$

Ранги матриц $||A||$ и $||B||$ равны 3; система совместна. Угловой верхний левый определитель 3-го порядка $D = \begin{vmatrix} 1 & -1 & 2 \\ 1 & -2 & -1 \\ 3 & -1 & 5 \end{vmatrix} \neq 0$; перестановки не требуется. $r = n$; система определенная. Решаем систему первых трех уравнений: $a = 10/7$; $b = -1/7$; $c = -2/7$; это же решение удовлетворяет и четвертому уравнению.

3)

$a - b + c - d = 1$

$a - b - c + d = 0$

$a - b - 2c + 2d = -0.5$

Ранги матриц ||*A*|| и ||*B*|| равны 2; система совместная. $r < n$; система неопределенная. Угловой верхний левый определитель 2-го порядка $D = 0$; переставляем столбец с *a* на 4-е место:

$-b + c - d + a = 1$

$-b - c + d + a = 0$

$-b - 2c + 2d + a = -0.5$

Решаем систему первых двух уравнений относительно неизвестных *b* и *c:*

$-b + c = 1 - a + d$

$-b - c = -a - d$

Решения $b = a - 0.5$; $c = d + 0.5$ удовлетворяют всем уравнениям при любых значениях *a* и *d*.

4)

$a + 2b - c + d = 1$

$2a - b + 2c + 2d = 2$

$3a + b + c + 3d = 3$

$a - 3b + 3c + d = 0$

В этом случае число уравнений равно числу неизвестных, $D = 0$, а также $D_a = D_b = D_c =: D_d = 0$. Ранг матрицы ||*A*|| равен 2, ранг матрицы ||*B*|| равен 3. Система несовместная, решений нет.

Однородные уравнения.

Система однородных уравнений

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = 0, \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = 0, \\ ..... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = 0, \end{cases} \qquad (2.3)$$

всегда имеет *нулевое решение* $x_1 = x_2 = ... = x_n = 0$. Если система имеет ненулевое решение $\{\alpha_1, \alpha_2, ..., \alpha_n\}$, то она имеет и бесконечное множество решений вида $\{k\alpha_1, k\alpha_2, ..., k\alpha_n\}$, где *k* – любое число. Если система (2.3) имеет *p* ненулевых решений

$$\{\alpha_1, \alpha_2, ..., \alpha_n\},\ \{\beta_1, \beta_2, ..., \beta_n\},\ ...,\ \{\mu_1, \mu_2, ..., \mu_n\}, \qquad (2.4)$$

то она имеет и бесконечное множество решений вида

$$\{k_1\alpha_1 + k_2\beta_1 + ... + k_p\mu_1,\ k_1\alpha_2 + k_2\beta_2 + ... + k_p\mu_2, ... \\ ..., k_1\alpha_n + k_2\beta_n + ... + k_p\mu_n\}, \qquad (2.5)$$

где $k_1, k_2, ..., k_p$ – любые числа (не все равные нулю). Решение (2.5) называется *линейной комбинацией* решений (2.4).

Решения (2.4) системы уравнении (2.3) называются *линейно независимыми*, если ни одно из них не является линейной комбинацией остальных; p линейно независимых решений образуют *фундаментальную систему решений*, если любое решение системы уравнений (2.3) является линейной комбинацией этих p решений (фундаментальных систем может быть бесконечное множество).

Если ранг r матрицы $\|A\|$ из коэффициентов уравнений (2.3) меньше числа n неизвестных, то уравнения (2.3) имеют фундаментальную систему решений; если же $r = n$, фундаментальной системы не существует, и уравнения имеют только нулевое решение. При $r < n$ фундаментальная система состоит из $n - r$ линейно независимых решений. Для нахождения фундаментальных систем ставим в системе (2.3) уравнения и неизвестные в таком порядке, чтобы в левом верхнем углу матрицы $\|A\|$ оказался минор r-го порядка, неравный нулю (для этого приходится иногда переставлять уравнения и неизвестные). Затем решаем систему (2.3) относительно первых r неизвестных $x_1, x_2, ..., x_r$, выражая их через остальные неизвестные:

$$\begin{cases} x_1 = x_1(x_{r+1}, x_{r+2}, ..., x_n), \\ x_2 = x_2(x_{r+1}, x_{r+2}, ..., x_n), \\ \quad ..... \\ x_r = x_r(x_{r+1}, x_{r+2}, ..., x_n), \end{cases} \qquad (2.6)$$

Неизвестным $x_{r+1}, x_{r+2}, ..., x_n$ можно давать любые значения, и они вместе с соответствующими значениями $x_1, x_2, ..., x_r$, определяемы-

ми по формулам (2.6), образуют одно из решений системы уравнений (2.3). Выбрав эти значения $n - r$ раз:

	x_{r+1}	x_{r+2}	…	x_n
1)	b_{11}	b_{12}	…	$b_{1,n-r}$
2)	b_{21}	b_{22}	…	$b_{2,n-r}$
…….				
$n-r$)	$b_{n-r,1}$	$b_{n-r,2}$	…	$b_{n-r,n-r}$

таким образом, чтобы определитель $B = |b_{ik}|$ не равнялся нулю, получаем одну из фундаментальных систем решений уравнений (2.3). В частности, можно получить $b_{ik} = 1$ при $i = k$ и $b_{ik} = 0$ при $i \neq k$; тогда $B = 1$ и решения

	x_{r+1}	x_{r+2}	…	x_n
1)	1	0	…	0
2)	0	1	…	0
…….				
$n-r$)	0	0	…	1

вместе с (2.5) определяют простейшим способом фундаментальную систему решений уравнений (2.3).

Примеры.

Найти фундаментальные системы решений уравнений

1)

$a - b + 5c - d = 0$

$a + b - 2c + 3d = 0$

$3a - b + 8c + d = 0$

$a + 3b - 9c + 7d = 0$

Ранг матрицы $\|A\|$ равен 2; определитель $\begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix} \neq 0$, перестановки делать не надо. Решаем систему относительно неизвестных a и b. Полагая $c = 1$, $d = 0$, получаем первое фундаментальное решение: $a = -3/2$; $b = 7/2$; $c = 1$; $d = 0$ или $\{-3/2, 7/2, 1, 0\}$.

Полагая $c = 0$, $d = 1$, получаем второе фундаментальное решение: $a = -1$; $b = -2$; $c = 0$; $d = 1$ или $\{-1, -2, 0, 1\}$.

Следовательно, любое решение данной системы можно представить в виде $\{-3/2k_1 - k_2, 7/2k_1 - 2k_2, k_1, k_2\}$.

2)
$2a + 3b - c = 0$
$a - b + c = 0$
$3a + 2b = 0$

Число уравнений равно числу неизвестных, $D = D_a = D_b = D_c = 0$.

Ранг матрицы $\|A\|$ равен 2; определитель $\begin{vmatrix} 2 & 3 \\ 1 & -1 \end{vmatrix} \neq 0$ перестановки делать не надо.

Решаем систему относительно a и b. Полагая $c = 1$, получаем единственное линейно независимое фундаментальное решение: $a = -2/5$; $b = 3/5$.

Следовательно, любое решение данной системы можно представить в виде: $a = -2/5k$, $b = 3/5k$, $c = k$ или а = $-2k$, $b = 3k$, $c = 5k$.

2.2. Прямые методы

В данном разделе будут рассмотрены только прямые методы, т. е. такие методы, которые приводят к решению за конечное число арифметических операций. Если операции реализуются точно, то и решение также будет точным (в связи с чем, к классу прямых методов применяют еще название точные методы).

Изучаем вопрос о численном решении систем вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2, \\ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n, \end{cases} \tag{2.7}$$

или иначе, векторно-матричных уравнений

$$\mathbf{Ax} = \mathbf{b}, \tag{2.8}$$

где $\mathbf{b} = (b_1, b_2, ..., b_n)^T$ – вектор свободных членов и $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ – вектор неизвестных (он же в другой интерпретации может означать и вектор-решение) с вещественными координатами, а $\mathbf{A} = (a_{ij})_{i,j=1}^{n}$ – вещественная $n \times n$-матрица коэффициентов данной системы. Эффективность способов решения системы (2.7) во многом зависит от структуры и свойств матрицы **A**: размера, обусловленности, симметричности, заполненности (т. е. соотношения между числом ненулевых и нулевых элементов), специфики расположения ненулевых элементов в матрице и др.

2.2.1. Метод Крамера

Размерность системы (т. е. число n) является главным фактором, заставляющим вычислителей не применять обоснованных в теоретическом плане и приемлемых на практике при небольших n *формул Крамера*

$$x_i = \frac{\det \mathbf{A}_i}{\det \mathbf{A}} \quad (i = 1, 2, ..., n), \tag{2.9}$$

позволяющих находить неизвестные компоненты вектора **x** в виде дробей, знаменателем которых является определитель матрицы системы, а числителем – определители матриц $\mathbf{A}_i$, полученные из **A** заменой столбца коэффициентов при вычисляемом неизвестном столбцом свободных членов. Если при реализации этих формул определители вычисляются понижением порядка на основе разложения по элементам какой-нибудь строки или столбца матрицы, то на вычисление определителя n-го порядка будет затрачиваться $n!$ операций умножения. Факториальный рост количества арифметических операций с увеличением размерности задачи называют «проклятьем размерности». Зафиксировав, например, n = 100 и оценив величину $100! \approx 10^{158}$, можно сделать вывод о том, что системы сотого порядка не могут быть решены по формулам Крамера современными вычислительными машинами. Метод Крамера будет неустойчив, т. е. погрешности округлений будут катастрофически нарастать, и размер-

ность $n = 100$ для современных задач не так и велика, часто решаются системы с сотнями и с тысячами неизвестных.

Если осуществлять вычисление обратной матрицы с помощью союзной матрицы, т. е. через алгебраические дополнения, то нахождение решения векторно-матричного уравнения (2.9) по формуле

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

фактически равнозначно применению формул Крамера и также практически непригодно по упомянутым выше причинам для вычислительных целей.

Рассмотрим случай, когда число неизвестных равно числу уравнений – *каноническая система*

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2, \\ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n. \end{cases} \qquad (2.10)$$

Напомним: $D = \begin{vmatrix} a_{11} & \ldots & a_{1n} \\ \ldots & \ldots & \ldots \\ a_{n1} & \ldots & a_{nn} \end{vmatrix}$ (*определитель системы*). D_j – определитель, получающийся из D заменой столбца, составленного из коэффициентов a_{kj}, при неизвестном x_j, столбцом, составленным из свободных членов b_k.; например $D_1 = \begin{vmatrix} b_1 & a_{12} & \ldots & a_{1n} \\ \ldots & \ldots & \ldots & \ldots \\ b_n & a_{n2} & \ldots. & a_{nn} \end{vmatrix}$. Система (2.10) называется *однородной*, если все $b_k=0$ (и значит все $D_j = 0$), и *неоднородной*, если хотя бы одно b_k отлично от нуля.

Решение системы (2.10). Если определитель системы $D \neq 0$, то система (2.10) – *определенная*; она имеет одно решение: корни x_i выражаются *формулами Крамера*:

$$x_1 = D_1 / D\,,\ x_2 = D_2 / D\,,\ \ldots,\ x_n = D_n / D\,.$$

Если $D = 0$ и не все $D_j = 0$, то система (2.10) – *несовместная*: она не имеет ни одного решения (для однородной системы такого случая быть не может).

Примеры.

1)

$2a + b + 3c = 9$

$a - 2b + c = -2$

$3a + 2b + 2c = 7$

$$D = \begin{vmatrix} 2 & 1 & 3 \\ 1 & -2 & 1 \\ 3 & 2 & 2 \end{vmatrix} = 13,\ D_a = \begin{vmatrix} 9 & 1 & 3 \\ -2 & -2 & 1 \\ 7 & 2 & 2 \end{vmatrix} = -13,\ D_b = \begin{vmatrix} 2 & 9 & 3 \\ 1 & -2 & 1 \\ 3 & 7 & 2 \end{vmatrix} = 26,$$

$$D_c = \begin{vmatrix} 2 & 1 & 9 \\ 1 & -2 & -2 \\ 3 & 2 & 7 \end{vmatrix} = 39.$$

$a = D_a / D = -13/13 = -1,\ b = D_b / D = 26/13 = 2,\ c = D_c / D = 39/13 = 3.$

Система определенная, неоднородная.

2)

$2a + 3b - c = 1$

$a - b + c = 2$

$3a + 2b = 5$

$$D = \begin{vmatrix} 2 & 3 & -1 \\ 1 & -1 & 1 \\ 3 & 2 & 0 \end{vmatrix} = 0,\ D_a = \begin{vmatrix} 1 & 3 & -1 \\ 2 & -1 & 1 \\ 5 & 2 & 0 \end{vmatrix} = 4.$$

Система несовместная.

2.2.2. Метод Гаусса с постолбцовым выбором главного элемента

Наиболее известным и популярным способом решения линейных систем вида (2.7) является метод Гаусса. Суть его проста – это последовательное исключение неизвестных. Нас будет интересовать

технология получения вектора-решения **x** из исходных матрицы **A** и вектора **b**, причем, по возможности, минимизирующая влияние неизбежных ошибок округлений. С этой целью, работая с уравнениями системы (2.7), выведем сначала совокупность формул, позволяющих в итоге получить искомые значения неизвестных, а затем на их основе запишем алгоритм решения поставленной задачи.

Будем поэтапно приводить систему (2.7) к треугольному виду, исключая последовательно сначала x_1 из второго, третьего, ... , n-го уравнений, затем x_2 из третьего, четвертого, ... , n-го уравнений преобразованной системы, и т. д.

На первом этапе заменим второе, третье, ... , n-е уравнения на уравнения, получающиеся сложением этих уравнений с первым, умноженным соответственно на $-a_{21}/a_{11}$, $-a_{31}/a_{11}$, …, $-a_{n1}/a_{11}$. Результатом этого этапа преобразований будет эквивалентная (2.7) система

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + ... + a_{1n}x_n = b_1, \\ \quad a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + ... + a_{2n}^{(1)}x_n = b_2^{(1)}, \\ \quad a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 + ... + a_{3n}^{(1)}x_n = b_3^{(1)}, \\ \quad ..... \\ \quad a_{n2}^{(1)}x_2 + a_{n3}^{(1)}x_3 + ... + a_{nn}^{(1)}x_n = b_n^{(1)}, \end{cases} \qquad (2.11)$$

коэффициенты которой (с верхним индексом 1) подсчитываются по формулам

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}}a_{ij},\ b_i^{(1)} = b_i - \frac{a_{i1}}{a_{11}}b_1, \text{ где } i, j = 2, 3, ..., n.$$

При этом можно считать, что $a_{11} \neq 0$, так как по предположению система (2.7) однозначно разрешима, значит, все коэффициенты при x_1 не могут одновременно равняться нулю, и на первое место всегда можно поставить уравнение с отличным от нуля первым коэффициентом.

На втором этапе проделываем такие же операции, как и на первом, с подсистемой системы (2.11), получающейся исключением первого

уравнения. Эквивалентный (2.11) (а значит, и (2.7)) результат второго этапа будет иметь вид

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1, \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2n}^{(1)}x_n = b_2^{(1)}, \\ a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}x_n = b_3^{(2)}, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n3}^{(2)}x_3 + \ldots + a_{nn}^{(2)}x_n = b_n^{(2)}, \end{cases}$$

где $a_{ij}^{(2)} = a_{ij}^{(2)} - \dfrac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{2j}^{(1)}$, $b_i^{(2)} = b_i^{(1)} - \dfrac{a_{i2}^{(1)}}{a_{22}^{(1)}} b_2^{(1)}$; $i, j = 3, \ldots, n$.

Продолжая этот процесс, на (n–1)-м этапе так называемого *прямого хода* метода Гаусса данную систему (2.7) приведем к треугольному виду:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ a_{22}^{(1)}x_2 + \ldots + a_{2n}^{(1)}x_n = b_2^{(1)}, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{nn}^{(n-1)}x_n = b_n^{(n-1)}. \end{cases} \qquad (2.12)$$

На основе предыдущих рассуждений и формул можно убедиться, что коэффициенты этой системы могут быть получены из коэффициентов данной системы последовательным пересчетом по формулам

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)}, \; b_i^{(k)} = b_i^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} b_k^{(k-1)}, \qquad (2.13)$$

где верхний индекс k (номер этапа) должен изменяться от 1 до n–1, нижние индексы i и j (в любой очередности) – от k+1 до n; по определению полагаем $a_{ij}^{(0)} := a_{ij}$, $b_i^{(0)} := b_i$.

Треугольная, точнее, трапециевидная структура системы (2.12) позволяет последовательно одно за другим вычислять значения неизвестных, начиная с последнего:

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}};$$

$$\cdots\cdots\cdots\cdots\cdots$$

$$x_2 = \frac{b_2^{(1)} - a_{23}^{(1)} x_3 - \ldots - a_{2n}^{(1)} x_n}{a_{22}^{(1)}};$$

$$x_1 = \frac{b_1 - a_{12} x_2 - \ldots - a_{1n} x_n}{a_{11}}.$$

Этот процесс последовательного вычисления значений неизвестных называют *обратным ходом* метода Гаусса. Он определяется одной формулой

$$x_k = \frac{1}{a_{kk}^{(k-1)}} \left(b_k^{(k-1)} - \sum_{j=k+1}^{n} a_{kj}^{(k-1)} x_j \right), \qquad (2.14)$$

где k полагают равным n, n–1, ..., 2, 1 и сумма по определению считается равной нулю, если нижний предел суммирования у знака Σ имеет значение больше верхнего.

Итак, решение СЛАУ вида (2.7) методом Гаусса сводится к последовательной реализации вычислений по формулам (2.13) и (2.14).

Так как реальные машинные вычисления производятся не с точными, а с усеченными числами, т. е. неизбежны ошибки округления, то анализируя, например, формулы (2.13), можно сделать вывод о том, что выполнение алгоритма может прекратиться или привести к неверным результатам, если знаменатели дробей на каком-то этапе окажутся равными нулю или очень маленькими числами. Чтобы уменьшить влияние ошибок округлений и исключить деление на нуль, на каждом этапе прямого хода уравнения системы (точнее, обрабатываемой подсистемы) обычно переставляют так, чтобы деление производилось на наибольший по модулю в данном столбце (обрабатываемом подстолбце) элемент. Числа, на которые производится деление в методе Гаусса, называются *ведущими* или *главными элементами*. Отсюда название рассматриваемой модификации метода, исключающей деление на нуль и уменьшающей вычислительные погрешности, – *метод Гаусса с постолбцовым выбором главного элемента* (или, иначе, *с частичным упорядочиванием по столбцам).*

Устойчивость алгоритма к погрешностям исходных данных и результатов промежуточных вычислений можно еще усилить, если выполнять деление на каждом этапе на элемент, наибольший по модулю во всей матрице преобразуемой на данном этапе подсистемы. Такая модификация метода Гаусса, называемая *методом главных элементов*, применяется довольно редко, поскольку сильно усложняет алгоритм. Усложнение связано как с необходимостью осуществления двумерного поиска главных элементов, так и с необходимостью запоминать номера столбцов, откуда берутся эти элементы (перестановка столбцов означает как бы переобозначение неизвестных, в связи с чем, требуется обратная замена).

Пример.

Решаем систему линейных уравнений.

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \quad \text{в } (1;1) \\ x_1 + 2x_2 + 4x_3 &= -1 \quad \text{в } (2;-1) \\ x_1 + 3x_2 + 9x_3 &= 1 \quad \text{в } (3;1). \end{aligned} \tag{2.15}$$

Если вычесть первое уравнение из второго и третьего, то исключается переменная x_1. Это применение преобразования замещения. В результате получаем эквивалентную линейную систему.

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_2 + 3x_3 &= -2 \\ 2x_2 + 8x_3 &= 0. \end{aligned} \tag{2.16}$$

Переменная x_2 исключается из третьего уравнения (2.16) путем двукратного вычитания из него второго уравнения. Мы пришли к эквивалентной верхней треугольной системе линейных уравнений.

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_2 + 3x_3 &= -2 \\ 2x_3 &= 4. \end{aligned} \tag{2.17}$$

А сейчас воспользуемся алгоритмом обратной подстановки для нахождения коэффициентов $x_3 = 4/2 = 2$, $x_2 = -2 - 3(2) = -8$ и $x_1 = 1 - (-8) - 2 = 7$.

Эффективнее всего можно хранить коэффициенты линейной системы $\mathbf{Ax} = \mathbf{b}$, как матрицу размера $N \times (N + 1)$. Коэффициенты $\mathbf{b}$ хранятся в $(N + 1)$-м столбце матрицы (т. е. $a_{kN+1} = b_k$). В каждой строке содержатся все необходимые для уравнения линейной системы коэффициенты. *Расширенная матрица* обозначается как $[\mathbf{A}|\mathbf{b}]$, и систему линейных уравнений представляют в следующем виде:

$$[\mathbf{A}\,|\,\mathbf{b}] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \ldots & a_{1N} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2N} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & \ldots & a_{NN} & b_N \end{array}\right]. \qquad (2.18)$$

Систему $\mathbf{Ax} = \mathbf{b}$ с заданной в (2.18) расширенной матрицей можно решить, выполнив ряд операций над расширенной матрицей $[\mathbf{A}|\mathbf{b}]$. Переменные x_k определяются положением коэффициентов и могут быть опущены до конца вычислений.

2.2.3. Метод исключения Гаусса и выбор главного элемента

Коэффициент a_{rr} матрицы $\mathbf{A}$, который используется, чтобы исключить элементы a_{kr}, где $k = r + 1, r + 2, \ldots, N$, называется *r-м главным элементом* и *r*-я строка – *главной строкой.*

Следующий пример иллюстрирует, как использовать операции, чтобы получить эквивалентную верхнюю треугольную линейную систему $\mathbf{Ux} = \mathbf{y}$ для линейной системы $\mathbf{Ax} = \mathbf{b}$, где $\mathbf{A}$ – матрица размера $N \times N$.

Пример.

Выразим следующую систему в форме расширенной матрицы, найдем эквивалентную ей верхнюю треугольную систему линейных уравнений и ее решение.

$x_1 + 2x_2 + x_3 + 4x_4 = 13$

$2x_1 + 4x_3 + 3x_4 = 28$

$4x_1 + 2x_2 + 2x_3 + x_4 = 20$
$-3x_1 + x_2 + 3x_3 + 2x_4 = 6.$

Расширенная матрица имеет вид

$$\begin{array}{r} \text{гл.эл.} \rightarrow \\ m_{21} = 2 \\ m_{31} = 4 \\ m_{41} = -3 \end{array} \left[\begin{array}{cccc|c} \underline{1} & 2 & 1 & 4 & 13 \\ 2 & 0 & 4 & 3 & 28 \\ 4 & 2 & 2 & 1 & 20 \\ -3 & 1 & 3 & 2 & 6 \end{array}\right].$$

Первая строка используется, чтобы исключить элементы под диагональю в первом столбце. Мы обращаемся к первой строке, как к главной, и называем элемент $a_{11} = 1$ главным. Значение m_{k1} является множителем строки 1, которую вычитаем из k строк, $k = 2, 3, 4$. Результатом первого исключения будет

$$\begin{array}{r} \\ \text{гл.эл.} \rightarrow \\ m_{32} = 1.5 \\ m_{42} = -1.75 \end{array} \left[\begin{array}{cccc|c} 1 & 2 & 1 & 4 & 13 \\ 0 & \underline{-4} & 2 & -5 & 2 \\ 0 & -6 & -2 & -15 & -32 \\ 0 & 7 & 6 & 14 & 45 \end{array}\right].$$

Вторая строка используется, чтобы исключить элементы под диагональю во втором столбце. Эта строка является главной, и значение m_{k1} является множителем строки 2, которую вычитаем из k строк, $k = 3, 4$. Результатом исключения будет

$$\begin{array}{r} \\ \\ \text{гл.эл.} \rightarrow \\ m_{43} = -1.9 \end{array} \left[\begin{array}{cccc|c} 1 & 2 & 1 & 4 & 13 \\ 0 & -4 & 2 & -5 & 2 \\ 0 & 0 & \underline{-5} & -7.5 & -35 \\ 0 & 0 & 9.5 & 5.25 & 48.5 \end{array}\right].$$

Наконец, умножаем $m_{43} = -1.9$ на третью строку, вычитаем из четвертой строки и в результате получаем верхнюю треугольную систему линейных уравнений

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 4 & 13 \\ 0 & -4 & 2 & -5 & 2 \\ 0 & 0 & -5 & -7.5 & -35 \\ 0 & 0 & 0 & -9 & -18 \end{array}\right]. \qquad (2.19)$$

Для решения системы линейных уравнений (2.19) воспользуемся алгоритмом обратной подстановки и получим
$x_4 = 2$, $x_3 = 4$, $x_2 = -1$, $x_1 = 3$.

Описанный выше процесс называется *методом исключения Гаусса* и должен быть модифицирован так, чтобы его можно было использовать в большинстве случаев. Если $a_{kk} = 0$, то строку k нельзя использовать для исключения элементов столбца k и строку k следует заменить такой же строкой под диагональю, чтобы получить не равный нулю главный элемент. Если этого сделать нельзя, значит матрица коэффициентов системы линейных уравнений является вырожденной, и система не имеет единственного решения.

2.3. Итерационные методы для линейных систем

Итерационными методам, т. е. методам, в которых точное решение может быть получено лишь в результате бесконечного повторения единообразных (как правило, простых) действий, и будет посвящен настоящий раздел.

Сходимость.

Следует дать определение близости двух векторов так, чтобы можно было ввести понятие сходимости $\{P_k\}$ к P. Расстояние Евклида между $P = (x_1, x_2,\ldots, x_N)$ и $Q = (y_1, y_2, \ldots, y_N)$ равно

$$\| P - Q \| = \left(\sum_{j=1}^{N} (x_j - y_j)^2 \right)^{1/2}. \qquad (2.20)$$

Это неудобное определение, так как оно требует значительных усилий для вычислений. Поэтому вводится другая норма $\|X\|_1$:

$$\| X \|_1 = \sum_{j=1}^{N} | x_j | . \qquad (2.21)$$

Следующий результат гарантирует, что норма $\|X\|_1$ имеет математическую структуру метрик и, следовательно, подходит для исполь-

зования в качестве обобщения «формулы расстояния». В векторном пространстве ограниченной размерности все нормы эквивалентны, т. е. если два вектора близки в норме $\|*\|_1$, они также близки в норме Евклида $\|*\|$.

Теорема. Пусть X и Y – векторы размера N и c – скаляр. Тогда функция $\|X\|_1$ имеет следующие свойства:

$$\| X \|_1 \geq 0 , \tag{2.22}$$

$$\| X \|_1 = 0 \text{, тогда и только тогда, когда } X = 0 , \tag{2.23}$$

$$\| cX \|_1 = | c | \| X \|_1 , \tag{2.24}$$

$$\| X + Y \|_1 \leq \| X \|_1 + \| Y \|_1 . \tag{2.25}$$

Определение. Предположим, что X и Y – две точки в N-мерном пространстве. Определим расстояние между X и Y в норме $\|*\|_1$ как

$$\| X - Y \|_1 = \sum_{j=1}^{N} | x_j - y_j | .$$

Пример. Определим расстояние Евклида и расстояние $\|*\|_1$ между точками P = (3; 4; 5) и Q = (2.5; 3.5; 4.5). Расстояние Евклида равно
$\| P - Q \| = ((3–2.5)^2 + (4–3.5)^2 + (5–4.5)^2)^{1/2} = 0.866$

Расстояние $\|*\|_1$ равно
$\| P - Q \|_1 = |3–2.5| + |4–3.5| + |5–4.5| = 0.750$

Расстояние $\|*\|_1$ легче вычисляется и используется для определения сходимости в N-мерном пространстве.

2.3.1. Итерация Якоби

Рассмотрим систему линейных уравнений

$$\begin{aligned} 4x - y + z &= 7 \\ 4x - 8y + z &= -21 \\ -2x + y + 5z &= 15. \end{aligned} \tag{2.26}$$

Уравнения можно записать в виде

$$x = (7 + y - z)/4$$
$$y = (21 + 4x + z)/8$$
$$z = (15 + 2x - y)/5. \qquad (2.27)$$

Это позволяет предложить следующий итеративный процесс Якоби:

$$x_{k+1} = (7 + y_k - z_k)/4$$
$$y_{k+1} = (21 + 4x_k + z_k)/8$$
$$z_{k+1} = (15 + 2x_k - y_k)/5. \qquad (2.28)$$

Покажем, что если начать с точки $P_0 = (x_0; y_0; z_0) = (1; 2; 2)$, то итерация (2.28) сходится к решению (2; 4; 3).

Подставим $x_0 = 1$, $y_0 = 2$ и $z_0 = 2$ в правую часть каждого уравнения в (2.28), чтобы получить новые значения:

$$x_1 = (7 + 2 - 2)/4 = 1.75$$
$$y_1 = (21 + 4 + 2)/8 = 3.375$$
$$z_1 = (15 + 2 - 2)/5 = 3.00$$

Новая точка $P_1 = (1.75; 3.375; 3.00)$ ближе к (2; 4; 3), чем P_0. Итерация, использующая (2.28), генерирует последовательность точек $\{P_k\}$, которая сходится к решению (2; 4; 3) (*см*. табл. 2.1).

Этот процесс называется *итерацией Якоби* и может использоваться для решения определенных типов линейных систем. После 19-и шагов итерация сходится к приближению с девятью значащими цифрами (2.00000000; 4.00000000; 3.00000000).

Линейные системы с таким большим количеством переменных, как 100 000, часто возникают в решениях дифференциальных уравнений в частных производных, Матрицы коэффициентов этих систем полупустые, т. е. большой процент элементов матрицы коэффициентов равен нулю. Если существует модель с не равными нулю элементами (т. е. трехдиагональная система), в таком случае итеративный процесс является эффективным методом решения этих больших систем. Иногда метод Якоби не работает. Проведем эксперимент и уви-

дим, что перестановка начальной линейной системы приведет к системе итерационных уравнений, которые дадут расходящуюся последовательность точек.

Таблица 2.1

Сходимость итерации Якоби для систем линейных уравнений (2.26)

k	x_k	y_k	z_k
0	1.0	2.0	2.0
1	1.75	3.375	3.0
2	1.84375	3.875	3.025
3	1.9625	3.925	2.9625
4	1.99062500	3.97656250	3.00000000
5	1.99414063	3.99531250	3.00093750
.	.	.	.
.	.	.	.
.	.	.	.
15	1.99999993	3.99999985	2.99999993
.	.	.	.
.	.	.	.
.	.	.	.
19	2.00000000	4.00000000	3.00000000

Пусть линейная система (2.26) переставлена следующим образом:

$$\begin{aligned} -2x + y + 5z &= 15 \\ 4x - 8y + z &= -21 \\ 4x - y + z &= 7. \end{aligned} \qquad (2.29)$$

Полученные уравнения можно записать в виде

$$\begin{aligned} x &= (-15 + y - 5z)/3 \\ y &= (21 + 4x + z)/8 \\ z &= 7 - 4x + y. \end{aligned} \qquad (2.30)$$

Это подразумевает следующий итеративный процесс Якоби:

$$\begin{aligned} x_{k+1} &= (-15 + y_k + 5z_k)/3 \\ y_{k+1} &= (21 + 4x_k + z_k)/8 \\ z_{k+1} &= 7 - 4x_k + y_k.. \end{aligned} \qquad (2.31)$$

Очевидно, что если начать с точки $P_0 = (x_0; y_0; z_0) = (1; 2; 2)$, то итерация (2.31) отклоняется от решения (2; 4; 3).

Подставим $x_0 = 1$, $y_0 = 2$ и $z_0 = 2$ в правую часть каждого уравнения (2.29), чтобы получить новые значения x_1, y_1 и z_1:

$$x_1 = (-15 + 2 + 10)/2 = -1.5$$
$$y_1 = (21 + 4 + 2)/8 = 3.375$$
$$z_1 = 7 - 4 + 2 = 5.00$$

Новая точка $P_1 = (-1.5; 3.375; 5.00)$ дальше от решения (2; 4; 3), чем P_0 и т. д. Используемые итерационные уравнения (2.31) порождают расходящуюся последовательность.

2.3.2. Итерация Гаусса–Зейделя

Иногда сходимость можно ускорить. Отметим, что итеративный процесс Якоби (2.28) производит три последовательности $\{x_k\}$, $\{y_k\}$ и $\{z_k\}$, которые сходятся соответственно к 2, 4 и 3 (*см.* табл. 2.1). Кажется разумным, что x_{k+1} может быть использовано вместо x_k в вычислении y_{k+1}. Аналогично x_{k+1} и y_{k+1} можно использовать в вычислении z_{k+1}. В следующем примере показано, что произойдет, когда применить это к уравнениям из примера раздела 2.3.1.

Пример.

Рассмотрим систему уравнений, заданную в (2.26), и итеративный процесс Гаусса–Зейделя, использующий (2.27):

$$\begin{aligned} x_{k+1} &= (7 + y_k - z_k)/4 \\ y_{k+1} &= (21 + 4x_{k+1} + z_k)/8 \\ z_{k+1} &= (15 + 2x_{k+1} - y_{k+1})/5. \end{aligned} \qquad (2.32)$$

Ясно, что если начать с точки $P_0 = (x_0; y_0; z_0) = (1; 2; 2)$, то итерация, заданная (2.32), сходится к решению (2; 4; 3).

Подставим $y_0 = 2$ и $z_0 = 2$ в первое уравнение системы (2.32) и получим

$$x_1 = (7 + 2 - 2)/4 = 1.75$$

Затем подставим $x_1 = 1.75$ и $z_0 = 2$ во второе уравнение и получим

$$y_1 = (21 + 4(1.75) + 2)/8 = 3.75$$

Наконец подстановка $x_1 = 1.75$ и $y_1 = 3.75$ в третье уравнение даст

$$z_1 = (15 + 2(1.75) – 3.75)/5 = 2.95$$

Новая точка $P_i = (1.75; 3.75; 2.95)$ ближе к точке (2; 4; 3), чем P_0, и лучше, чем значение, заданное в примере из раздела 2.3.1. Итерация, использующая уравнения (2.32), генерирует последовательность $\{P_k\}$, которая сходится к (2; 4; 3).

Принимая во внимание примеры раздела 2.3.1, необходимо иметь некоторый критерий, определяющий, будет ли сходиться итерация Якоби. Поэтому дадим следующее определение.

Определение. Говорят, что матрица A размера $N \times N$ называется *строго диагонально доминирующей*, если выполняется условие, что

$$| a_{kk} | > \sum_{\substack{j=1 \\ j \neq k}}^{N} | a_{kj} | \text{ для } k = 1, 2, \ldots, N. \qquad (2.33)$$

Это означает, что в каждой строке матрицы величина элемента на главной диагонали должна превышать сумму величин всех остальных элементов в строке. Матрица коэффициентов линейной системы (2.26) строго диагонально доминирующая, потому что:

в строке 1: $|4| > |–1| + |1|$,
в строке 2: $|–8| > |4| + |1|$,
в строке 3: $|5| > |–2| + |1|$.

Все строки удовлетворяют соотношению (2.33) определения; следовательно, матрица коэффициентов **A** линейной системы (2.26) строго диагонально доминирующая.

Матрица коэффициентов **A** линейной системы (2.29) не строго диагонально доминирующая, так как

в строке 1: $|–2| < |1| + |5|$,
в строке 2: $|–8| > |4| + |1|$,
в строке 3: $|1| < |4| + |–1|$.

Строки 1 и 3 не удовлетворяют соотношению (2.33) в определении, поэтому матрица коэффициентов **A** для линейной системы (2.29) не строго диагонально доминирующая.

А сейчас обобщим итерационные процессы Якоби и Гаусса–Зейделя. Предположим, что задана линейная система

$$\begin{array}{llll} a_{11}x_1 + a_{12}x_2 & +\ldots + a_{1j}x_j + \ldots + & a_{1N}x_N & = b_1 \\ a_{21}x_1 + a_{22}x_2 & +\ldots + a_{2j}x_j + \ldots + & a_{2N}x_N & = b_2 \\ \vdots & \vdots & \vdots & \vdots \\ a_{j1}x_1 + a_{j2}x_2 & +\ldots + a_{jj}x_j + \ldots + & a_{jN}x_N & = b_j \\ \vdots & \vdots & \vdots & \vdots \\ a_{N1}x_1 + a_{N2}x_2 & +\ldots + a_{Nj}x_j + \ldots + & a_{NN}x_N & = b_N. \end{array} \qquad (2.34)$$

Пусть *k-я* точка имеет вид $P_k = (x_1^{(k)}, x_2^{(k)}, ..., x_j^{(k)}, ..., x_N^{(k)})$. Следующая точка, (*k* + 1)-я, – $P_{k+1} = (x_1^{(k+1)}, x_2^{(k+1)}, ..., x_j^{(k+1)}, ..., x_N^{(k+1)})$. Верхний индекс (*k*) координат точки P_k предоставляет возможность устанавливать координаты, которые относятся к этой точке. Итерационная формула использует *j*-ю строку (2.34), чтобы выразить $x_j^{(k+1)}$ в терминах линейной комбинации предыдущих значений $x_1^{(k)}, x_2^{(k)}, ..., x_j^{(k)}, ..., x_N^{(k)}$.

Итерация Якоби:

$$x_j^{(k+1)} = \frac{b_j - a_{j1}x_1^{(k)} - \ldots - a_{jj-1}x_{j-1}^{(k)} - a_{jj+1}x_{j+1}^{(k)} - \ldots - a_{jN}x_N^{(k)}}{a_{jj}} \qquad (2.35)$$

для *j* = 1, 2, ..., *N*.

Итерация Якоби использует все старые координаты, чтобы получить все новые координаты, тогда как итерация Гаусса–Зейделя использует новые координаты по мере их появления.

Итерация Гаусса–Зейделя:

$$x_j^{(k+1)} = \frac{b_j - a_{j1}x_1^{(k+1)} - \ldots - a_{jj-1}x_{j-1}^{(k+1)} - a_{jj+1}x_{j+1}^{(k)} - \ldots - a_{jN}x_N^{(k)}}{a_{jj}} \qquad (2.36)$$

для $j = 1, 2, \ldots, N$.

В следующей теореме приводятся эффективные условия для сходимости итерации Якоби.

Теорема (итерация Якоби). Предположим, что **A** – строго диагонально доминирующая матрица. Тогда **Ax** = **b** имеет единственное решение $x = P$. Итерационная формула (2.35) порождает последовательность векторов $\{P_k\}$, которые сходятся к P для любого выбора начального вектора P_0.

Можно доказать, что метод Гаусса–Зейделя также будет сходиться, когда **A** – строго диагонально доминирующая матрица. Во многих случаях метод Гаусса–Зейделя сходится быстрее, чем метод Якоби, поэтому предпочитают обычно его (*см.* примеры из раздела 2.3.1). Важно понять, что достаточно небольшой модификации формулы (2.35), чтобы получить формулу (2.36). В некоторых случаях метод Якоби будет сходиться даже несмотря на то, что метод Гаусса–Зейделя не сходится.

3. ВЫЧИСЛЕНИЯ В СРЕДЕ *MATLAB*

3.1. Простейшие вычисления

Перед началом работы в пакете *MatLab* следует понимать, что основным видом данных, который используется в *MatLab* (сокр. от англ. *Matrix Laboratory* – дословно «Матричная лаборатория»), являются матрицы. Даже общепринятые скалярные переменные рассматриваются как матрицы размерности 1×1.

По умолчанию после запуска пакета *MatLab* на экране появляется комбинированное окно, включающее четыре наиболее важные панели – *Command Window* (Окно Команд), *Command History* (История Команд), *Workspace* (Рабочее пространство) и *Current Folder* (Текущая Папка), также окно снабжено следующими элементами – меню, панель с кнопками, строки состояния (рис. 3.1). Самой используемой панелью является *Command Window*. В ней набираются команды пользователя, подлежащие немедленному исполнению. Здесь же выдаются результаты исполняемых команд. Символ >> обозначает приглашение к вводу командной строки; набор любой команды или выражения заканчиваться нажатием на <Enter> (длинные команды, не помещающиеся целиком в одной строке, можно продолжать в следующей строке, используя условный знак переноса в виде трех подряд идущих точек (…), затем надо нажать комбинацию клавиш <Shift>+<Enter>).

Вычислим выражение 6·6. В командном окне будет отображаться следующее:

```
>> 6*6
ans =
   36
>>
```

Программа вычислила произведение, затем записала результат в системную переменную ans, вывела ее значение в командное окно и показала готовность к выполнению следующей команды символом >>.

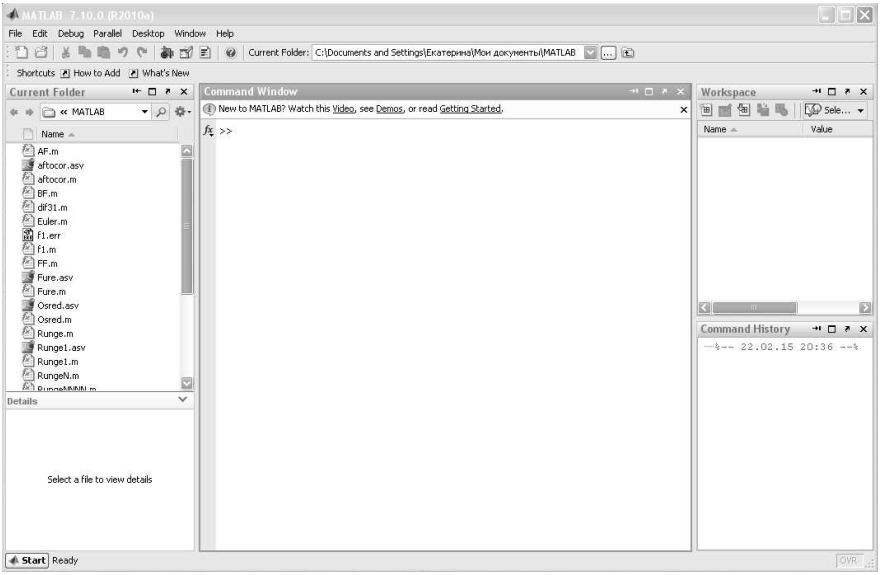


Рис. 3.1. Окно *MatLab*.

В *MatLab* предусмотрена возможность работы с переменными. Для того чтобы присвоить, например, переменной *m* значение 6.35, достаточно набрать в командной строке *m* = 6.35. При этом *MatLab* сразу же выведет значение *m*

```
>> m = 6.35

m =
   6.3500
>>
```

Часто не очень удобно после каждого присвоения получать еще и результат. Поэтому предусмотрена возможность завершать оператор присвоения точкой с запятой (;) для подавления вывода результатов в командное окно.

Значения всех промежуточных переменных, использованных в многошаговых вычислениях, *MatLab* запоминает в рабочем пространстве (*Workspace*). Переменные идентифицируются по следующим правилам: можно использовать латинские буквы, цифры и символ подчеркивания; большие и малые буквы в именах различаются;

имя должно начинаться с буквы; длина имени может быть любой, но первый 31 символ должен обеспечивать уникальность имен.

Числовые данные, с которыми оперирует *MatLab*, в памяти компьютера представлены в формате *double*. Это означает, что каждое вещественное число занимает 8 байт в оперативной памяти и принимает по модулю значения из диапазона $[10^{-308}, 10^{+308}]$. Изменить формат данных можно в командной строке следующим образом:

```
>> format short
>> x= 2^0.5   % ^ – оператор возведения в степень
x =
   1.4142
>> format hex
>> x=sqrt(2)    % sqrt – функция извлечения квадратного корня
x =
  3ff6a09e667f3bcd
```

В основном, используются следующие форматы: *short* (короткое 5-значное), *long* (длинное 15-значное), *hex* (шестнадцатеричное), *bank* (доллары и центы), *shortE* и *longE* (экспоненциальное), *rational* (отношение целых чисел).

Установка формата вывода данных на экран (т. е. внешний вид числа) производится в подпункте *Preference* пункта *File* меню. В списке левой панели выбирается пункт *Command Window* и задается формат в раскрывающемся списке *Numeric format* панели *Text display*. В пункте *Font* можно выбрать гарнитуру и параметры шрифта для изменения визуального отображения данных.

При работе с достаточно большим количеством переменных необходимо знать, какие переменные уже использованы, а какие нет. Помимо окна *Workspace* для этой цели служат также команды *who* и *whos*. Первая команда выводит только список имен переменных, а вторая сообщает более подробную информацию об именах переменных (*Name*), их размерности (*Size*), количестве занятых байтов в оперативной памяти (*Bytes*), классе объектов, представляющих соответствующий тип данных (*Class*), и атрибутах вывода данных (*Attributes*):

```
>> who
Your variables are:
ans  m   x
>> whos
```

```
Name      Size          Bytes  Class     Attributes
ans       1x1               8  double
m         1x1               8  double
x         1x1               8  double
```

MatLab снабжен множеством встроенных элементарных функций (тригонометрических, гиперболических, экспоненциальных и логарифмических, а также функций для работы с комплексными числами и для округления различными способами), например, *см.* табл. 3.1.

Таблица 3.1

Математические функции

Тип	Обозначение
Тригонометрические	cos, cot, csc, sec, sin, tan
Обратные тригонометрические	acos, acot, acsc, asec, asin, atan, atan2
Гиперболические	cosh, coth, csch, sech, sinh, tanh
Обратные гиперболические	acosh, acoth, acsch, asech, asinh, atanh
Экспонента, логарифмы, корень	exp, log, log2, log10, sqrt
Округления	ceil, fix, floor, round
Наибольший общий делитель	gcd
Наименьшее общее кратное	lcm
Модуль числа	abs
Знак числа	sign
Остаток от деления с учетом знака	mod
Остаток от деления	rem

При использовании встроенных функций аргументы заключаются в круглые скобки, а имена функций набираются строчными буквами. Для изменения порядка выполнения арифметических операций следует использовать круглые скобки.

Имена и значения переменных рабочего пространства можно запомнить в файле либо через команды главного меню *File* → *Save Workspace As*, либо через командное окно:

```
>> save  qq      % qq – имя файла
```

MatLab добавит к имени файла расширение mat и запомнит все переменные и их значения в файле qq.mat. В начале следующего сеанса достаточно выполнить команду:

```
>> load qq
```

Можно также открыть сохраненный ранее файл при помощи подпункта *Open* меню *File*.

3.2. Работа с массивами

Все данные в *MatLab* представляются в виде массивов. Массив – упорядоченная, пронумерованная совокупность однородных данных. У массива должно быть имя. Доступ к элементам массива осуществляется при помощи индекса. В *MatLab* нумерация элементов массивов начинается с единицы.

Значения элементов массива вводятся в командной строке в квадратных скобках:

```
>> a=[1 2 3]
a =
    1    2    3
>> b =[1; 2; 3]
b =
    1
    2
    3
>> c=[1 2 3; 4 5 6]
c =
    1    2    3
    4    5    6
```

Массив *a* называется вектор-строка, массив b – вектор-столбец, массив *c* имеет размер 2×3, т. е. две строки и три столбца.

Узнать размер массива можно при помощи встроенной функции:

```
>> size (c)
ans =
    2    3
```

Из нескольких вектор-столбцов можно составить один, используя квадратные скобки и разделяя исходные вектор-столбцы точкой с запятой:

```
>> v1 = [1; 2];
>> v2 = [3; 4];
>> v = [v1; v2]
v =
    1
    2
    3
    4
```

Для сцепления вектор-строк также применяются квадратные скобки, но сцепляемые вектор-строки отделяются пробелами или запятыми.

Доступ к элементам одномерного массива (вектор-строка, вектор-столбец) осуществляется при помощи индекса, заключаемого в круглые скобки после имени массива. Доступ к элементам двумерного массива осуществляется при помощи двух индексов – номера строки и столбца, заключенных также в круглые скобки, например, *c*(*k*, 3) определяет третий элемент *k*-ой строки, *c*(: ,3) – весь третий столбец.

Например. Пусть есть вектор:

```
>> v = [2  4  8  16  32];
```

Для вывода его второго значения, используется индексация:

```
>> v(2)
ans =
   4
```

MatLab предоставляет удобный способ обращения к блокам последовательно расположенных элементов вектора. Для этого служит индексация при помощи знака двоеточия (:). Например, пусть есть вектор-строка *X* из десяти элементов, и требуется заменить нулями элементы со шестого по десятый.

```
>> X = [200 220 250 250 500 500 250 240 230 200];
>> X(6:10) = 0
X =
  200   220   250   250   500     0     0     0     0     0
```

При формировании векторов можно пользоваться списками *i*:*k* и *i*:*j*:*k*. В первом варианте понимается, что «от *i* до *k* с шагом 1» и во втором – то же, но с шагом *j*, например:

```
>> X1=[0:5]
X1 =
    0     1     2     3     4     5
>> X2=[0:0.5:2]
X2 =
         0    0.5000    1.0000    1.5000    2.0000
```

Для работы с векторами предусмотрены некоторые функции, основные из них перечислены в табл. 3.2.

Таблица 3.2

Встроенные функции для работы с векторами

Написание	Назначение	Пример использования
prod( )	перемножение элементов вектора или вектора-строки	>> v=[1; 2; 3; 4; 5]; >> prod(v) ans = 120
length( )	определяет число элементов в векторе	>> length(v) ans = 5
sum( )	суммирует элементы вектора	>> sum(v) ans = 15 >> vv=[1 2; 2 3; 3 4; 4 5; 5 6]; >> sum(vv) ans = 15 20
mean( )	вычисляет среднее значение	>> mean(v) ans = 3 >> mean(vv) ans = 3 4
max( )	находит максимальное значение	>> max(v) ans = 5 >> max(vv) ans = 5 6
min( )	находит минимальное значение	>> min(v) ans = 1
sort( )	упорядочивает элементы по возрастанию	>> v1 = [8 9 4 1 2 3 5]; >> sort(v1) ans = 1 2 3 4 5 8 9

Следует различать поэлементные операции с массивами и операции над матрицами по правилам линейной алгебры (для массивов перед знаком операции ставят точку).

Например, пусть есть два вектора-строки:

```
>> v1=[2 3 -8 4];
>> v2 = [-1 2 1 -2];
>> v1v2=v1 .* v2
```

```
v1v2 =
   -2     6    -8    -8
```

Операция .* приводит к поэлементному умножению векторов одинаковой длины. В результате получаем вектор v1v2 с элементами, равными произведению соответствующих элементов исходных векторов.

При помощи .^ осуществляется поэлементное возведение в степень, ./ – поэлементное деление, .\ – обратное деление.

По правилам линейной алгебры умножение матрицы на матрицу производится следующим образом:

```
>> v1=[2 3 -8 4];
>> v2 = [-1 2 1 -2];
>> v3 = v2'   % транспонируем вектор-строку в вектор-столбец
v3 =
   -1
    2
    1
   -2
>> v1v3=v1*v3
v1v3 =
  -12
```

3.3. Графика

3.3.1. Диаграмма

Отображение вектора в виде столбчатой диаграммы осуществляется функцией bar (рис. 3.2).

Пример:

```
>> d = [9.54 6.50 9.17 6.54 1.66 0.56 6.94 3.08 5.06 0.22];
>> bar(d)
```

Разметку горизонтальной оси можно задать вектором с возрастающими значениями. В этом случае первый аргумент является вектором с координатами точек разметки, а второй – вектором значений (необходимо, чтобы длины этих векторов совпадали). Удобно использовать этот способ построения диаграммы для отображения значений элементов векторов в зависимости от времени.

Пример:

```
>> t = [0.0 0.1 0.3 0.5 0.7 0.9 1.1 1.3];
>> d = [9.54 6.50 9.17 6.54 1.66 0.56 6.94 3.08];
>> bar(t, d)
```

а) *б*)

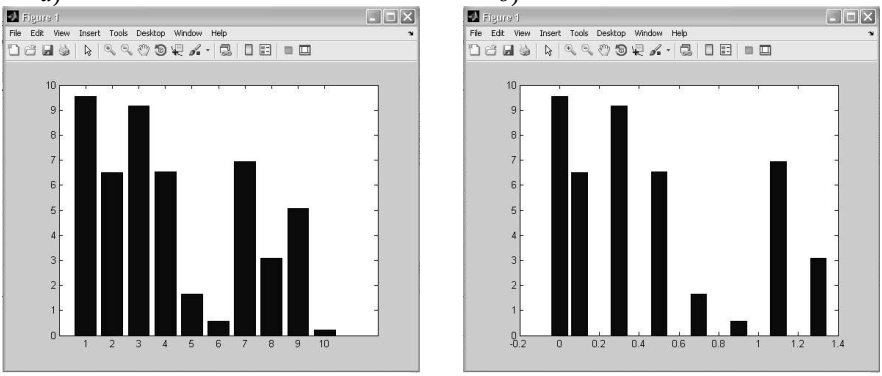


Рис. 3.2. Пример использования функции bar без задания горизонтальной оси (*а*) и с временной горизонтальной осью (*б*).

По умолчанию ширина столбца равна 0.8; изменить ширину столбцов можно с помощью третьего дополнительного аргумента функции. Диаграмма без промежутков получается, если установить ширину равную единице: >> bar(t, d, 1)

3.3.2. Круговая диаграмма

Если требуется оценить вклад каждого из элементов в общую сумму, то удобно строить круговую диаграмму функцией pie (рис. 3.3).

Пример:

```
>> d = [3 4 9 10 4];
>> pie(d)
```

При необходимости отодвинуть от круга диаграммы сектор, соответствующий некоторому элементу, задается второй аргумент вектор, состоящий их единиц и нулей: единица соответствует номеру отделяемой части.

Пример:

```
>> d = [3 4 9 10 4];
>> sector = [0 0 0 1 0];
>> pie(d, sector)
```

а) *б*)

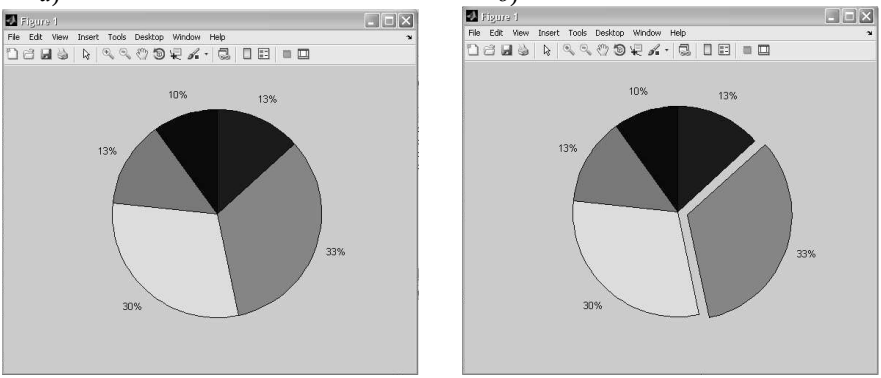


Рис. 3.3. Пример использования функции *pie* с различным числом параметров.

3.3.3. Трехмерная диаграмма

Визуализация векторных данных может быть осуществлена при помощи функций *pie*3 и *bar*3, которые строят трехмерные круговые и столбчатые диаграммы (рис. 3.4).

Пример:

```
>> d = [3 4 9 10 4];
>> sector = [0 0 0 1 0];
>> bar3(d)
>> pie3(d, sector)
```

3.3.4. Гистограммы

Для получения наглядного представления о распределении данных (т. е. о том, сколько данных попало в тот или иной интервал) служит функция hist (рис. 3.5).

Пример:

```
>> d = randn(100, 1);
```

% вектор d размером 100 × 1 заполняется числами, распределенными по нормальному закону

```
>> hist(d)
```

Функция *hist* может содержать второй аргумент – число интервалов, на которое разбивается диапазон чисел (по умолчанию число интервалов равняется 10). Кроме того, вместо автоматического разбиения на равные интервалы можно использовать собственные, задав в качестве второго аргумента вектор, содержащий центры интервалов.

Например:

```
>> d = [0.3 0.6 0.9 1.1 1.4 1.7 2.0 3.0 4.0];
>> center = [0.6 1.5 3.0];
>> hist(d, center)
```

а) *б*)

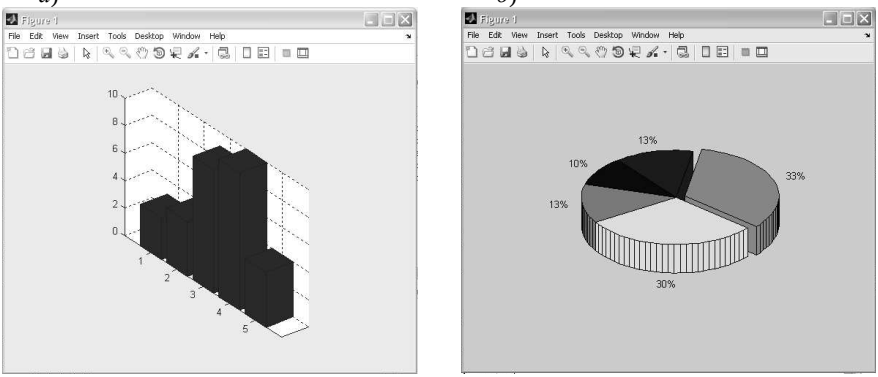


Рис. 3.4. Примеры использования функции bar3 (*а*) и pie3 (*б*).

а) *б*)

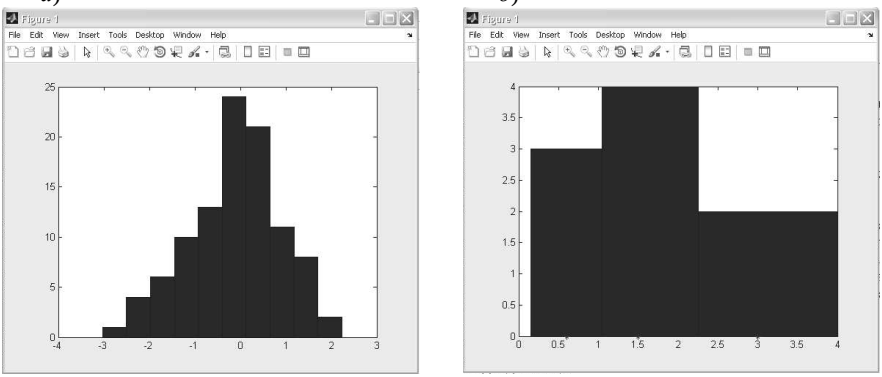


Рис. 3.5. Примеры использования функции *hist*: *а* – с аргументами по умолчанию, *б* – с заданными центрами интервалов.

3.3.5. Построение графиков функции одной переменной

Для того чтобы построить график функции $y = f(x)$, необходимо сформировать два вектора одинаковой размерности – вектор значений аргументов (x) и вектор соответствующих значений функции (y), и обратиться к функции *plot*.

Пример: построение графика функции $y = cos\,(x) \cdot e^x$

```
>> x = 0:0.1:6.28;
% изменение аргумента x от 0 до 6,28 с шагом 0,1
>> y = cos(x).*exp(x);
>> plot (x, y, ':g')
```

Дополнительный параметр в функции *plot* – аргумент в апострофах, задающий один из 8 предусмотренных цветов: y – желтый (от *yellow*); g – зеленый (от *green*); m – малиновый (от *magenta*); b – синий (от *blue*); c – циановый (от *cyan*); w – белый (от *white*); r – красный (от *red*); k – черный (от *black*).

Вид графика можно изменить, задав стиль линии и форму маркера, которым метятся точки. Стиль линии предусматривает выбор одной из четырех возможностей: сплошная линия – тире (–), пунктирная линия – двоеточие (:), штрихпунктирная линия – тире и точка (–.), штриховая линия – два тире (– –). Форма маркеров: . – жирная точка, o – кружок, x – косоугольный крестик, + – прямоугольный крестик, * – восьмиконечная снежинка, s – квадратик, d – ромбик, v – треугольник вершиной вниз, ^ – треугольник вершиной вверх, < – треугольник вершиной налево, > – треугольник вершиной направо, p – пятиконечная звезда, h – шестиконечная звезда. Управляющие символы, определяющие стиль линии, задаются в строке третьего параметра вместе с цветом. Порядок следования символов – любой.

Дополнительное оформление графика заключаются в возможности снабдить его заголовком (функция *title*), подписать оси (функции *xlabel*, *ylabel*), нанести координатную сетку (функция *grid on*) и разместить легенду (функция *legend*).

Пример: построение графика норм месячных температур воздуха для Санкт-Петербурга и Владивостока (рис. 3.6).

```
>> t = [1 2 3 4 5 6 7 8 9 10 11 12];
>> TSPb = [-7.8 -6.9 -2.2 4.0 10.9 15.6 17.7 16.2 11.1 5.7 0.1 -4.6];
>> TVladik = [-12.4 -8.3 -1.8 5.0 9.8 13.5 17.5 19.7 15.9 8.8 -0.9 -8.9];
>> plot (t, TSPb, 'k<-', t, TVladik, 'b>--')
```

```
>> grid on
>> title ('Температура по месяцам')
>> xlabel('Время (мес)')
>> ylabel('Температура (С)')
>> legend('Санкт-Петербург', 'Владивосток', 4)
```

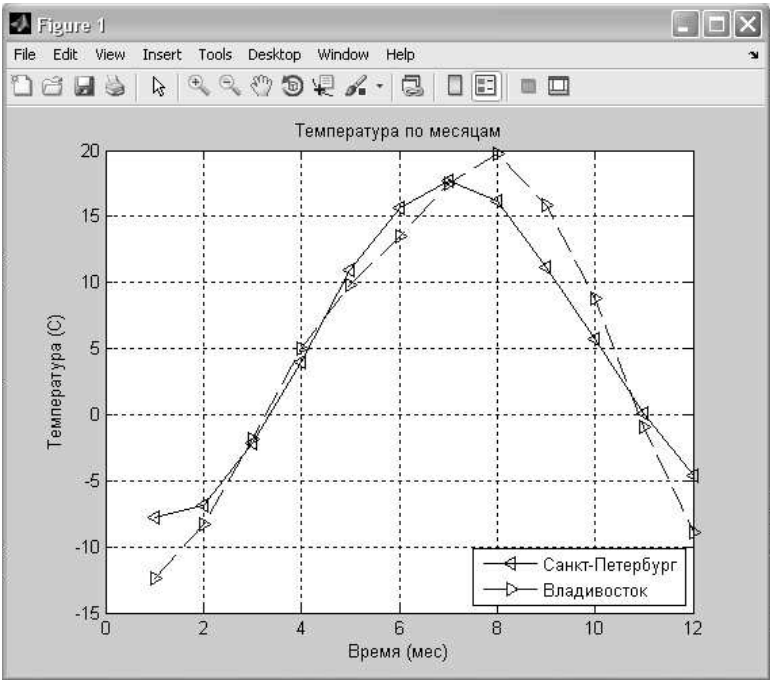


Рис. 3.6. Пример оформления графика.

В функции legend третий аргумент позволяет размещать легенду на усмотрение пользователя: -1 – легенда размещается вне поля графика, вверху справа; 0 – система выбирает лучшее место в поле графика, не перекрываемое данными; 1 – легенда размещается в правом верхнем углу (по умолчанию – там же); 2 – легенда размещается в левом верхнем углу поля графика; 3 – легенда размещается в правом нижнем углу поля графика; 4 – легенда размещается в левом нижнем углу поля графика.

Иногда требуется построить графики в логарифмическом и полулогарифмическом масштабах. Для этого служат функции:

loglog – логарифмический масштаб по двум осям;
semilogx – логарифмический масштаб по оси абсцисс;
semilogy – логарифмический масштаб по оси ординат.
Пример:

```
>> x = [0:0.01:6.28];
>> y1 = log (2*x);
>> y2 = cos (log(x));
>> semilogx(x, y1, x, y2)
```

3.3.6. Построение графиков нескольких функций

Для совмещения двух графиков в одном окне *MatLab* предлагает процедуру *plotyy*, которая производит двойную оцифровку осей. Для первой функции цифруются ось *x* внизу, ось *y* слева, а для второй функции ось *x* размечается вверху, ось *y* – справа. Цвет оцифровки при этом совпадает с цветом кривых.

Может оказаться, что в одном графическом окне необходимо отобразить большее число графиков. Тогда можно прибегнуть к функции *subplot* (рис. 3.7), которая позволяет разделить область рисования на несколько прямоугольных областей равного размера, расположенных подобно элементам матрицы:

```
>> subplot (row, col, cur);
```

Первые два аргумента задают количество рядов (*row*) и колонок (*col*). Третий параметр (*cur*) объявляет порядковый номер подобласти, в котором очередная функция *plot* будет строить свой график.

Пример.

```
>> x = [0:0.1:6.28];
>> y1 = sin (x);
>> y2 = cos (x);
>> y3 = tan (x);
>> y4 = cot (x);
>> subplot (2, 2, 1);
>> plot (x, y1);
>> title ('Синус')
```

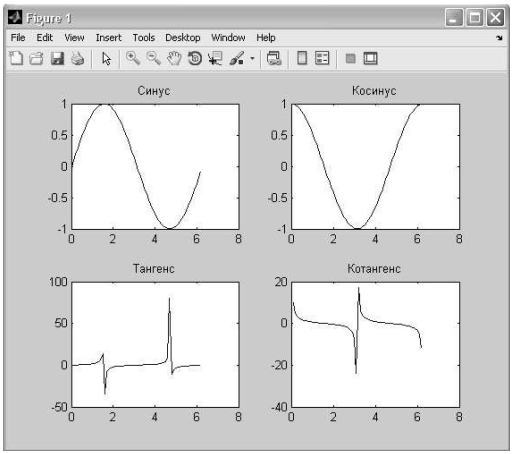


Рис. 3.7. Пример применения функции *subplot*.

```
>> subplot (2, 2, 2); plot (x, y2);
>> title ('Косинус')
>> subplot (2, 2, 3);  plot (x, y3);
>> title ('Тангенс')
>> subplot (2, 2, 4);  plot (x, y4);
>> title ('Котангенс')
```

3.3.7. Построение графиков функций двух переменных

Построение графика функции двух переменных включает два предварительных этапа:

1) разбиение области определения прямоугольной сеткой;

2) вычисление значений функции в точках пересечения линий сетки и запись их в матрицу.

Для генерации массивов сетки по координатам узлов используется функция *meshgrid*, для построения графика в виде каркасной сетки – функция *mesh*.

Пример: построение графика функции $z(x, y) = x + y$ (рис. 3.8).

```
>> [x, y] = meshgrid (0:0.1:1, 0:0.1:1);
>> z = x + y;
>> mesh(x, y, z)
```

а) *б*)

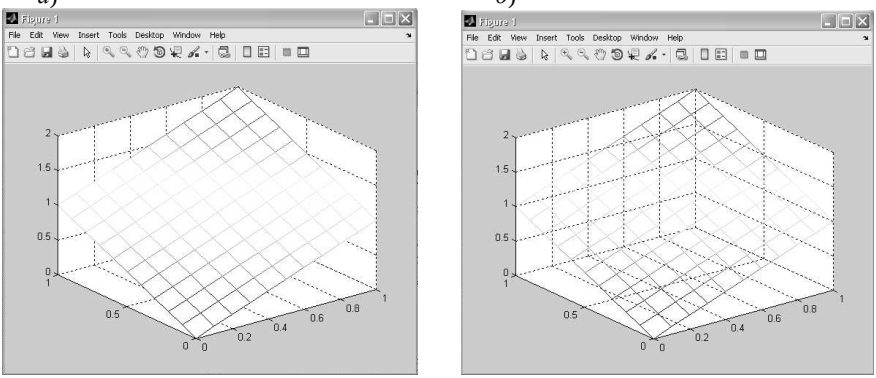


Рис. 3.8. Пример применения функции *mesh* с непрозрачной (*а*) и прозрачной (*б*) каркасной поверхностью.

С помощью команды *hidden off* можно сделать каркасную поверхность «прозрачной». Команда *hidden on* убирает невидимую часть.

Функция *surf* строит каркасную поверхность и заливает каждую клетку поверхности определенным цветом. Команда *shading flat* позволяет убрать каркасные линии. Для получения поверхности, плавно залитой цветом, служит команда *shading interp*. Команда *colorbar* позволяет вывести легенду графика, устанавливающую соответствие между цветом и значением функции. Последовательность действия команд *см*. на рис. 3.9.

```
>> surf (x, y, z)
>> shading flat
>> shading interp
>> colorbar
```

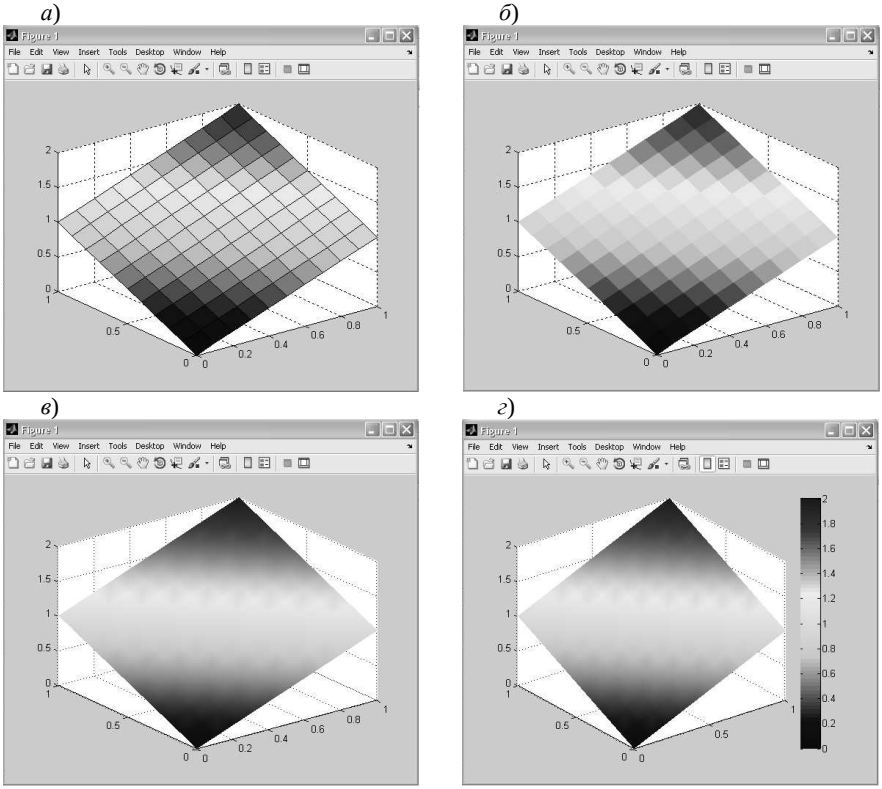


Рис. 3.9. Результаты выполнения функции *surf* (*а*) с последовательным применением команд *shading flat* (*б*), *shading interp* (*в*), *colorbar* (*г*).

Функции *meshc* и *surfc* позволяют получить представление о поведении функции в точках плоскости, так как размещают на плоскости *xy* линии уровня функции (рис. 3.10).

```
>> [x, y] = meshgrid (10:50, 10:50);
>> z = x.*2+y.*2;
>> meshc(x, y, z)
>> colorbar
>> surfc(x, y, z)
>> colorbar
```

а) *б*)

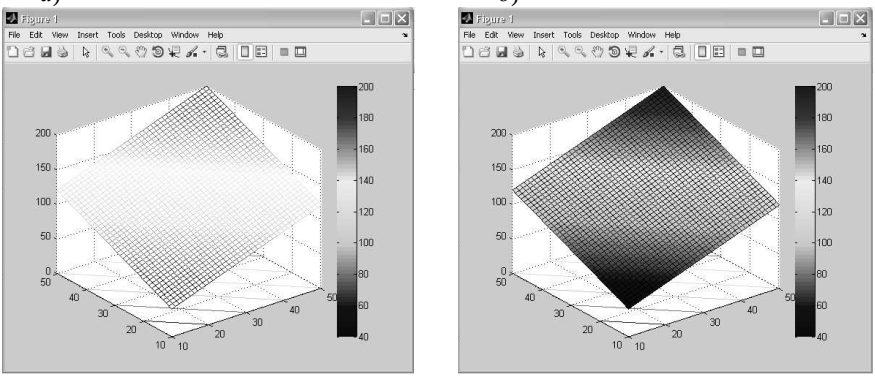


Рис. 3.10. Пример применения функций *meshc* (*а*) и *surfc* (*б*).

Функции *meshz* строит перпендикуляры к плоскости *xy* (рис. 3.11):

```
>> [x, y] = meshgrid (10:50, 10:50);
>> z = x.*2+y.*2;
>> meshz(x, y, z)
```

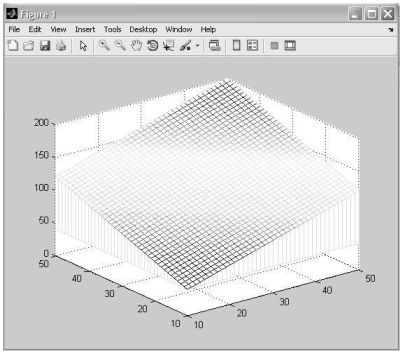


Рис. 3.11. Визуализация работы функции *meshz*.

Можно построить поверхность из линий уровня при помощи функции *contour*3. Имеется возможность задать четвертым аргументом либо число линий уровня, либо вектор, элементы которого равны значениям функции, отображаемой в виде линий уровня (рис. 3.12).

```
>> [x, y] = meshgrid (10:50, 10:50);
>> z = x.*2+y.*2;
>> contour3(x, y, z)   % рис. 3.12, а
>> L=[10:0.5:50];
>> contour3(x, y, z, L)   % рис. 3.12, б
>> colorbar
```

а) *б*)

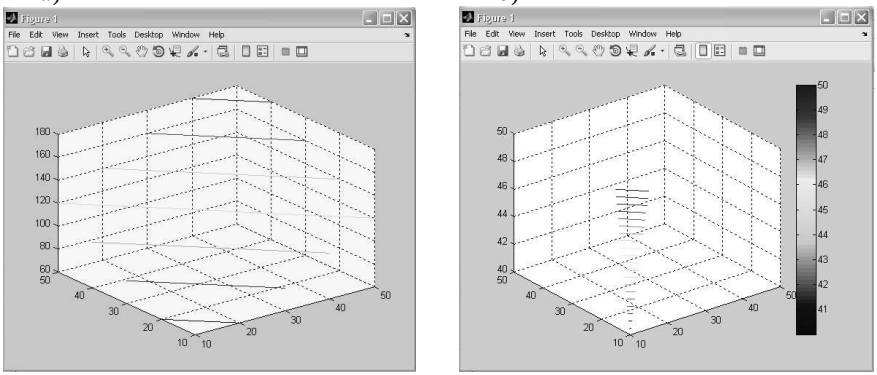


Рис. 3.12. Пример применения функции *contour*3 (*а*) с дополнительным аргументом (*б*).

3.3.8. Анимированные графики

В *Matlab* есть возможность следить за движением точки по заданной траектории при помощи анимированных графиков, строящихся функциями *comet* (для плоскости) и *comet3* (для трехмерного пространства).

Пример: построение траектории движения точки в течение 10 секунд, координаты которой меняются по законам $x(p) = \cos(p)$ $y(p) = \sin(p)$ (рис. 3.13, *а*).

```
>> p=[0:10:360]; % шаг управляет скоростью построения графика
>> x = cos(p);
>> y = sin(p);
>> comet (x, y)
```

Использование функции *comet* с одним аргументом приводит к построению динамически рисующегося графика значений элементов номера в зависимости от их номеров.

Функцию *comet* можно вызвать с третьим параметром, который задает длину хвоста (по умолчанию он равен 0.1).

Для построения траектории точки, перемещающейся в пространстве, используется функция *comet*3 (рис. 3.13, *б*):

```
>> p=[0:1:360];
>> x = cos(p);
>> y = sin(p);
>> comet3 (x, y, p)
```

а) *б*)

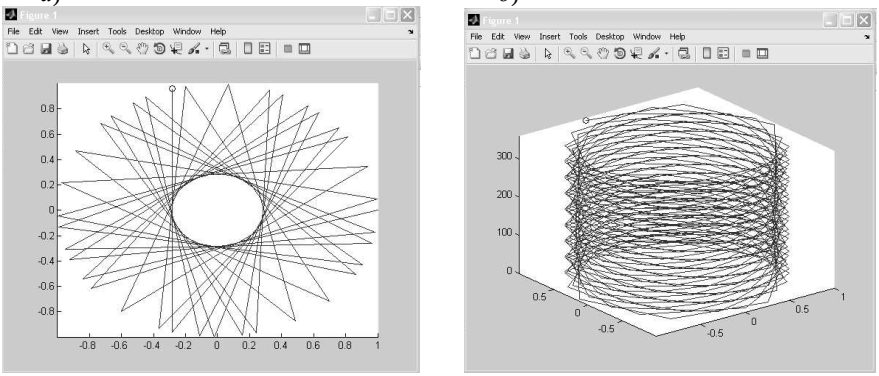


Рис. 3.13. Пример применения функций *comet* (*а*) и *comet*3 (*б*).

3.4. *M*-файлы

Для написания и реализации программ на языке *MatLab* служат *m*-файлы. Подготовленный и записанный на диск *m*-файл становится частью системы, и его можно вызывать как из командной строки, так и из другого *m*-файла. Есть два типа *m*-файлов: файлы-сценарии и файлы-функции.

Файл-сценарий, именуемый также *Script*-файлом, является просто записью серии команд без входных и выходных параметров. Он имеет следующую структуру:

```
% Основной комментарий
% Дополнительный комментарий
Тело файла с любыми выражениями
```

Важны следующие свойства файлов-сценариев:

– они не имеют входных и выходных аргументов;

– работают с данными из рабочей области;

– в процессе выполнения не компилируются;

– представляют собой зафиксированную в виде файла последовательность операций, полностью аналогичную той, что используется в сессии.

Основным комментарием является первая строка текстовых комментариев, а дополнительным – последующие строки. Основной комментарий выводится при выполнении команд *lookfor* и *help* имя_каталога. Полный комментарий выводится при выполнении команды *help* имя_файла.

M-файл-функция является типичным объектом языка программирования системы *MatLab*. Структура этого структурированного модуля с одним выходным параметром

function var = f_name(Список_параметров)

%Основной комментарий

%Дополнительный комментарий

Тело файла с любыми выражениями

var = выражение

M-файл-функция имеет следующие свойства:

– он начинается с объявления *function*, после которого указывается имя переменной *var* – выходного параметра, имя самой функции и список ее входных параметров;

– функция возвращает свое значение и может использоваться в виде name (Список_параметров) в математических выражениях;

– все переменные, имеющиеся в теле файла-функции, являются локальными, т. е. действуют только в пределах тела функции;

– файл-функция является самостоятельным программным модулем, который общается с другими модулями через свои входные и выходные параметры;

– правила вывода комментариев те же, что у файлов-сценариев;

– файл-функция служит средством расширения системы *MatLab*;

– при обнаружении файла-функции он компилируется и затем исполняется, а созданные машинные коды хранятся в рабочей области системы *MatLab*.

Последняя конструкция *var* = выражение вводится, если требуется, чтобы функция возвращала результат вычислений. Если выходных

параметров больше, то они указываются в квадратных скобках после слова *function*.

Любая серьезная программа имеет нелинейную структуру. Для создания таких программ необходимы специальные управляющие структуры.

1. Диалоговый ввод и вывод:

input ('Комментарий', 's') или var = input ('Введите var');

disp ('Комментарий') или disp (var).

В первых случаях функции *input* и *disp* используются для ввода/вывода произвольного строкового выражения, а во вторых – для ввода/вывода значения *var*.

2. Условный оператор:

if Условие
Инструкции_1
elself Условие
Инструкции_2
else
Инструкции_3
end

Пока Условие возвращает логическое значение 1 (т. е. «истина»), выполняются Инструкции, составляющие тело структуры *if ... end*. Инструкции в списке разделяются оператором , (запятая) или ; (точка с запятой). В Условии должны быть использованы следующие операторы отношения: ==, <, >, <=, >= или ~=. Все эти операторы представляют собой пары символов без пробелов между ними.

3. Циклы типа *for ... end* обычно используются для организации вычислений с заданным числом повторяющихся циклов. Конструкция такого цикла имеет следующий вид:

for var = Выражение, Инструкция, …, Инструкция end

Выражение чаще всего записывается в виде *s*:*d*:*e*, где *s* – начальное значение переменной цикла *var*, *d* – приращение этой переменной и *e* – конечное значение управляющей переменной, при достижении которого цикл завершается. Возможна и запись в виде *s*:*e* (в этом случае *d* = l). В примере иллюстрируется формирование двумерной матрицы:

for i=1:3
for j=l:3
A(i, j) = i+j;

```
end
end
```

Следует отметить, что формирование матриц с помощью оператора : (двоеточие) обычно занимает намного меньше времени, чем с помощью цикла.

4. Цикл типа *while* выполняется до тех пор, пока выполняется Условие:

```
while Условие
Инструкции
end
```

3.5. Примеры реализации численных методов в *MatLab*

3.5.1. Метод Эйлера

Метод Эйлера разберем на примерах следующих гидрологических моделей:

– модель склонового стока с сосредоточенными параметрами:

$$\tau dQ/dt + Q/k = \dot{X}\ , \qquad (3.1)$$

где τ – время релаксации речного бассейна, k – коэффициент, характеризующий потери на водосборе (аналог коэффициента стока) стока, $\dot{X}$ – интенсивность осадков, t – координата по времени;

– модель водоема с сосредоточенными параметрами:

$$dH/dt + k_{\text{м}} H = \dot{\xi}\,, \qquad (3.2)$$

где H – уровень водоема (средний по площади водоема), $k_{\text{м}}$ – морфометрический коэффициент, характеризующий форму чаши водоема, $\dot{\xi}$ – интенсивность внешнего воздействия на уровень водоема (притоки, оттоки, испарение, осадки и т. д.);

– модель кинематической волны:

$$\partial F / \partial t + a \partial F / \partial x = q \,. \qquad (3.3)$$

F – площадь живого сечения, a – скорость перемещения волны; q – внешнее воздействие, которое может быть представлено, например, боковым притоком со склонов, притоком (оттоком) воды в русло в результате взаимодействия с грунтовыми водами и т. д., x – координат по расстоянию.

Рассматриваемые модели переписываем следующим образом.

Модель водоема:

$$H_{i+1} = H_i + \Delta t (X_i - k_{\text{морф}} H_i) \,; \qquad (3.4)$$

модель склонового стока с сосредоточенными параметрами:

$$Q_{i+1} = Q_i + \Delta t (kX_i - Q_i) / \tau \,; \qquad (3.5)$$

модель кинематической волны:

$$F_{i+1,\,j} = F_{i,\,j} - a(\Delta t / \Delta x)(F_{i,\,j} - F_{i,\,j-1}) + q_{i,\,j} \Delta t \,, \qquad (3.6)$$

где Δt – шаг по времени, Δx – шаг по пространственной координате.

Для модели кинематической волны метод Эйлера будет работать следующем образом: расход воды в начальный момент времени и на границе известны (на внешних узлах сетки), находим решение во внутреннем узле по алгоритму, передвигаясь последовательно от узла к узлу, находим F во всех узлах сетки (рис. 3.14). Для устойчивости решения необходимо соблюдение определенного соотношения шагов по времени Δt и по продольной координате Δx: $a \cdot \Delta t / \Delta x \le 1$.

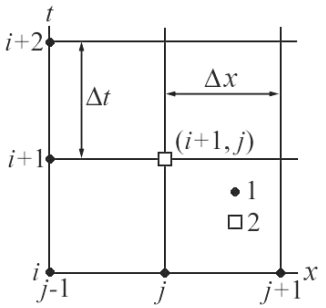


Рис. 3.14. Прямоугольная сетка: 1 – внешний узел, 2 – внутренний узел.

Для реализации моделей водоема (3.2) и склонового стока с сосредоточенными параметрами (3.1)

создано 3 *m*-файла: по одному *m*-файлу на каждую модель, откуда вызывается функция, реализующая алгоритм Эйлера по формулам (3.4), (3.5):

```
function [X, Y] = Euler(Q0, t1, N, p, ii)
```

Входные аргументы:

Q0 – начальное значение (для модели водоема – уровень воды (м), для модели склонового стока – расход воды в замыкающем створе ($м^3$/ч)),

t1 и *N* характеризуют шаг дискретизации *dt = t1/N*,

p – вектор интенсивности осадков (м/ч),

ii – аргумент идентифицирующий модель (*ii* = 1 – модель водоема, *ii* = 2 – модель водосбора).

Выходные аргументы:

X – значения характеризующие водный объект (уровень воды, расход воды),

Y – время (ч).

```
function F = f1( t, Q, p, kmorf)
% Правая часть модели водоема
F = p - kmorf*Q;
end

function F = f( t, Q, p, kst, tau)
% Правая часть модели склонового стока
F = (kst*p-Q)/tau;
end

function [X, Y] = Euler(Q0, t1, N, p, ii)
dt = t1/N;
t(1) = 0;
Q(1) = Q0;
N1 = length(p)+1;
p(N1:N) = 0;
% Модель водоема
if (ii == 1)
kmorf = input('Введите морфометрический коэффициент: ');
```

```
for i = 1:N
   t(i+1) = t(1)+dt*i;
   Q(i+1) = Q(i)+dt*f1(t(i), Q(i), p(i), kmorf); % обращение к правой
части модели
   if Q(i+1) < Q0
      Q(i+1) = Q0;
   end;
end;
end;
% Модель склонового стока
if  (ii == 2)
kst = input('Введите коэффициент стока: ');
tau = input('Введите время добегания: ');
for i = 1:N
   t(i+1) = t(1)+dt*i;
   Q(i+1) = Q(i)+dt*f(t(i), Q(i), p(i), kst, tau); % обращение к правой
части модели
   if Q(i+1) < Q0
      Q(i+1) = Q0;
   end;
end;
end;
X=t;
Y=Q;
end
```

Результат моделирования моделей водоема и склонового стока выводится в виде графиков, функцией *plot*(*t*, *Q*).

Пример.

Модель водоема: $k_{морф}$ (морфометрический коэффициент) = 0.15; $\dot{X}$ (интенсивность осадков) = 0.2 0.3 0.5 1 0.2 (м/ч); N = 10; $t1$ = 10; H_0 (начальный уровень) = 1 м.

Модель склонового стока: k_{st} (коэффициент стока) = 0.75; τ (время добегания) = 2 (ч); N = 10; t1 = 10; Q_0 (начальный расход) = 1.0 ($м^3$/ч); $\dot{X}$ = 2 3 4 7 10 6 2 (м/ч).

```
% Метод Эйлера для модели водоема
```

```
>> Q0 = 1.0;
>> t1 = 10;
>> N = 10;
>> p = [0.2 0.3 0.5 1 0.2];
>> [x,y]=Euler(Q0, t1, N, p, 1);
>> plot(x,y)
>> xlabel('t');
>> ylabel('H')

% Метод Эйлера для модели склонового стока
>> Q0 = 1.0;
>> t1 = 10;
>> N = 10;
>> p = [2 3 4 7 10 6 2];
>> [x,y]=Euler(Q0, t1, N, p, 2);
>> plot(x,y)
>> xlabel('t');
>> ylabel('Q')
```

а) *б*)

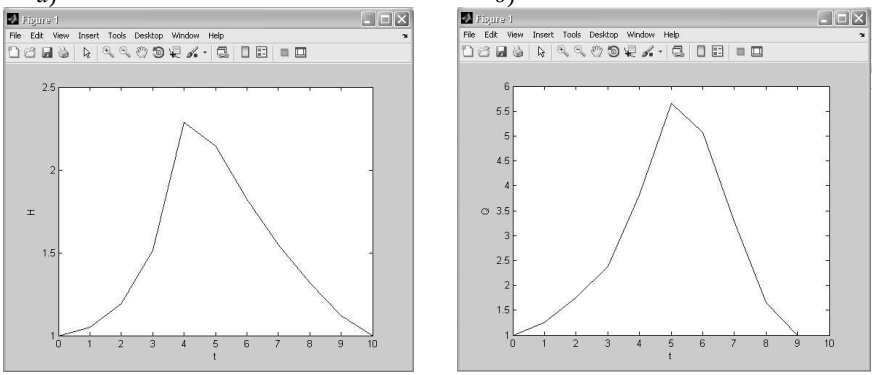


Рис. 3.15. Результат моделирования моделей водоема (*а*) и склонового стока (*б*) методом Эйлера.

Для визуализации модели руслового стока с распределенным притоком (модель кинематической волны) необходимы входные параметры: скорость перемещения волны, величина притока, начальные и

граничные условия, шаги дискретизации по расстоянию и времени. Эта модель реализуется алгоритмом Эйлера по формуле (3.6). Результат представлен в виде двухмерного графика при помощи функции *surf*(*T*,*X*,*Q*).

Пример.

Q_0 = 1,0 (м^3/ч); граничные условия = 2 3 4 8 5 3 2 (м^3/ч); *a* (скорость перемещения паводочной волны) = 0,8 (м/ч); шаги дискретизации по длине русла и времени соответственно – 10 и 10; *q* (боковой приток) = 0,03 (м^3/ч).

```
% m-сценарий
nachysl = 1.0; % начальные условия
granysl = [2 3 4 8 5 3 2]; % граничные условия
aa = 0.8;    % скорость перемещения паводочной волны
dt = 10;
dx = 10;
q = 0.03;
Ng = length(granysl);
N = 10;
T=1:N;
X=1:N;
Uys = aa*dt/dx;
% Проверка условия устойчивости
if Uys <= 1
for L=1:N
  Q(L, 1) = nachysl;
   for t=1:N
      if t < Ng
         Q(1,t) = granysl(t);
    else
    Q(1,t) = nachysl;
 end;
end;
end;
for t=2:N
   for L=2:N
      Q(L,t) = Q(L,t-1)-aa*dt/dx*(Q(L,t-1)-Q(L-1,t-1))+q*dt;
end;
end;
```

```
surf (T, X, Q);
xlabel('Time');
ylabel ('X');
else
   msgbox ('Uys > 1', 'Error');
end
```

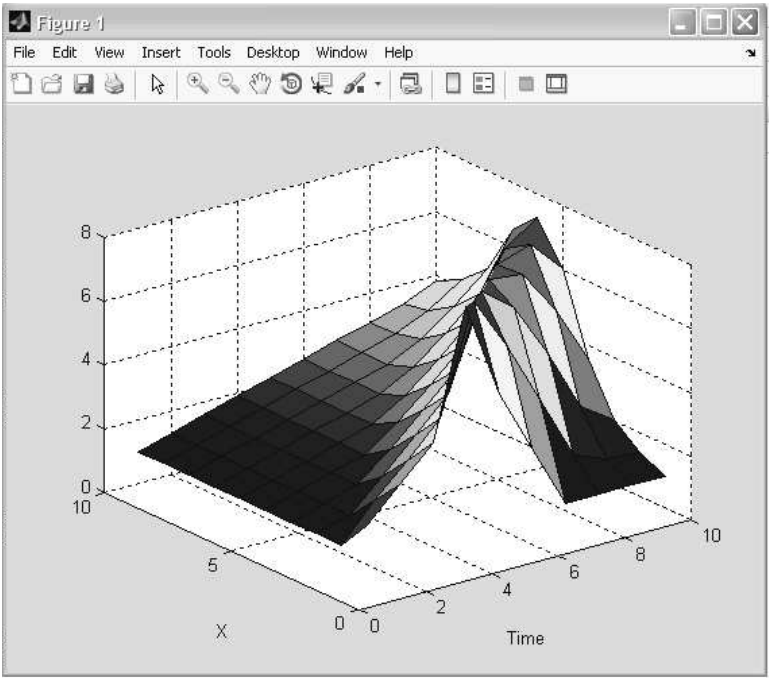


Рис. 3.16. Результат моделирования модели руслового стока методом Эйлера.

3.5.2. Метод Рунге–Кутты

Для решения систем обыкновенных дифференциальных уравнений (ОДУ) в *Matlab* реализованы различные методы. Их реализации названы решателями (*solver*) ОДУ. Рассмотрим некоторые из них:

ode45 – одношаговые явные методы Рунге–Кутты 4-го и 5-го порядка. Это классический метод (в модификации Дорманда и Принца),

рекомендуемый для начальной пробы решения. Во многих случаях он дает хорошие результаты;

ode23 – одношаговые явные методы Рунге–Кутты 2-го и 4-го порядка (в модификации Богацки и Шампина). При умеренной жесткости системы ОДУ и низких требованиях к точности этот метод может дать выигрыш в скорости решения;

ode113 – многошаговый метод Адамса–Башворта–Мултона переменного порядка. Это адаптивный метод, который может обеспечить высокую точность решения;

ode23tb – неявный метод Рунге–Кутты в начале решения и метод, использующий формулы обратного дифференцирования 2-го порядка в последующем. Несмотря на сравнительно низкую точность, этот метод может оказаться более эффективным, чем *ode15s*;

ode15s – многошаговый метод переменного порядка (от 1 до 5, по умолчанию 5), использующий формулы численного дифференцирования. Это адаптивный метод, его стоит применять, если решатель *ode45* не обеспечивает решения;

ode23s – одношаговый метод, использующий модифицированную формулу Розенброка 2-го порядка. Может обеспечить высокую скорость вычислений при низкой точности решения жесткой системы дифференциальных уравнений;

ode23t – метод трапеций с интерполяцией. Этот метод дает хорошие результаты при решении задач, описывающих колебательные системы с почти гармоническим выходным сигналом.

Все решатели могут решать системы уравнений явного вида $y' = F(t, y)$. Решатели *ode15s* и *ode23t* способны найти корни дифференциально-алгебраических уравнений $M(t)y' = F(t, y)$, где M называется матрицей массы. Решатели *ode15s*, *ode23s*, *ode23t* и *ode23tb* могут решать уравнения неявного вида $M(t, y)\, y' = F(t, y)$. Все решатели, за исключением *ode23s*, который требует постоянства матрицы массы, могут находить корни матричного уравнения вида $M(t, y)\, y' = F(t, y)$. Решатели *ode23tb*, *ode23s* служат для решения жестких дифференциальных уравнений , *ode15s* – жестких дифференциальных и дифференциально-алгебраических уравнений, *ode23t* – умеренно жестких дифференциальных и дифференциально-алгебраических уравнений.

В описанных функциях для решения дифференциальных уравнений (и систем) приняты следующие обозначения и правила:

options – аргумент, создаваемый функцией *odeset* (и функция – *odeget* – позволяет вывести параметры, установленные по умолчанию или с помощью функции *odeset*);

tspan – вектор, определяющий интервал интегрирования [*t*0 *tfinal*]. Для получения решений в конкретные моменты времени *t*0, *t*l, ..., *tfinal* (расположенные в порядке уменьшения или увеличения) нужно использовать *tspan* = [*t*0 *t*l ... *tfinal*];

*y*0 – вектор начальных условий;

pi, *p*2, ... – произвольные параметры, передаваемые в функцию *F*;

T, *Y* – матрица решений *Y*, где каждая строка соответствует времени, возвращенном в векторе-столбце *T*.

Описание функций для решения систем дифференциальных уравнений:

[*T*, *Y*] = *solver* (*@F*,*tspan*, *y*0) – где вместо *solver* подставляем имя конкретного решателя – интегрирует систему дифференциальных уравнений вида *y*' = *F* (*t*, *y*) на интервале *tspan* с начальными условиями *y*0. *@F* – дескриптор ODE-функции. Каждая строка в массиве решений *Y* соответствует значению времени, возвращаемому в векторе-столбце *T*;

[*T*, *Y*] = *solver* (*@F*, *tspan*, *y*0, *options*) – дает решение, подобное описанному выше, но с параметрами, определяемыми значениями аргумента *options*, созданного функцией *odeset*. Обычно используемые параметры включают допустимое значение относительной погрешности *RelTol* (по умолчанию 1e-3) и вектор допустимых значений абсолютной погрешности *AbsTol* (все компоненты по умолчанию равны 1e-6);

[*T*, *Y*] = *solver* (*@F*, *tspan* ,*y*0, *options*, *p1*, *p2*...) – дает решение, подобное описанному выше, передавая дополнительные параметры *p1*, *p2*,... в *m*-файл *F* всякий раз, когда он вызывается. Используйте *options* = [], если никакие параметры не задаются;

[*T*, *Y*, *TE*, *YE*, *IE*] = *solver* (*@F*, *tspan*, *y*0, *options*) – в дополнение к описанному решению содержит свойства *Events*, установленные в структуре *options* ссылкой на функции событий. Когда эти функции событий от (*t*, *y*) равны нулю, производятся действия в зависимости от значения трех векторов *value*, *isterminal*, *direction* (их величины можно установить в *m*-файлах функций событий). Для *i*-й функции событий *value* (*i*) – значение функции, *isterminal* (*i*) – прекратить интеграцию при достижении функцией нулевого значения,

direction (*i*) = 0, если все нули функции событий нужно вычислять (по умолчанию), +1 – только те нули, где функция событий увеличивается, –1 – только те нули, где функция событий уменьшается. Выходной аргумент *TE* – вектор-столбец времен, в которые происходят события (*events*), строки *YE* являются соответствующими решениями, а индексы в векторе *IE* определяют, какая из *i* функций событий (*event*) равна нулю в момент времени, определенный *TE*. Когда происходит вызов функции без выходных аргументов, по умолчанию вызывается выходная функция *odeplot* для построения вычисленного решения. В качестве альтернативы можно, например, установить свойство *OutputFcn* в значение *'odephas2'* или *'odephas3'* для построения двумерных или трехмерных фазовых плоскостей.

Параметры интегрирования (*options*) могут быть определены и в *m*-файле, и в командной строке с помощью команды *odeset*. Если параметр определен в обоих местах, определение в командной строке имеет приоритет.

Решатели используют в списке параметров различные параметры (некоторые из них):

NormControl – управление ошибкой в зависимости от нормы вектора решения [*on* | {*off*}]. Флаг «*on*», чтобы *norm*(*e*) <= max (*RelTol*norm*(*y*), *AbsTol*). По умолчанию все решатели используют более жесткое управление по каждой из составляющих вектора решения;

RelTol – относительный порог отбора [положительный скаляр]. По умолчанию 1e-3 (0.1% точность) во всех решателях; оценка ошибки на каждом шаге интеграции *e*(*i*) <= max (*RelTol*abs*(*y*(*i*)), *AbsTol*(*i*));

AbsTol – абсолютная точность [положительный скаляр или вектор {1e-6}]. Скаляр вводится для всех составляющих вектора решения, а вектор указывает на компоненты вектора решения. *AbsTol* по умолчанию 1e-6 во всех решателях;

Refine – фактор уточнения вывода [положительное целое число] – умножает число точек вывода на этот множитель. По умолчанию всегда равен 1, кроме *ode45*, где он 4 (не применяется, если *tspan* > 2);

OutputFcn – дескриптор функция вывода [*function*] – имеет значение в том случае, если решатель вызывается без явного указания функции вывода, *OutputFcn* по умолчанию вызывает функцию *odeplot* (эту установку можно поменять именно здесь);

OutputSel – индексы отбора [вектор целых чисел] Устанавливаются компоненты, которые поступают в *OutputFcn*. *OutputSel* по умолчанию выводит все компоненты;

Stats – статистика стоимости вычислений [*on* | {*off*}];

Vectorized – векторизованная ode-функция [*on* | {*off*}]. Устанавливается на *'on'*, если ODE-функция *F* *F*(*t*,[*y*1 *y*2...]) возвращает вектор [*F*(*t*, *y*1) *F*(*t*, *y*2) ...];

Events – [*function*] – вводятся дескрипторы функций событий, содержащих собственно функцию в векторе *value*, и векторы *isterminal* и *direction*;

Mass – матрица массы [*constant matrix* | *function*]. Для задач $M \cdot y' = f(t, y)$ установливается имя постоянной матрицы. Для задач с переменной *M* вводится дескриптор функции, описывающей матрицу массы;

InitialStep – предлагаемый начальный размер шага, по умолчанию каждый решатель определяет его автоматически по своему алгоритму;

InitialSlope – вектор начального уклона *yp*0 *yp*0 = *F* (*t*0, *y*0) / *M* (*t*0, *y0*);

MaxStep – максимальный шаг, по умолчанию во всех решателях равен одной десятой интервала *tspan*;

BDF (*Backward Differentiation Formulas*) [*on* | {*off*}] – указывает, нужно ли использовать формулы обратного дифференцирования (методы *Gear*) вместо формул численного дифференцирования, используемых в *ode15s* по умолчанию;

MaxOrder – максимальный порядок *ode15s* [1 2 3 | 4 | {5}].

Изменить значения параметров можно в окне *Workspace*, сделав двойной щелчок по *Option*. Список параметров показан на рис. 3.17.

Для решения дифференциальных уравнений методом Рунге–Кутты, алгоритм которого реализован в функциях *Toolbox*, сначала создаются *m*-файлы, содержащие определение функций моделей.

```
function dHdt = RungeH (t, H)
% Модель водоема
p = [0.2  0.3  0.5  1  0.2]; % массив осадков
N = 10; % число шагов по времени
kmorf = 0.15; % морфометрический коэффициент
N1=length(p)+1;
p(N1:N) = 0;
```

```
dHdt = p(round(t)) - kmorf*H;
end
```

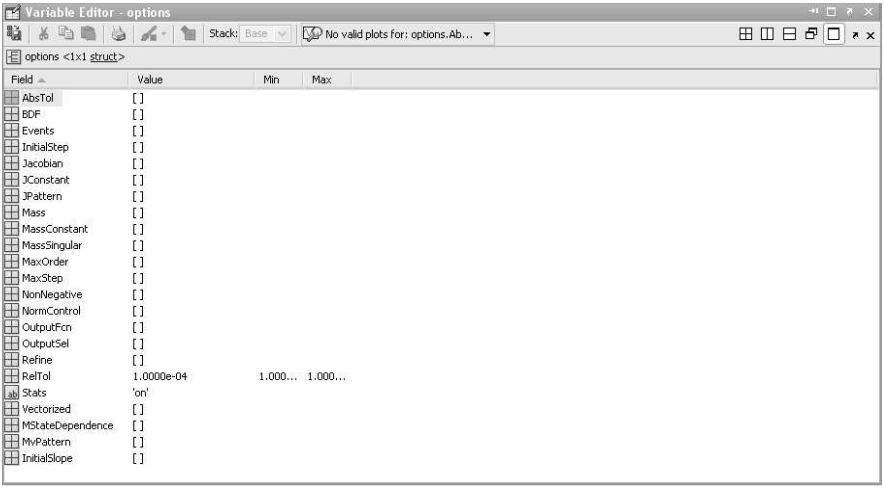


Рис. 3.17. Список параметров решателей.

```
function dQdt = RungeQ(t, Q)
% Модель склонового стока
p = [2 3 4 7 10 6 2]; % массив осадков
N = 10; % число шагов по времени
k = 0.75; % коэффициент стока
tau = 2; % время добегания
N1=length(p)+1;
p(N1:N)=0;
dQdt = (k*p(round(t)) - Q)/tau;
end
```

Вызов функции осуществляется стандартным способом с параметрами, описанными выше.

Для модели водоема:

```
>> H0 = 1;
>> options = odeset('RelTol', [1e-4], 'Stats', 'on');
>> [t, H] = ode45('RungeH', [1:0.01:10], H0, options);
>> plot(t, H)
>> xlabel ('t');
```

```
>> ylabel ('H');
```

Результат представлен на рис. 3.18, *а*.

Для модели склонового стока:

```
>> Q0 = 1;
>> options = odeset('RelTol', [1e-4], 'Stats', 'on');
>> [t, Q] = ode45('RungeQ', [1:0.01:10], Q0, options);
>> plot(t, Q)
>> xlabel ('t');
>> ylabel ('Q');
```

Результат представлен на рис. 3.18, *б*.

а) *б*)

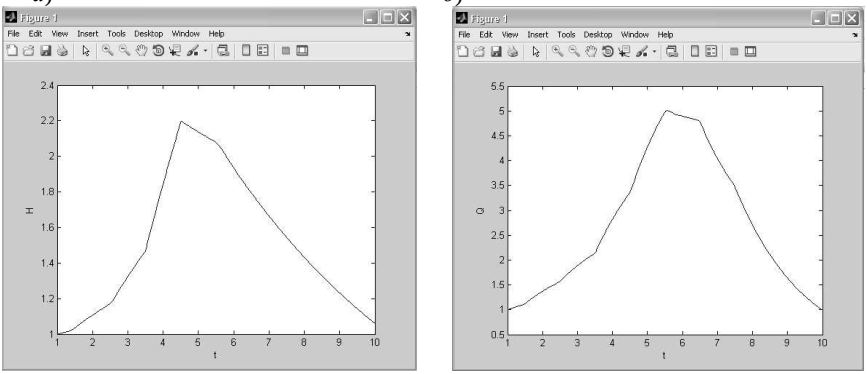


Рис. 3.18. Результаты моделирования моделей водоема (*а*) и склонового стока (*б*) методом Рунге–Кутты.

3.5.3. Решение уравнений в частных производных

Для решения систем параболических и эллиптических дифференциальных уравнений в частных производных введен пакет расширения *Partial Differential Equations Toolbox*, который содержит мощные средства решения и визуализации.

Для решения системы уравнений в частных производных с одной пространственной координатой в *Matlab* применяется функция *pdepe*. Общий вид системы относительно неизвестной вектор-функции $U(x, t)$ может быть представлен следующим уравнением:

$$C\frac{\partial U}{\partial t} = x^{-m}\frac{\partial}{\partial x}(x^m F) + S\,. \qquad (3.7)$$

Здесь матрица *C*, векторы *F* и *S* зависят от переменных $x, t, U, \frac{\partial U}{\partial x}$. Матрица *C* является диагональной, по крайней мере, один элемент ее должен быть ненулевым. Например, для уравнения теплопроводности вектор *F* определяет поток, а вектор *S* описывает источники. Введение параметра *m* позволяет решать одномерные задачи для радиальной координаты в цилиндрической или полярной (*m* = 1), а также сферической (*m* = 1) системах координат.

Начальные условия определяются для вектора неизвестных:

$$U(x, t_0) = U_0(x)\,. \qquad (3.8)$$

Краевые условия задаются в следующем виде:

$$P(x, t, U) + Q(x, t)F(x, t, U, \frac{\partial U}{\partial x}) = 0\,. \qquad (3.9)$$

Обращение к функции решения системы уравнений в частных производных имеет вид:

SOL = pdepe (m, PDE, IC, BC, XX, TT, OPT, P1, P2, ...)

Здесь параметр *m* определяет систему координат: 0 отвечает декартовым координатам, 1 – цилиндрическим, 2 – сферическим. Вычисление правой части рассматриваемой системы производится в функции *PDE*, данные о начальных условиях берутся из функции *IC*, а краевые условия задаются функцией *BC*.

Сетка по координате *x* на интервале [*a*, *b*] должна содержать не менее трех узлов и задается массивом *XX*, содержащим узлы в порядке возрастания координаты. Лучше использовать более мелкий шаг между узлами в тех областях, где решение меняется значительно, и более крупный шаг в местах плавного изменения решения. Для *m* > 0 (цилиндрические или сферические координаты) не обязательно сгущать сетку в окрестности особой точки *x* = 0, и при *m* > 0 значение *a* не может быть отрицательным.

Для интегрирования задачи по времени в функции *pdepe* используются команда для решения задачи Коши (*ode15s* и др.), а применяемый метод и шаг интегрирования выбираются функцией *pdepe*. Массивом *TT* определяется набор временных слоев, в которых будут запомнены решения, таких слоев должно быть не менее трех. Некоторые расчетные параметры можно изменить при помощи команды *odeset*. В частности, доступны такие параметры, как относительная (*RelTol*) и абсолютная (*AbsTol*) величины погрешности, начальный (*Initial Step*) и максимальный (*MaxStep*) шаги интегрирования.

Решение выводится в трехмерный массив *SOL*, причем первый индекс (*i*) отвечает временному слою t_i, второй индекс (*j*) определяет номер узла x_j, а третий индекс дает номер компоненты вектора решения. Таким образом, элемент *SOL* (*i*, *j*, *k*) дает значение решения $U^k(x^j, t^i)$.

Рассмотрим заголовки функций вычисления правой части (РОЕ) и задания начальных (*IC*) и краевых (*BC*) условий. Входными параметрами для функции вычисления правой части с именем *PDE* являются точка отрезка *X*, вектор значений в этой точке *Y* и параметры задачи *P*1, *P*2, … Выходные данные представлены диагональной матрицей *C* и векторами *F*, *S*:

[C, F, S] = PDE (X, T, U, P1, P2, ...)

Начальные условия даются функцией:

U = IC (X, P1, P2, …)

Краевые условия обрабатываются функцией:

[Pa, Qa, Pb, Qb] = BC (a, Ua, b, Ub, T, P1, P2, ...)

Здесь векторы *Ua* и *Ub* дают значения вектора на разных концах интервала [*a*, *b*], *T* – время, а выходные параметры представлены векторами на левом (*Pa*, *Qa*) и правом (*Pb*, *Qb*) концах интервала.

В дополнение к *pdepe* имеется команда *pdeval* для пересчета решения на другую сетку и получения массива решения пространственных производных. Обращение к этой команде имеет следующий вид:

[Uout, DUout] = pdeval (m, XX, U, Xout)

Здесь *U* – решение, полученное на сетке *XX*, *Uout* и *DUout* есть соответственно функция и ее пространственная производная на новой сетке *Xout*.

3.6. Подсистема *Simulink*

Подсистема *Simulink* – это интерактивная среда для моделирования и анализа широкого класса динамических систем, использующая графический язык блок-диаграмм.

Основным элементом в процессе построения модели в пакете *Simulink* является *блок*. Блок представляет собой систему типа «вход–состояние–выход» (или просто «вход–выход») и может быть как простым, так и составным. С помощью связей экземпляров блок (параметры, которых могут быть изменены в зависимости от требований, предъявляемых к моделируемой системе) объединяются в единую схему, которая создаются в окне построения модели.

Библиотека *Simulink* состоит из 15 разделов, тринадцать из которых являются главными и не могут быть изменены пользователем (табл. 3.3).

Таблица 3.3

Общая характеристика разделов *Simulink*

Раздел	Характеристика
Sinks	Визуализация результатов, получаемых при моделировании
Sources	Формирование сигналов, которые при моделировании обеспечивают работу S-модели в целом или отдельных ее частей
Continuous	Реализация динамических звеньев, описываемых линейными дифференциальными уравнениями с постоянными коэффициентами
Discrete	Моделирование систем с дискретным временем
Math Operations	Реализация встроенных математических функций
Discontinuities	Реализация некоторых типовых нелинейных зависимостей выходной величины от входной
User Defined Functions	Содержаться блоки, назначение которых определяет сам пользователь
Signals Routing	Обеспечение пересылки сигналов
Signals Attributes	Определение или изменение атрибутов сигнала
Ports & Subsystems	Разработка сложных S-моделей
Look-Up Tables	Формирование выходного сигнала, зависимость которого от входного сигнала задана с помощью таблицы соответствий
Model Verification	Проверка статистических и динамических характеристик модели
Model-Wide Utilities	Линеаризация динамической модели и оформление документации

Подробное описание подсистемы *Simulink* можно найти в источниках списка литературы. Здесь подробно опишем только те разделы, которые необходимы для моделирования систем – разделы *Sinks* и *Continuous*.

В раздел *Sinks* входят следующие блоки:

– *Scope* – блок с одним входом, который используется для вывода в графическое окно графика зависимости величины, подаваемой на его вход, от модельного времени;

– *Floating Scope* – блок с одним входом, выполняющий аналогичные функции;

– *XYGraph* – блок с двумя входами, который обеспечивает построение графика зависимости одной моделируемой величины о другой;

– *Display* – блок с одним входом, предназначенный для отображения числовых значений входной величины;

– *Out* – выходной порт для вывода результатов вне модели;

– *Terminator* – порт для вывода результатов «в никуда»;

– *To File* – блок, обеспечивающий сохранение результатов моделирования на диске в MAT-файле;

– *To Workspace* – блок, который сохраняет результаты в рабочем пространстве.

Раздел *Continuous* (Непрерывные элементы) содержит блоки, которые разделяются на следующие группы:

– блоки общего назначения (интеграторы, дифференциаторы);

– блоки задержки сигнала;

– блоки линейных стационарных звеньев.

Они реализуют динамические звенья, описываемые линейными дифференциальными уравнениями с постоянными коэффициентами, т. е. линейные стационарные звенья:

– *Integrator* – представляет идеальное интегрирующее звено (интегратор);

– *Derivative* – представляет идеальное дифференцирующее звено;

– State-Space – позволяет задать линейное звено путем ввода четырех матриц его пространства состояний;

– *Transfer Fen* – позволяет задать линейное звено путем ввода его передаточной функции;

– *Zero-Pole* – используется для того, чтобы задать звено посредством указания векторов значений его полюсов и нулей, а также значения коэффициента передачи;

– *Transport Delay* – обеспечивает задержку сигнала на заданное количество шагов модельного времени (не обязательно целое число);

– *Variable Transport Delay* – позволяет задавать управляемую извне величину задержки.

Пользуясь этими блоками, можно, собирая в окне блок-схемы различные звенья и соединяя их между собой, составлять модели сложных систем автоматического управления, а затем осуществлять моделирование во времени поведения этих систем, задавая разные входные воздействия.

Блок *Integrator* позволяет осуществлять интегрирование входного сигнала в непрерывном времени. Окно его настройки содержит следующие параметры:

– *External reset* – подключение дополнительного управляющего сигнала;

– *Initial condition source* – определение источника (внутренний или внешний) установки начального значения выходного сигнала;

– *Initial condition* – начальное значение выходной величины, вводится в строке редактирования или как числовая константа либо в виде вычисляемого выражения;

– *Limit output* – ограничение величины выхода (этот флажок определяет, будут ли использоваться следующие три параметра);

– *Upper saturation limit* – верхнее предельное значение выходной величины, по умолчанию не ограничено (значение *inf*);

– *Lower saturation limit* – нижнее предельное значение выходной величины (по умолчанию параметр имеет значение *-inf*);

– *Show saturation port* – показать порт насыщения;

– *Show state port* – показать порт состояния;

– *Absolute tolerance* – допустимая предельная величина абсолютной погрешности.

Параметр *External reset* может принимать такие значения: *none* – дополнительный управляющий сигнал не используется; *rising* – для управления используется нарастающий сигнал; *falling* – для управления используется убывающий сигнал; *either* – на работу блока влияет изменение управляющего сигнала в любом направлении.

Параметр *Initial condition source* принимает одно из двух значений: *internal* – используется внутренняя установка начального значения выходной величины; *external* – установка начальных условий будет осуществляться извне.

Настройка блока *Transport Delay*, обеспечивающего задержку сигнала на заданное число шагов модельного времени, происходит по трем параметрам:

– *Time delay* – количество шагов модельного времени, на которое следует задержать сигнал (может вводиться в числовой форме или в форме вычисляемого выражения);

– *Initial input* – начальное значение входа, по умолчанию равняется 0;

– *Initial buffer size* – начальный объем буфера памяти (в байтах), который выделяется в рабочем пространстве *Matlab* для сохранения параметров задержанного сигнала (значение должно быть кратным 8, по умолчанию – 1024).

Особенность применения подсистемы *Simulink* состоит в том, что модель представляется в виде набора соединенных между собой блоков – схемы. А решение дифференциальных уравнений реализуется решателями, описанными выше. Функция использующая *Simulink* выглядит следующим образом:

[*T, X, Y*] = *sim* (*@mode1, tspan, y0, options, ut, p*1, *p*2 ....)

Для лучшего объяснения схем необходимо модели переписать следующем образом:

модель водоема

$$\frac{dH_{\text{ср}}}{dt} = -k_{\text{морф}} H_{\text{ср}} + \vec{\xi},$$

модель склонового стока

$$\frac{dQ}{dt} = -\frac{1}{k_{\text{st}}\tau} Q + \frac{\dot{X}}{\tau},$$

модель кинематической волны

$$\frac{\partial F}{\partial t} = -a\frac{\partial F}{\partial x} + q(x,t)\,.$$

На рис. 3.19 представлена схема решения модели водоема.

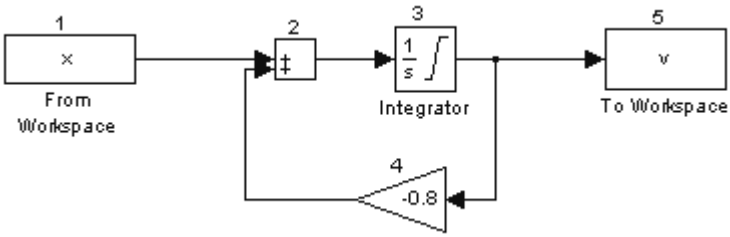


Рис. 3.19. Схема решения модели водоема.

1 – блок From Workspace обеспечивает ввод в модель данных непосредственно из рабочего пространства *MatLab*, раздел *Sources*. Параметры: входная матрица (*x*) должна содержать время и значение входа для этого времени.

2 – блок Sum суммирует поступающие на него сигналы, раздел *Math Operations*. Поступают осадки и значение уровня водоема за предшествующий шаг расчета, умноженное на коэффициент морфометрии (по умолчанию равный 0.8). Организован цикл из блоков 2, 3 и 4. Параметры: суммирование двух входов.

3 – блок *Integrator*, интегрирующее звено, раздел Continuous. Параметры: начальное значение уровня водоема равное 1 и нижний предел равный начальному условию.

4 – блок *Gain* является линейным усилительным звеном, раздел *Math Operations*. Параметры: значение коэффициента морфометрии с отрицательным знаком.

5 – Блок *To Workspace* сохраняет результаты моделирования в рабочем пространстве, раздел *Sinks*. Параметры: имя выходной матрица – *v*.

Пример использования данной реализации из рабочего пространства:

```
>> x = [1 0.2; 2 0.3; 3 0.5; 4 1; 5 0.2];
```

```
% Входная матрица – момент времени и значение осадков.
% Для расчетных шагов больше 5 значения осадков считаются равными 0
% (данное свойство настраивается в первом блоке).
>> [t, v] = sim('sx_vod', [1 10]);
% Стандартная функция запуска процесса моделирования S–модели.
% Входные аргументы функции – символьная строка, содержащая
% имя mdl–файла, который включает запись соответствующей модели;
% вектор, состоящий из двух элементов – значений начального и конечного
% моментов времени моделирования.
>> plot (t, v)
 % Графическое изображение результатов моделирования.
```

Т.е.:

```
>>x = [1 0.2; 2 0.3; 3 0.5; 4 1; 5 0.2]
x =
    1     0.2
    2     0.3
    3    0.5
    4     1.0
    5     0.2
>> [t,v] = sim('sx_vod', [1 10]);
>> plot(t, v)
>> xlabel('Time')
>> ylabel('Level of lake')
```

Результат представлен на рис. 3.21, *а*.

Под рис. 3.20, на котором представлена схема модели склонового стока, приведено описание блоков этой схемы отличных от модели водоема.

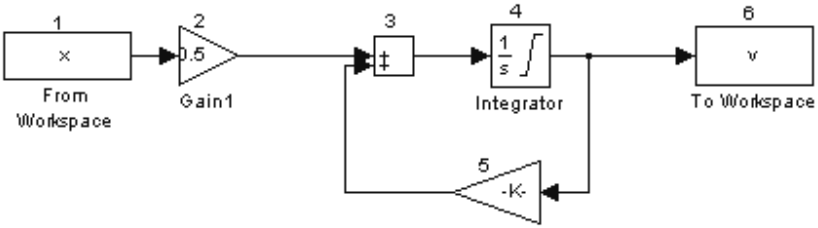


Рис. 3.20. Схема решения модели склонового стока.

2 – блок Gain раздела *Math Operations*. Параметры: единица, деленная на значение времени добегания.

3 – блок Sum раздела *Math Operations*. Поступают осадки и значение расхода воды за предшествующий шаг расчета, поделенное на произведение коэффициента стока на время добегания, по умолчанию 0,5 и 2 соответственно (блок 5). Параметры: суммирование двух входов.

4 – блок *Integrator* раздела *Continuous*. Параметры: начальное значение расхода воды – 1 и нижний предел равный начальному условию.

Использование модели из рабочего пространства:

```
x = [1 2; 2 3; 3 4; 4 7; 5 10; 6 6; 7 2]
x =
    1    2
    2    5
    3   10
    4    6
    5    1
>> [t,v] = sim('sx_sklon', [1 10]);
>> plot(t, v)
>> xlabel('Time')
>> ylabel('Dischage')
```

Результат представлен на рис. 3.21, *б*.

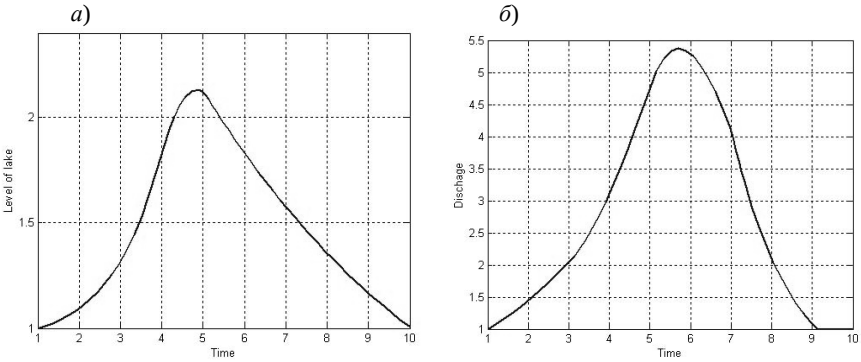


Рис. 3.21. Результаты моделирования моделей водоема (*а*) и склонового стока (*б*) в *Simulink.*

Список литературы

1. Абиев Р.Ш. Вычислительная гидродинамика и тепломассообмен. – СПб.: Изд-во НИИХимии СПбГУ, 2002. – 576 с.
2. Бенькович Е., Колесов Ю., Сениченков Ю. Практическое моделирование динамических систем.– СПб.: БХВ-Петербург, 2002.– 464 с.
3. Вагер Б.Г. Численные методы решения дифференциальных уравнений. – СПб.: изд. РГГМУ, 2005. – 126 с.
4. Вержбицкий В.М. Основы численных методов: Учебник для вузов. – М.: Высш. Шк., 2002. – 840 с.
5. Выгодский М.Я. Справочник по высшей математике. – М.: АСТ: Астрель, 2006. – 991с.
6. Дьяконов В. *Matlab*. Полный самоучитель. – М.: ДМК Пресс, 2012. – 768 с.
7. Дьяконов В.П. *Matlab* 6/6.1/6.5, *Simulink* 4/5 в математике и моделировании.– М.: Изд. «СОЛОН-Пресс», 2003.– 576 с.
8. Дьяконов В.П. *Simulink*. Самоучитель. – М.: ДМК Пресс, 2012. – 784 с.
9. Емельянов В.Н. Численные методы: введение в теорию разностных схем: учебное пособие для академического бакалавриата. 2-е изд., испр. и доп. – М.: Издательство Юрайт, 2018. – 188 с. – Режим доступа: www.biblio-online.ru/book/5A97B60B-81DD-46CA-A884-DB21BDE8C603.
10. Кетков Ю., Кетков А., Шульц М. *Matlab* 7: программирование, численные методы.– СПб.: БХВ-Петербург, 2005.– С. 752.
11. Коваленко В.В. Метод характеристик в частично инфинитной гидрологии. – СПб.: изд. РГГМУ, 2012. – 136 с.
12. Коваленко В.В., Викторова Н.В., Гайдукова Е.В. Моделирование гидрологических процессов. – СПб.: изд. РГГМУ, 2006. – 559 с.
13. Мэтьюз Джон Г., Финк Куртис Д. Численные методы. Использование *MatLab*. – М.: изд. дом «Вильямс», 2001. – 720 с.
14. Поршнев С.В. MATLAB 7. Основы работы и программирования. Учебник. – М.: ООО «Бином-Пресс», 2011. – 320 с.
15. Турчак Л.И., Плотников П.В. Основы численных методов. – М.: Физматлит, 2002. – 304 с.

Содержание

Учебное издание

Екатерина Владимировна Гайдукова
Наталья Владимировна Викторова

Численные методы в гидрологии

Учебное пособие

Печатается в авторской редакции

Подписано в печать 30.12.2019 Формат 60×90 1/16. Гарнитура Times New Roman.
Печать цифровая. Усл. печ. л. 7. Тираж 200 экз. Заказ № 864.
РГГМУ, 192007, Санкт-Петербург, Воронежская ул., 79.