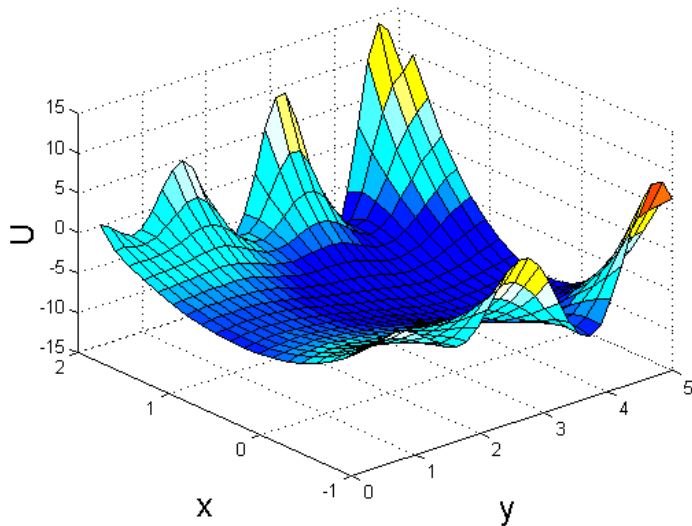




Е.А. Рындин, И.В. Куликова, И.Е. Лысенко

ОСНОВЫ ЧИСЛЕННЫХ МЕТОДОВ: ТЕОРИЯ И ПРАКТИКА

e-Book



2015

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южный федеральный университет»

Е.А. Рындин, И.В. Куликова, И.Е. Лысенко

Основы численных методов: теория и практика

Электронное учебное пособие

Рекомендовано УМО РАЕ по классическому университетскому и техническому образованию в качестве учебного пособия для студентов высшего профессионального образования, обучающихся по направлениям подготовки: 11.03.03 – «Конструирование и технология электронных средств», 11.03.04 – «Электроника и микроэлектроника», 28.03.04 – «Нанотехнологии и микросистемная техника».

Таганрог
2015

Содержание

Введение	v
1 Постановка задачи. Уравнения непрерывной модели	1
1.1. Эллиптические уравнения	3
1.1.1. Уравнение Пуассона	3
1.1.2. Уравнение Лапласа	8
1.2. Параболические уравнения	10
1.2.1. Уравнение теплопроводности	10
1.3. Гиперболические уравнения	12
1.3.1. Волновое уравнение	12
2 Граничные и начальные условия	14
2.1. Граничные условия	15
2.2. Начальные условия	18
3 Нормировка уравнений	19
4 Координатные сетки и сетки по времени	23
4.1. Разновидности сеток	24
4.2. Прямоугольные координатные сетки	24
4.3. Триангулярные координатные сетки	26
5 Дискретизация дифференциальных уравнений в частных производных	33
5.1. Методы дискретизации дифференциальных уравнений	33
5.2. Метод конечных разностей	34
5.2.1. Сеточные функции	34
5.2.2. Конечные разности первого порядка	36

5.2.3.	Конечные разности второго порядка	38
5.2.4.	Смешанные конечные разности	38
5.2.5.	Шаблоны	39
5.2.6.	Пример конечно-разностного представления дифференциальных уравнений	42
5.3.	Метод контрольных объемов	44
5.3.1.	Пример дискретизации дифференциального уравнения с использованием метода контрольных объемов на триангулярной координатной сетке	44
6	Решение систем алгебраических уравнений	48
6.1.	Методы решения систем линейных алгебраических уравнений	49
6.1.1.	Классификация методов решения систем линейных алгебраических уравнений	49
6.1.2.	Матричный метод	50
6.1.3.	Методы прямой и обратной подстановки	50
6.1.4.	Метод исключения Гаусса	52
6.1.5.	Метод LU-разложения	53
6.1.6.	Итерационные методы решения СЛАУ	56
6.2.	Методы решения систем нелинейных алгебраических уравнений	74
6.2.1.	Итерация неподвижной точки	75
6.2.2.	Критерий сходимости итерации неподвижной точки	76
6.2.3.	Метод Ньютона – Рафсона	81
6.2.4.	Критерий сходимости метода Ньютона – Рафсона	84
7	Классификация погрешности	87
7.1.	Абсолютная и относительная погрешности	88
	Заключение	89
А	Примеры решения задач матфизики в Matlab	90
A.1.	Примеры решения уравнения Пуассона	90
A.1.1.	Решение одномерного уравнения Пуассона методом конечных разностей	90

А.1.2. Решение двухмерного уравнения Пуассона методом конечных разностей	97
А.1.3. Решение двухмерного уравнения Пуассона методом конечных элементов	108
А.2. Примеры решения уравнения теплопроводности	117
А.3. Примеры решения волнового уравнения	133
Б Справочник MatLab	148
Б.1. Введение	148
Б.1.1. Алфавит языка программирования	148
Б.1.2. Арифметические и логические операторы	148
Б.1.3. Элементарные функции	150
Б.1.4. Понятие о файлах-сценариях и файлах-функциях	150
Б.2. Основы программирования	153
Б.2.1. Оператор присваивания	153
Б.2.2. Перенос строки	153
Б.2.3. Ввод и вывод данных	153
Б.2.4. Форматы чисел	154
Б.2.5. Формирование векторов и матриц	154
Б.2.6. Оператор двоеточие :	156
Б.2.7. Формирование массивов специального вида	157
Б.2.8. Оператор разветвления if	158
Б.2.9. Операторы циклов	159
Б.3. Работа с массивами	161
Б.3.1. Решение систем линейных алгебраических уравнений	161
Б.3.2. Основные операции над массивами	162
Б.4. Графика	166
Б.4.1. Двумерные графики	166
Б.4.2. Трехмерные графики	168
Б.4.3. Надписи и пояснения к графикам	180
Б.4.4. Специальная графика	185
В Векторный анализ	189
В.1. Умножение векторов	189
В.2. Градиент скалярного поля	190
В.3. Дивергенция векторного поля	190
В.4. Оператор Лапласа	190

Г Краткая инструкция по работе в GMSH	191
Г.1. Начало работы	192
Г.2. Создания файла в командном режиме	192
Г.3. Наложение сетки	194
Г.4. Формат файлов .msh ASCII	194
Д Реализация метода Якоби	199
Е Реализация методов Якоби, Гаусса-Зейделя и CG	204
Предметный указатель	216
Библиографический список	216

Введение

Разработка современных сверхбольших интегральных схем (СВИС) и микрооптикоэлектромеханических систем (МОЭМС) и даже их отдельных элементов, представляющих собой сложные микро- и наноразмерные структуры, в основу функционирования которых положены различные физические эффекты, практически невозможна без решения уравнений математической физики, представляющих собой, как правило, дифференциальные уравнения (ДУ) в частных производных [1].

Нахождение точного аналитического решения рассматриваемых дифференциальных уравнений и их систем, к сожалению, возможно лишь для весьма ограниченного круга задач при использовании целого ряда допущений, негативно отражающихся на адекватности полученных результатов. Для решения задач математической физики в случае нескольких измерений необходимо использовать численные методы, позволяющие преобразовать дифференциальные уравнения или их системы в системы алгебраических уравнений. Для решения полученных нелинейных систем алгебраических уравнений или линейных систем большой размерности используют итерационные методы. При этом одной из наиболее сложных проблем является обеспечение сходимости итерационного процесса, в значительной степени определяющей время вычислений. Точность решения определяется шагом координатной сетки, количеством итераций и разрядной сеткой компьютера.

В общем виде маршрут численного решения уравнений математической физики включает следующие основные этапы:

- 1) постановка задачи:

- анализ возможных допущений и приближений;

- составление уравнений непрерывной модели (дифференциальных уравнений в частных производных, граничных и начальных условий);
 - выбор числа пространственных измерений;
- 2) нормировка (при необходимости):
- выбор системы нормирующих коэффициентов;
 - запись уравнений непрерывной модели в нормированном (безразмерном) виде;
- 3) формирование координатной и (при необходимости) временной сеток:
- выбор вида координатной сетки (триангулярная или прямоугольная, равномерная или неравномерная);
 - определение шагов координатной сетки, сетки по времени и, при необходимости, сеток по параметрам;
 - построение сеток;
- 4) дискретизация уравнений непрерывной модели (дифференциальных уравнений в частных производных, граничных и начальных условий):
- выбор метода дискретизации (метод конечных разностей или метод конечных элементов);
 - построение системы алгебраических уравнений (линейной или нелинейной);
- 5) решение системы алгебраических уравнений:
- анализ системы алгебраических уравнений, выбор метода ее решения;
 - обеспечение сходимости;
 - нахождение значений искомых функций в точках координатной и временной сеток;
- 6) интерполяция полученного решения (при необходимости);

- 7) денормировка интерполированного решения (при необходимости);
- 8) вывод результатов решения задачи.

Знакомство студентов технических вузов с численными методами вызвано необходимостью осознанного их применения при проведении модельных исследований с использованием специализированных систем инженерного анализа (CAE — Computer Aided Engineering), а также в процессе разработки собственных программных средств численного решения ДУ.

В настоящее время сложилась традиция выделять в приведенном выше маршруте численного решения уравнений математической физики три основных блока:

- *препроцессор* — разработка геометрической модели, формирование координатной и временной сеток, ввод параметров материалов, граничных и начальных условий;
- *решатель* (солвер) — дискретизация системы дифференциальных уравнений и решение полученной системы алгебраических уравнений;
- *постпроцессор* — обработка и визуализация полученных решений.

В учебном пособии кратко представлены все перечисленные блоки, но основное внимание уделено рассмотрению методов разработки решателя, являющегося ядром численного эксперимента. Показаны основные уравнения математической физики, особенности задания граничных и начальных условий, методы дискретизации дифференциальных уравнений в частных производных, методы решения систем алгебраических уравнений, основные этапы решения задач матфизики, включая постановку задачи, выбор метода дискретизации, формирование координатной сетки, выбор шаблона, метода решения, анализ сходимости и др.

Рассмотренные методы решения уравнений проиллюстрированы примерами исходных текстов MATLAB и C++ с комментариями и рекомендациями, позволяющими составить представление

об основных правилах и приемах разработки компьютерных программ для решения уравнений математической физики. Теоретический материал дан в минимальном объеме, необходимом для изучения основ численных методов решения систем дифференциальных уравнений в частных производных. Основное внимание уделено практической (программной) реализации данных методов. Приведены примеры использования ряда эффективных программных приложений: приложение PDETOOL системы MATLAB, сеточный генератор GMSH — препроцессор для построения параметризованных моделей, формирования сеток и визуализации результатов и др. Методам решения подобных задач посвящено достаточно много монографий, учебников и учебных пособий [2, 3, 4, 5, 6]. В данном учебном пособии предпринята попытка достичь более полного соответствия целям подготовки специалистов в области проектирования электронно-вычислительных средств и микросистем по характеру материала, стилю его изложения и приводимым примерам.

Электронное учебное пособие разработано при финансовой поддержке Министерства образования и науки Российской Федерации (проект №213.01–11/2014–12 «Разработка и исследование методов построения многоосевых функционально-интегрированных микро- и наномеханических сенсоров угловых скоростей и линейных ускорений» в рамках базовой части государственного задания, регистрационный №01201458539).

Глава 1

Постановка задачи. Уравнения непрерывной модели

Разработка и исследование значительной части элементов современных СБИС и МОЭМС связана с решением так называемых задач математической физики (или сокращенно – матфизики), к которым относятся задачи теплопроводности, диффузии, электростатики и электродинамики, задачи о течении жидкости, о распределении плотности электрического тока в проводящей среде, задачи о деформациях твердых тел и многие другие.

Подобные задачи описываются дифференциальными уравнениями (ДУ) в частных производных с дополнительными уравнениями, выражающими граничные и начальные условия. В учебном пособии рассматриваются ДУ в частных производных не выше второго порядка, поскольку эти уравнения охватывают достаточно широкий диапазон физических явлений, положенных в основу функционирования элементов СБИС и МОЭМС и, кроме того, рассмотренные ниже методы решения применимы и к ДУ в частных производных более высоких порядков. В общем случае линейное дифференциальное уравнение в частных производных второго порядка с n незави-

симыми переменными имеет вид

$$\sum_{\alpha, \beta=1}^n A_{\alpha\beta}(\mathbf{x}) \frac{\partial^2 u}{\partial x_\alpha \partial x_\beta} + \sum_{\alpha=1}^n B_\alpha(\mathbf{x}) \frac{\partial u}{\partial x_\alpha} + C(\mathbf{x})u = f(\mathbf{x}), \quad (1.1)$$

где $x = [x_1, x_2, \dots, x_n]$ – вектор (матрица-строка) независимых переменных¹; u – искомая функция независимых переменных; $A_{\alpha\beta}(\mathbf{x})$, $B_\alpha(\mathbf{x})$, $C(\mathbf{x})$, $f(\mathbf{x})$ – некоторые вещественные функции независимых переменных [2].

Уравнение (1.1) всегда может быть приведено к одной из трех стандартных канонических форм. По соотношению значений $A_{\alpha\beta}(\mathbf{x})$ уравнения относят к эллиптическим, параболическим или гиперболическим в точке \mathbf{x} . В частности, для дифференциальных уравнений в частных производных второго порядка с двумя независимыми переменными x, y , которые могут быть представлены в виде

$$\begin{aligned} A_{xx}(x, y) \frac{\partial^2 u}{\partial x^2} + A_{xy}(x, y) \frac{\partial^2 u}{\partial x \partial y} + A_{yy}(x, y) \frac{\partial^2 u}{\partial y^2} + \\ + B_x(x, y) \frac{\partial u}{\partial x} + B_y(x, y) \frac{\partial u}{\partial y} + C(x, y)u = f(x, y), \end{aligned} \quad (1.2)$$

тип ДУ определяется знаком выражения, называемого **дискриминантом**

$$D(x, y) = A_{xy}^2(x, y) - 4A_{xx}(x, y)A_{yy}(x, y). \quad (1.3)$$

Если $D(x, y) < 0$, дифференциальное уравнение является **эллиптическим** в точке (x, y) .

Если $D(x, y) = 0$, дифференциальное уравнение является **параболическим** в точке (x, y) .

Если $D(x, y) > 0$, дифференциальное уравнение является **гиперболическим** в точке (x, y) .

Если коэффициенты A_{xx} , A_{xy} , A_{yy} постоянные и значение D не зависит от x, y , то в зависимости от знака D уравнение является полностью эллиптическим, гиперболическим или параболическим [2].

¹В дальнейшем по тексту векторные величины и матрицы будут выделяться жирным шрифтом

1.1. Эллиптические уравнения

Рассмотрим некоторые задачи математики, приводящие к решению эллиптических уравнений.

1.1.1. Уравнение Пуассона

Многие стационарные физические процессы (т. е. процессы, при протекании которых искомые функции не изменяются во времени) описываются уравнениями эллиптического типа, в частности, уравнением Пуассона [2], которое для трех направлений координат (x, y, z) может быть записано в виде

$$\begin{aligned} \frac{\partial}{\partial x} \left(A(x, y, z) \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(A(x, y, z) \frac{\partial u}{\partial y} \right) + \\ + \frac{\partial}{\partial z} \left(A(x, y, z) \frac{\partial u}{\partial z} \right) = f(x, y, z), \end{aligned} \quad (1.4)$$

где $u = u(x, y, z)$ — искомая функция; $A(x, y, z)$, $f(x, y, z)$ — некоторые функции независимых переменных.

В операторной форме уравнение Пуассона (1.4) может быть представлено следующим образом:

$$\nabla (A(x, y, z) \cdot \nabla u) = f(x, y, z), \quad (1.5)$$

где ∇ — оператор Наббла, определяемый выражением

$$\nabla = \frac{\partial}{\partial x} + \frac{\partial}{\partial y} + \frac{\partial}{\partial z}. \quad (1.6)$$

Рассмотрим задачу о стационарном распределении тепла в некотором объеме V , ограниченном замкнутой поверхностью S трехмерного пространства $\mathbf{x} = (x, y, z)$.

Процесс теплопроводности, или кондукции, определяется законом Фурье, согласно которому вектор плотности теплового потока \mathbf{W} пропорционален градиенту температуры $T = T(x, y, z)$ [2]:

$$\mathbf{W} = -k \cdot \text{grad}(T), \quad (1.7)$$

где $k = k(x, y, z)$ — коэффициент теплопроводности. Плотность теплового потока равна количеству теплоты, протекающему в единицу времени через единичную площадь изотермической поверхности [1].

Как правило, цель стационарной задачи теплопроводности сводится к необходимости нахождения зависимости температуры от координат (x, y, z) при известном распределении плотности источников тепла $f(x, y, z)$. Поскольку функция $f(x, y, z)$ не входит непосредственно в уравнение Фурье (1.7), необходимо выполнить ряд предварительных преобразований.

Из приведенного выше определения плотности теплового потока следует, что суммарное количество тепла Q_S , прошедшее в единицу времени через замкнутую поверхность S , ограничивающую объем V , в общем случае выражается интегралом

$$Q_s = \oint_S \mathbf{W} \cdot \mathbf{dS}, \quad (1.8)$$

где \mathbf{dS} – вектор, модуль которого численно равен площади dS соответствующего бесконечно малого элемента поверхности, а направление совпадает с направлением нормали к этому элементу; $\mathbf{W} \cdot \mathbf{dS} = W \cdot dS \cdot \cos(\gamma)$ – скалярное произведение векторов \mathbf{W} и \mathbf{dS} ; γ – угол между ними.

Суммарное количество тепла Q_V , выделяющегося в единицу времени в объеме V , ограниченном поверхностью S , определяется интегралом

$$Q_V = \iiint_V f(x, y, z) dV. \quad (1.9)$$

Поскольку по условию задачи рассматриваемый процесс является стационарным, в данном случае уравнение баланса тепла должно отражать факт равенства количества тепла Q_S , прошедшего в единицу времени через замкнутую поверхность S , ограничивающую объем V , и количества тепла Q_V , выделяющегося в единицу времени в этом объеме:

$$\oint_S \mathbf{W} \cdot \mathbf{dS} = \iiint_V f(x, y, z) dV. \quad (1.10)$$

Согласно теореме Остроградского – Гаусса

$$\oint_S \mathbf{W} \cdot \mathbf{dS} = \iiint_V \operatorname{div} \mathbf{W} dV. \quad (1.11)$$

Тогда, подставив (1.11) в (1.10), получим

$$\iiint_V \operatorname{div} \mathbf{W} dV = \iiint_V f(x, y, z) dV, \quad (1.12)$$

что позволяет приравнять подынтегральные выражения

$$\operatorname{div} \mathbf{W} = f(x, y, z). \quad (1.13)$$

Подставив в уравнение (1.13) закон Фурье (1.7), получим уравнение Пуассона для стационарной задачи теплопроводности в векторной форме

$$\operatorname{div}(k(x, y, z) \cdot \operatorname{grad}(T)) = -f(x, y, z). \quad (1.14)$$

Учитывая, что по определению градиент некоторого скалярного поля $u = u(x, y, z)$ определяется выражением

$$\operatorname{grad} U = \frac{\partial U}{\partial x} \mathbf{e}_x + \frac{\partial U}{\partial y} \mathbf{e}_y + \frac{\partial U}{\partial z} \mathbf{e}_z, \quad (1.15)$$

где $\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z$ – единичные векторы (орты) в направлениях соответствующих координатных осей, а дивергенция некоторого векторного поля $\mathbf{v} = \mathbf{v}(x, y, z)$ – выражением

$$\operatorname{div} \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}, \quad (1.16)$$

где v_x, v_y, v_z – проекции вектора \mathbf{v} на соответствующие оси координат, уравнение (1.14) можно переписать в частных производных

$$\begin{aligned} \frac{\partial}{\partial x} \left(k(x, y, z) \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(k(x, y, z) \frac{\partial T}{\partial y} \right) + \\ + \frac{\partial}{\partial z} \left(k(x, y, z) \frac{\partial T}{\partial z} \right) = f(x, y, z), \end{aligned} \quad (1.17)$$

или в операторной форме

$$\nabla (k(x, y, z) \cdot \nabla T) = f(x, y, z). \quad (1.18)$$

Если среда однородна ($k(x, y, z) = \text{const}$), то k можно вынести за знак частной производной в выражении (1.1.1.) или за знак оператора Наббла в выражении (1.18). В результате получим частный случай уравнения Пуассона в виде

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = \frac{f(x, y, z)}{k(x, y, z)}, \quad (1.19)$$

или в операторной форме

$$\Delta T = \frac{f(x, y, z)}{k(x, y, z)}. \quad (1.20)$$

Если среда анизотропна, т.е. коэффициент теплопроводности k зависит от направления распространения тепла и является тензором

$$\mathbf{k} = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}, \quad (1.21)$$

то уравнение преобразуется к виду [2]

$$\sum_{i,j=1}^3 \frac{\partial}{\partial x_i} \left(k_{ij} \frac{\partial}{\partial x_j} \right) = f(x_1, x_2, x_3), \quad (1.22)$$

где пространство (x_1, x_2, x_3) соответствует (x, y, z) .

Если в тензоре \mathbf{k} только элементы главной диагонали отличны от нуля ($k_{ij} = 0$ для $i \neq j$), то уравнение (1.22) может быть записано в виде

$$\frac{\partial}{\partial x} \left(k_{11} \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(k_{22} \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left(k_{33} \frac{\partial T}{\partial z} \right) = f(x, y, z). \quad (1.23)$$

Процессы диффузии вещества во многом аналогичны процессам теплопроводности. При описании диффузии аналогом закона Фурье является закон Нернста, согласно которому вектор плотности потока вещества \mathbf{W} пропорционален градиенту концентрации $N = N(x, y, z)$ [2]:

$$\mathbf{W} = -D \cdot \text{grad}(N), \quad (1.24)$$

где $D = D(x, y, z)$ — коэффициент диффузии.

Плотность потока вещества равна количеству частиц вещества (атомов, молекул), диффундирующему в единицу времени через единичную площадь поверхности.

Выполняя для стационарной задачи диффузии вещества рассуждения, аналогичные приведенным выше для стационарной задачи теплопроводности, получим по аналогии с выражениями (1.8) – (1.14) уравнение Пуассона в векторной форме

$$\operatorname{div}(D(x, y, z) \cdot \operatorname{grad}(N)) = f(x, y, z), \quad (1.25)$$

в частных производных

$$\begin{aligned} \frac{\partial}{\partial x} \left(D(x, y, z) \frac{\partial N}{\partial x} \right) + \frac{\partial}{\partial y} \left(D(x, y, z) \frac{\partial N}{\partial y} \right) + \\ + \frac{\partial}{\partial z} \left(D(x, y, z) \frac{\partial N}{\partial z} \right) = f(x, y, z), \end{aligned} \quad (1.26)$$

или в операторной форме

$$\nabla (D(x, y, z) \cdot \nabla N) = f(x, y, z). \quad (1.27)$$

Для однородной среды ($D(x, y, z) = \text{const}$) аналогично выражениям (1.19), (1.20) можно записать

$$\frac{\partial^2 N}{\partial x^2} + \frac{\partial^2 N}{\partial y^2} + \frac{\partial^2 N}{\partial z^2} = \frac{f(x, y, z)}{D(x, y, z)}, \quad (1.28)$$

или в операторной форме

$$\Delta N = \frac{f(x, y, z)}{D(x, y, z)}. \quad (1.29)$$

К уравнению Пуассона приводят многие другие задачи, например, задача о распределении электростатического потенциала в однородной непроводящей среде в присутствии электрических зарядов.

В общем виде данная задача описывается уравнениями Максвелла

$$\operatorname{rot}(\mathbf{E}) = 0; \quad (1.30)$$

$$\operatorname{div}(\varepsilon(x, y, z)\mathbf{E}) = \frac{\rho(x, y, z)}{\varepsilon_0}, \quad (1.31)$$

где $\mathbf{E} = \mathbf{E}(x, y, z)$ – вектор напряженности электрического поля; $\rho = \rho(x, y, z)$ – объемная плотность электрических зарядов; $\varepsilon =$

$\varepsilon(x, y, z)$ – диэлектрическая проницаемость среды; ε_0 – электрическая постоянная. Уравнение (1.30) выражает отсутствие вихревых электрических полей.

Поскольку напряженность электрического поля \mathbf{E} связана с электрическим потенциалом φ равенством [2]

$$\mathbf{E} = -\text{grad}(\varphi), \quad (1.32)$$

то, подставляя (1.32) в (1.31) и учитывая выражения (1.15) и (1.16), получим уравнение Пуассона в векторной форме

$$\text{div}(\varepsilon(x, y, z) \cdot \text{grad}(\varphi)) = -\frac{\rho(x, y, z)}{\varepsilon_0}, \quad (1.33)$$

в частных производных

$$\begin{aligned} \frac{\partial}{\partial x} \left(\varepsilon(x, y, z) \frac{\partial \varphi}{\partial x} \right) + \frac{\partial}{\partial y} \left(\varepsilon(x, y, z) \frac{\partial \varphi}{\partial y} \right) + \\ + \frac{\partial}{\partial z} \left(\varepsilon(x, y, z) \frac{\partial \varphi}{\partial z} \right) = -\frac{\rho(x, y, z)}{\varepsilon_0}, \end{aligned} \quad (1.34)$$

или в операторной форме

$$\nabla (\varepsilon(x, y, z) \cdot \nabla \varphi) = -\frac{\rho(x, y, z)}{\varepsilon_0}. \quad (1.35)$$

Для однородной среды ($\varepsilon(x, y, z) = \text{const}$) аналогично выражениям (1.19), (1.20) можно записать

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} + \frac{\partial^2 \varphi}{\partial z^2} = -\frac{\rho(x, y, z)}{\varepsilon \varepsilon_0}, \quad (1.36)$$

или в операторной форме

$$\Delta \varphi = -\frac{\rho(x, y, z)}{\varepsilon \varepsilon_0}. \quad (1.37)$$

1.1.2. Уравнение Лапласа

В общем случае уравнение Лапласа имеет вид [2]

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0, \quad (1.38)$$

где $u = u(x, y, z)$ – искомая функция координат.

В операторной форме уравнение Лапласа (1.38) может быть представлено следующим образом:

$$\Delta u = 0, \quad (1.39)$$

где $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ – оператор Лапласа.

Из сопоставления выражений (1.36) и (1.38) видно, что уравнение Лапласа является частным случаем уравнения Пуассона для равной нулю правой части. Покажем это на примерах, приведенных выше.

Рассмотрим задачу о стационарном распределении тепла в некотором объеме V , ограниченном замкнутой поверхностью S трехмерного пространства $\mathbf{x} = (x, y, z)$, которая в векторной форме описывается уравнением (1.14).

Если источники тепла отсутствуют ($f(x, y, z) = 0$) и среда однородна ($k(x, y, z) = \text{const}$), уравнение (1.14) можно переписать в виде

$$\text{div}(\text{grad}(T)) = 0. \quad (1.40)$$

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = 0. \quad (1.41)$$

или в операторной форме

$$\Delta T = 0. \quad (1.42)$$

Процессы диффузии при отсутствии источников диффундирующего вещества ($f(x, y, z) = 0$) и однородной среде ($D(x, y, z) = \text{const}$) описываются уравнением Лапласа в векторной форме

$$\text{div}(\text{grad}(N)) = 0, \quad (1.43)$$

в частных производных

$$\frac{\partial^2 N}{\partial x^2} + \frac{\partial^2 N}{\partial y^2} + \frac{\partial^2 N}{\partial z^2} = 0, \quad (1.44)$$

и в операторной форме

$$\Delta N = 0. \quad (1.45)$$

Задача о распределении электростатического поля в однородной непроводящей среде ($\varepsilon(x, y, z) = \text{const}$) при отсутствии электрических зарядов или когда электрические заряды уравновешены ($\rho(x, y, z) = 0$) описывается уравнениями

$$\operatorname{div}(\mathbf{E}) = 0. \quad (1.46)$$

$$\operatorname{div}(\operatorname{grad}(\varphi)) = 0, \quad (1.47)$$

в частных производных

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} + \frac{\partial^2 \varphi}{\partial z^2} = 0, \quad (1.48)$$

и в операторной форме

$$\Delta \varphi = 0. \quad (1.49)$$

1.2. Параболические уравнения

Рассмотрим задачу матфизики, приводящую к решению параболического уравнения.

1.2.1. Уравнение теплопроводности

Многие нестационарные (т.е. изменяющиеся во времени) физические процессы описываются уравнениями параболического типа. Рассмотрим в качестве примера нестационарное уравнение теплопроводности, которое является более общим случаем уравнения (1.18) и получается на основании закона Фурье (1.7) в результате следующих рассуждений.

Рассмотрим задачу о нестационарном распределении тепла в некотором объеме V , ограниченном замкнутой поверхностью S трехмерного пространства $\mathbf{x} = (x, y, z)$.

Количество тепла q_V , выделившегося в объеме V , ограниченном поверхностью S , за некоторый промежуток времени dt , можно определить как

$$q_v = Q_V dt, \quad (1.50)$$

где Q_V – суммарное количество тепла, выделяющегося в единицу времени в объеме V , ограниченном поверхностью S , определяемое интегралом (1.9).

Учитывая, что рассматривается неравновесное состояние системы, часть тепла $q_T < q_V$ идет на изменение во времени температуры в объеме V и определяется выражением

$$q_T = q dT, \quad (1.51)$$

где q – суммарное количество тепла, необходимого для изменения температуры объема V на один градус; dT – изменение температуры объема V за промежуток времени dt .

Остальная часть тепла q_S протекает через ограничивающую поверхность площадью S :

$$q_s = Q_s dt, \quad (1.52)$$

где Q_S – суммарное количество тепла, протекающего в единицу времени через поверхность S , определяемое интегралом (1.8).

Из закона сохранения энергии следует, что рассматриваемый нестационарный процесс теплопроводности может быть описан уравнением

$$q dT + Q_s dt = Q_v dt. \quad (1.53)$$

Учитывая, что в общем случае неоднородной среды суммарное количество тепла q , необходимого для изменения температуры объема V на один градус, определяется выражением

$$q = \iiint_V \rho(x, y, z) c(x, y, z) dV, \quad (1.54)$$

где $\rho(x, y, z)$ – плотность вещества; $c(x, y, z)$ – удельная теплоемкость вещества, подставляя выражения (1.54), (1.8), (1.9) в уравнение (1.53) и применяя теорему Остроградского – Гаусса (1.11), получим

$$\begin{aligned} \left[\iiint_V \rho(x, y, z) c(x, y, z) \right] dT + \left[\iiint_V \operatorname{div} \mathbf{W} dV \right] dt = \\ = \left[\iiint_V f(x, y, z) dV \right] dt, \end{aligned} \quad (1.55)$$

откуда, разделив левую и правую части на dt , вынося подынтегральные выражения и подставляя уравнение Фурье (1.7), можем

записать нестационарное уравнение теплопроводности в векторной форме:

$$\rho(x, y, z)c(x, y, z)\frac{\partial T}{\partial t} - \operatorname{div}(k(x, y, z)\operatorname{grad}(T)) = f(x, y, z). \quad (1.56)$$

В операторной форме уравнение (1.57) имеет вид

$$\rho(x, y, z)c(x, y, z)\frac{\partial T}{\partial t} - \nabla(k(x, y, z)\nabla(T)) = f(x, y, z). \quad (1.57)$$

1.3. Гиперболические уравнения

Рассмотрим задачу матфизики, приводящую к решению гиперболических уравнений.

1.3.1. Волновое уравнение

Многие физические процессы связаны с возникновением колебаний в некоторой среде. Например, колебания струны, колебания мембраны, распространение звуковых колебаний и др. Они описываются волновым уравнением, относящимся к уравнениям гиперболического типа.

Рассмотрим в качестве примера незатухающие колебания вдоль координаты x физической величины $u(x, t)$ в некоторой среде, описываемые выражением

$$u(x, t) = A \sin(kx - wt + \varphi_0), \quad (1.58)$$

где A – амплитуда колебаний; k – волновое число, определяющее период изменения физической величины $u(x, t)$ по координате; w – круговая (циклическая) частота, определяющая период изменения физической величины $u(x, t)$ во времени t ; s_0 – фаза колебания в точке $x = 0$; φ_0 – начальная фаза колебания.

Волновое число определяется выражением

$$k = \frac{2\pi}{\lambda}, \quad (1.59)$$

где λ – длина волны.

Круговая частота определяется выражением

$$w = \frac{2\pi}{T}, \quad (1.60)$$

где T – период колебаний.

Производная от $u(x, t)$ по координате x равна

$$\frac{\partial u}{\partial x} = Ak \cos(kx - wt + \varphi_0). \quad (1.61)$$

Производная от $u(x, t)$ по времени t равна

$$\frac{\partial u}{\partial t} = -Aw \cos(kx - wt + \varphi_0). \quad (1.62)$$

Вторая производная от $u(x, t)$ по координате x имеет вид

$$\frac{\partial^2 u}{\partial x^2} = -Ak^2 \sin(kx - wt + \varphi_0). \quad (1.63)$$

Вторая производная от $u(x, t)$ по времени t имеет вид

$$\frac{\partial^2 u}{\partial t^2} = -Aw^2 \sin(kx - wt + \varphi_0). \quad (1.64)$$

Сравнивая выражения (1.63) и (1.64) с (1.58), легко увидеть, что

$$\frac{\partial^2 u}{\partial x^2} = -k^2 u(x, t); \quad (1.65)$$

$$\frac{\partial^2 u}{\partial t^2} = -w^2 u(x, t). \quad (1.66)$$

Выражая $u(x, t)$ из (1.65), (1.66) и приравнявая правые части полученных выражений, имеем уравнение в частных производных

$$\frac{1}{w^2} \frac{\partial^2 u}{\partial t^2} = \frac{1}{k^2} \frac{\partial^2 u}{\partial x^2}, \quad (1.67)$$

называемое волновым [6].

В простейшем случае, при $w = 1$ и $k = 1$, получим

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}. \quad (1.68)$$

Обобщая (1.68) для случая трех координат, можем записать волновое уравнение в операторной форме

$$\frac{\partial^2 u}{\partial t^2} = \Delta u. \quad (1.69)$$

Глава 2

Граничные и начальные условия

Из курса высшей математики известно, что дифференциальные уравнения, как правило, имеют бесконечное множество решений. Это связано с появлением в процессе интегрирования констант, при любых значениях которых решение удовлетворяет исходному уравнению [9].

Решение задач матфизики связано с нахождением зависимостей от координат и времени определенных физических величин, которые, безусловно, должны удовлетворять требованиям однозначности¹, конечности² и непрерывности [9]. Иными словами, любая задача матфизики предполагает поиск единственного решения (если оно вообще существует). Поэтому математическая формулировка физической задачи должна, помимо основных уравнений (дифференциальных уравнений в частных производных), описывающих искомые функции внутри рассматриваемой области, включать дополнительные уравнения (дифференциальные или алгебраические), которые определяют искомые функции на границах рассматриваемой области в любой момент времени и во всех внутренних точках области в начальный момент времени. Эти дополнительные уравнения называют соответственно *граничными* и *начальными условиями*

¹Физическая величина не может иметь двух или более различных значений в данной точке пространства и в данный момент времени.

²Физическая величина не может иметь бесконечных по модулю значений.

задачи.

Задачу с начальными и граничными условиями (ГУ) называют краевой. Различают три основных типа краевых задач для дифференциальных уравнений [10]:

- **задача Коши** для нестационарных уравнений: задаются начальные условия для всей заданной области (геометрии), граничные условия отсутствуют;
- **краевая задача** для стационарных уравнений: задаются граничные условия на границе заданной области, начальные условия отсутствуют;
- **смешанная задача** для нестационарных уравнений: задаются и граничные, и начальные условия.

2.1. Граничные условия

Предположим, необходимо решить определенную задачу, описываемую уравнениями матфизики, для некоторой области Θ . Тогда для нахождения единственного решения необходимо задать граничные условия (ГУ), т. е. выразить искомые переменные на границе Ω области Θ некоторыми уравнениями.

Если область Θ представляет собой некоторый объем в трехмерном пространстве, то граница Ω будет представлять собой замкнутую поверхность в этом пространстве, ограничивающую заданный объем. Если область Θ представляет собой некоторую поверхность в двухмерном пространстве, то граница Ω будет представлять собой замкнутый контур в этом пространстве, ограничивающий заданную поверхность. И, наконец, если область Θ представляет собой некоторый отрезок в одномерном пространстве, то граница Ω будет представлять собой две точки на границах заданного отрезка.

Граничные условия в общем виде могут быть представлены следующим образом [10]:

$$\left(\alpha u + \beta \frac{\partial u}{\partial \mathbf{n}} \right) \Big|_{\Omega} = g, \quad (2.1)$$

где $u = u(\mathbf{x}, t)$ – искомая функция; \mathbf{x} – координаты граничной точки в пространстве (например, для трехмерного пространства

$\mathbf{x} = (x, y, z)$; t – время; \mathbf{n} – нормаль к границе Ω ; α, β, g – кусочно-непрерывные функции, заданные на границе Ω , причем $\alpha \geq 0$, $\beta \geq 0$, $\alpha + \beta > 0$

В зависимости от значений α, β, g различают граничные условия первого рода (условия Дирихле), второго рода (условия Неймана) и третьего рода [2, 10].

Граничные условия первого рода, или краевая задача Дирихле, имеют вид ($\alpha = 1, \beta = 0$)

$$\alpha u(\mathbf{x}, t) = g(\mathbf{x}, t). \quad (2.2)$$

Если иметь в виду задачу теплопроводности, то ГУ первого рода задают температуру на границе Ω . В задаче о распределении электростатического поля в непроводящей среде ГУ первого рода задают электрический потенциал на границе Ω и т.д.

Граничные условия второго рода, или краевая задача Неймана, имеют вид ($\alpha = 0, \beta = 1$)

$$\frac{\partial u}{\partial \mathbf{n}} = g(\mathbf{x}, t). \quad (2.3)$$

Иными словами, условия Неймана задают поток на границе, точнее, проекцию вектора потока на внутреннюю нормаль к границе. Например, в задачах теплопроводности ГУ второго рода задают тепловой поток, в задаче о распределении электростатического поля в непроводящей среде – проекцию вектора напряженности электрического поля на нормаль к границе и т.д.

Граничные условия третьего рода являются более общим случаем краевых задач Дирихле и Неймана и имеют вид ($\alpha \geq 0, \beta \geq 0$)

$$\left(\alpha u + \beta \frac{\partial u}{\partial \mathbf{n}} \right) \Big|_{\Omega} = g. \quad (2.4)$$

Например, в тепловых задачах ГУ третьего рода используют для задания на границе конвективного и излучательного теплообмена.

В соответствии с законом Ньютона, плотность теплового потока, отводимого в газовую или жидкую среду (или подводимого из нее) посредством конвекции с поверхности твердого тела в единицу времени, определяется выражением

$$\mathbf{W} = -\alpha(T - T_0), \quad (2.5)$$

где α – коэффициент конвективного теплообмена; T – температура поверхности твердого тела; T_0 – температура окружающей среды [11].

Применяя для плотности теплового потока вдоль нормали к границе закон Фурье 1.7 и приравнивая правые части уравнений 2.5 и 1.7, получим

$$\begin{aligned} -k \frac{\partial T}{\partial n} &= -\alpha(T - T_0); \\ k \frac{\partial T}{\partial n} - \alpha T &= -\alpha T_0, \end{aligned} \quad (2.6)$$

т. е. граничные условия третьего рода (см. выражение 2.4).

В соответствии с законом Стефана – Больцмана, плотность теплового потока, отводимого посредством излучения с поверхности твердого тела в единицу времени, определяется выражением

$$\mathbf{W} = -\varepsilon(T)c_0(T^4 - T_0^4), \quad (2.7)$$

где c_0 – коэффициент лучеиспускания абсолютно черного тела³; $\varepsilon(T)$ – относительная излучательная способность, или степень черноты тела⁴ [11].

Применяя для плотности теплового потока вдоль нормали к границе закон Фурье 1.7 и приравнивая правые части уравнений 2.7 и 1.7, получим условия третьего рода в виде

$$\begin{aligned} -k \frac{\partial T}{\partial n} &= -\varepsilon(T)c_0(T^4 - T_0^4); \\ \frac{k}{\varepsilon(T)} \frac{\partial T}{\partial n} - c_0 T^4 &= -c_0 T_0^4. \end{aligned} \quad (2.8)$$

В уравнениях 2.5 – 2.8 температура является функцией координат $T = T(x, y, z)$ для точек, принадлежащих поверхности тела Ω .

³ Абсолютно черным называют тело, от нагретой поверхности которого не происходит отражения света.

⁴ Отношение излучательных способностей рассматриваемого тела и абсолютно черного тела.

Следует отметить, что количество граничных условий для каждой переменной определяется максимальным порядком производных по координатам в дифференциальных уравнениях [6]: для уравнений первого порядка – одно ГУ, для уравнений второго порядка – два, для уравнений третьего порядка – три ГУ и т. д.

2.2. Начальные условия

Для нахождения единственного решения в задачах, описывающих нестационарные, т.е. изменяющиеся во времени физические процессы, помимо граничных необходимо задавать еще и начальные условия, определяющие значения переменных или их градиентов во всех точках рассматриваемой области Θ в начальный момент времени:

$$u(\mathbf{x}, 0) = \xi(\mathbf{x}) \text{ при } \mathbf{x} \in \Theta; \quad (2.9)$$

$$\frac{\partial u(\mathbf{x}, 0)}{\partial t} = \xi(\mathbf{x}) \text{ при } \mathbf{x} \in \Theta; \quad (2.10)$$

$$\delta(\mathbf{x}) \frac{\partial u(\mathbf{x}, 0)}{\partial t} + \sigma(\mathbf{x}) u(\mathbf{x}, 0) = \xi(\mathbf{x}) \text{ при } \mathbf{x} \in \Theta, \quad (2.11)$$

где $u(\mathbf{x}, 0)$ – искомая функция в начальный момент времени; $\xi(\mathbf{x})$, $\sigma(\mathbf{x})$, $\delta(\mathbf{x})$ – некоторые функции координат.

Аналогично граничным условиям, количество начальных условий для каждой переменной определяется максимальным порядком производной по времени в дифференциальных уравнениях [6].

Глава 3

Нормировка уравнений

Зачастую численное решение систем дифференциальных уравнений в частных производных в исходном виде весьма затруднительно. Причиной этого может быть значительная разница в диапазонах изменения значений искомых функций или коэффициентов, в результате чего возникают недопустимые погрешности округления результатов решения задачи на разрядной сетке компьютера.

Для устранения этой проблемы, а также с целью приведения всех физических величин, входящих в систему, к безразмерному виду, уравнения подвергаются нормировке, суть которой состоит в замене всех переменных и параметров уравнений их безразмерными аналогами, получаемыми делением исходных физических величин на соответствующие нормирующие коэффициенты – константы, размерность которых совпадает с размерностью нормируемой величины.

Значения нормирующих коэффициентов задают таким образом, чтобы минимизировать разницу в диапазонах изменения значений искомых функций. Кроме того, при наличии в исходных уравнениях физических констант с очень большими или очень малыми значениями нормирующие коэффициенты по возможности выражают через эти физические константы с целью их сокращения в процессе нормировки.

В качестве примера приведем процедуру нормировки фундаментальной системы уравнений (ФСУ) полупроводника в диффузионно-дрейфовом приближении. В операторной форме в предположении,

что процессы генерации и рекомбинации электронов и дырок уравновешены, ФСУ может быть записана следующим образом [4, 5]:

$$\frac{\partial n}{\partial t} = -\nabla[\mu_n(n \cdot \nabla\phi - \phi_T \nabla n)]; \quad (3.1)$$

$$\frac{\partial p}{\partial t} = \nabla[\mu_p(p \cdot \nabla\phi + \phi_T \nabla p)]; \quad (3.2)$$

$$\Delta\phi = -\frac{e}{\varepsilon\varepsilon_0}(p - n + N), \quad (3.3)$$

где n – концентрация электронов; p – концентрация дырок; ϕ – электростатический потенциал; N – эффективная концентрация легирующих примесей; μ_n – подвижность электронов; μ_p – подвижность дырок; ε – относительная диэлектрическая проницаемость полупроводника; ε_0 – диэлектрическая проницаемость вакуума; ϕ_T – температурный потенциал; e – элементарный заряд; t – время; ∇ – оператор набла; Δ – оператор Лапласа.

В частных производных для случая одного пространственного измерения система (3.1) – (3.3) имеет вид:

$$\frac{\partial n}{\partial t} = -\frac{\partial}{\partial x} \left[\mu_n \left(n \cdot \frac{\partial\phi}{\partial x} - \phi_T \frac{\partial n}{\partial x} \right) \right]; \quad (3.4)$$

$$\frac{\partial p}{\partial t} = \frac{\partial}{\partial x} \left[\mu_p \left(p \cdot \frac{\partial\phi}{\partial x} + \phi_T \frac{\partial p}{\partial x} \right) \right]; \quad (3.5)$$

$$\frac{\partial^2 \phi}{\partial x^2} = -\frac{e}{\varepsilon\varepsilon_0}(p - n + N). \quad (3.6)$$

В уравнениях (3.4) – (3.6) электростатический потенциал, как правило, изменяется в пределах единиц вольт, а концентрации электронов и дырок могут достигать значений $10^{24} - 10^{25} \text{ м}^{-3}$. Кроме того, в уравнениях используются физические константы, например, элементарный заряд $e = 1,6 \cdot 10^{-19} \text{ Кл}$. Столь значительная разница в диапазонах переменных и значениях физических констант приведет к увеличению погрешности округления результатов на разрядной сетке компьютера. С целью решения данной проблемы можно провести нормировку с использованием системы нормирующих коэффициентов, приведенных в табл. 3.1 [5].

Поскольку в нормированном безразмерном виде переменная определяется как отношение размерной физической величины и нормирующего коэффициента в соответствии с данными табл. 3.1, при

Таблица 3.1. Нормировочные коэффициенты ФСУ

Нормируемая физическая величина	Нормировочный коэффициент
Время	τ_0 – среднее время жизни неосновных носителей в слаболегированной области полупроводника
Потенциал	ϕ_T
Концентрация	n_i – собственная концентрация носителей ¹
Длина, координата	$L_0 = \sqrt{\frac{\varepsilon\varepsilon_0\phi_T}{en_i}}$
Коэффициент диффузии	$D_0 = \frac{L_0^2}{\tau_0}$
Подвижность	$\mu_0 = \frac{D_0}{\phi_T}$

проведении нормировки в уравнениях (3.4) – (3.6) все переменные следует представить произведениями безразмерных величин и соответствующих нормировочных коэффициентов (безразмерные величины отмечены символом *):

$$\frac{\partial n^*}{\partial t^*} \frac{n_i}{\tau_0} = -\frac{\partial}{\partial x^* L_0} \left[\mu_n^* \mu_0 \left(n^* n_i \cdot \frac{\partial \phi^*}{\partial x^*} \cdot \frac{\phi_T}{L_0} - \phi_T \frac{\partial n^*}{\partial x^*} \cdot \frac{n_i}{L_0} \right) \right]; \quad (3.7)$$

$$\frac{\partial p^*}{\partial t^*} \frac{n_i}{\tau_0} = \frac{\partial}{\partial x^* L_0} \left[\mu_p^* \mu_0 \left(p^* n_i \cdot \frac{\partial \phi^*}{\partial x^*} \cdot \frac{\phi_T}{L_0} + \phi_T \frac{\partial p^*}{\partial x^*} \cdot \frac{n_i}{L_0} \right) \right]; \quad (3.8)$$

$$\frac{\partial^2 \phi^*}{\partial x^{*2}} \cdot \frac{\phi_T}{L_0^2} = -\frac{e}{\varepsilon\varepsilon_0} (p^* n_i - n^* n_i + N^* n_i). \quad (3.9)$$

Выражая по табл. 3.1 $\mu_0 = L_0^2/(\phi_T \tau_0)$, легко видеть, что все размерные величины в уравнениях (3.7) – (3.9) сокращаются. В результате получим ФСУ в нормированном виде (опустив символы *,

поскольку все величины в полученных уравнениях безразмерные):

$$\frac{\partial n}{\partial t} = -\frac{\partial}{\partial x} \left[\mu_n \left(n \cdot \frac{\partial \phi}{\partial x} - \frac{\partial n}{\partial x} \right) \right]; \quad (3.10)$$

$$\frac{\partial p}{\partial t} = \frac{\partial}{\partial x} \left[\mu_p \left(p \cdot \frac{\partial \phi}{\partial x} + \frac{\partial p}{\partial x} \right) \right]; \quad (3.11)$$

$$\frac{\partial^2 \phi}{\partial x^2} = n - p - N. \quad (3.12)$$

В результате нормировки диапазон изменения электростатического потенциала увеличился примерно на 2 порядка, а диапазоны изменения концентраций носителей заряда уменьшились на 16 порядков, что существенно уменьшило разницу данных диапазонов. При этом все физические константы, входящие в ФСУ, сократились.

После выполнения численного решения нормированной системы уравнений полученные результаты необходимо денормировать, умножив их на соответствующие нормирующие коэффициенты в соответствии с табл. 3.1.

¹Концентрация электронов проводимости или дырок в беспримесном (собственном) полупроводнике

Глава 4

Координатные сетки и сетки по времени

К сожалению, аналитическое решение уравнений математической физики может быть получено лишь для весьма ограниченного круга задач. В большинстве случаев решение дифференциальных уравнений в частных производных возможно только с использованием численных итерационных методов [1, 2, 3, 4, 5, 6].

Суть данных методов состоит в дискретизации дифференциальных уравнений, т. е. представлении всех или части производных в виде приближенных алгебраических выражений (конечных разностей или конечных элементов), что позволяет преобразовать дифференциальные уравнения в системы алгебраических уравнений. Для этого рассматриваемая область решения задачи Θ покрывается координатной сеткой, а все функции непрерывных аргументов (координат) заменяются сеточными функциями. Иными словами, значения искомых функций исследуются не для всего бесконечно-го множества точек области Θ , а лишь для некоторого конечного подмножества $G \subset \Theta$. Причем при решении нестационарных задач, помимо координатной сетки, вводится сетка по времени.

Число алгебраических уравнений в полученной системе (размерность дискретной задачи) определяется произведением общего числа точек координатной и временной сеток на количество искомых функций в исходной системе дифференциальных уравнений.

4.1. Разновидности сеток

По числу анализируемых пространственных измерений в решаемой системе уравнений координатные сетки могут быть *одномерными*, *двухмерными* и *трехмерными*. Сетки по времени, очевидно, только одномерные.

В свою очередь, двухмерные и трехмерные координатные сетки могут быть прямоугольными или триангулярными.

Прямоугольными называются координатные сетки, элементами которых являются прямоугольники для двух измерений и параллелепипеды для трех измерений.

Триангулярными называются координатные сетки, элементами которых являются треугольники для двух измерений и тетраэдры или призмы для трех измерений.

4.2. Прямоугольные координатные сетки

Рассмотрим одномерную область Θ , представляющую собой отрезок $[0, s]$. Разобьем этот отрезок точками $x_i = ih$, $i = 0, 1, 2, \dots, n$ на n равных частей длины $h = s/n$ каждая. Множество точек $G = \{x_i = ih | i = 0, 1, 2, \dots, n\}$ называется равномерной одномерной координатной сеткой, а число h – шагом сетки [2].

Отрезок $[0, s]$ можно разбить на n частей, вводя произвольные точки

$$0 < x_1 < x_2 < \dots < x_{i-1} < x_i < x_{i+1} < \dots < x_{n-1} < s.$$

Координатная сетка $G = \{x_i | i = 0, 1, 2, \dots, n, x_0 = 0, x_n = s\}$ будет иметь шаг $h_i = x_i - x_{i-1}$, который зависит от номера i узла x_i . Если $h_i \neq h_{i+1}$ хотя бы для одного номера i , координатная сетка G называется неравномерной (рис. 4.1).

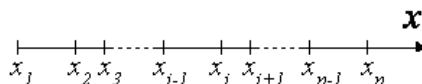


Рис. 4.1. Одномерная координатная сетка

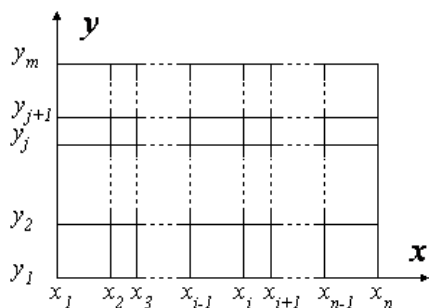


Рис. 4.2. Двумерная координатная сетка

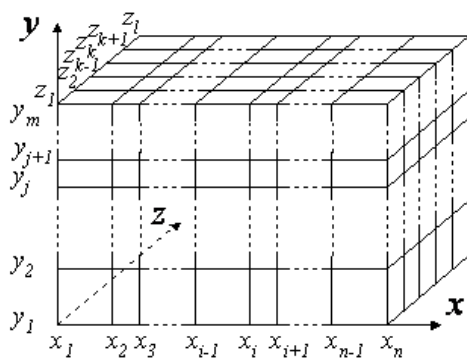


Рис. 4.3. Трехмерная координатная сетка

Аналогично двухмерной равномерной сеткой называют множество точек $G = \{(x_i = ih_1, y_j = jh_2) | i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m\}$. Неравномерная двухмерная сетка представляет собой множество $G = \{(x_i, y_j) | i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m; x_0 = 0, x_n = s_1, y_0 = 0, y_m = s_2\}$ (рис. 4.2).

Соответственно неравномерная трехмерная сетка представляет собой множество $G = \{(x_i, y_j, z_k) | i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m; k = 0, 1, 2, \dots, l; x_0 = 0, x_n = s_1, y_0 = 0, y_m = s_2, z_0 = 0, z_l = s_3\}$ (рис. 4.3).

4.3. Триангулярные координатные сетки

Пусть необходимо покрыть некоторую двумерную область Θ триангулярной координатной сеткой.

Рассмотрим некоторый набор точек $\{p_i | i = 1, 2, \dots, n\}$ в области Θ . Тогда i -я **ячейка Дирихле** Ω_i представляет собой множество всех точек области Θ , лежащих ближе к точке p_i , чем к любой другой точке p_j (рис. 4.4).

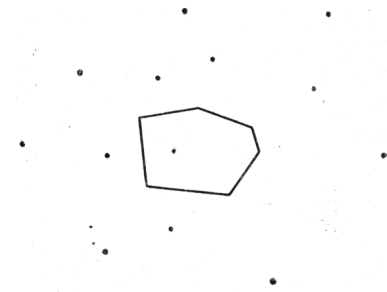


Рис. 4.4. Ячейка Дирихле

Такое множество ячеек $\{\Omega_i | i = 1, 2, \dots, n\}$ называется разбиением Дирихле [7]. Оно полностью покрывает область Θ без наложений (рис. 4.5).

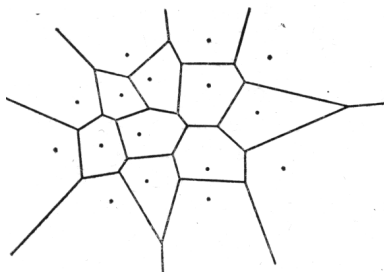


Рис. 4.5. Разбиение Дирихле

Триангулярные координатные сетки получают в результате **триангуляции Делоне** [16], которая определяется следующим образом.

Пусть две точки сетки называются *соседними в смысле Дирихле* тогда и только тогда, когда многоугольники Дирихле, содержащие эти точки, имеют общую грань ненулевой длины.

Триангуляцией Делоне называется граф, образованный соединением соседних в смысле Дирихле точек отрезками прямых линий (рис. 4.6), удовлетворяющий следующим условиям [7]:

1. Вершины любого элемента сетки (треугольника) лежат на некоторой окружности;
2. Не существует точек сетки внутри окружности, описанной около любого элемента;
3. Не существует двух элементов с общей описанной окружностью или, иными словами, не существует более 3 точек сетки, принадлежащих одной окружности.

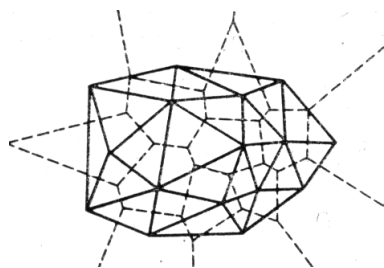


Рис. 4.6. Триангуляция Делоне (сплошные линии) и соответствующее разбиение Дирихле (пунктирные линии)

Из сопоставления приведенных определений видно, что *соответствующие линии разбиения Дирихле и триангуляции Делоне взаимно перпендикулярны* (см. рис. 4.6). Иными словами, построение перпендикуляров в серединах триангуляционных линий дает разбиение Дирихле [7, 16].

Если какое-либо из приведенных выше условий триангуляции Делоне не выполняется, возникают нежелательные проблемы. Например, если четыре или более точек сетки лежат на одной окружности, то они могут быть триангулированы произвольным образом (рис. 4.7). В этом случае триангуляция будет неоднозначной.

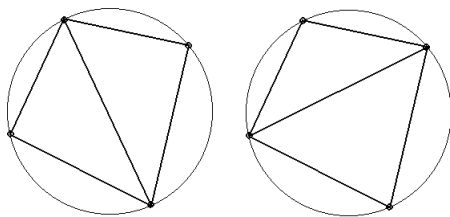


Рис. 4.7. Два варианта триангуляции четырех точек сетки, лежащих на одной окружности)

Если при триангуляции образуются тупоугольные треугольники, это может приводить к наложению ячеек Дирихле (рис. 4.8), что вызовет локальное нарушение законов сохранения при дискретизации дифференциальных уравнений [7].

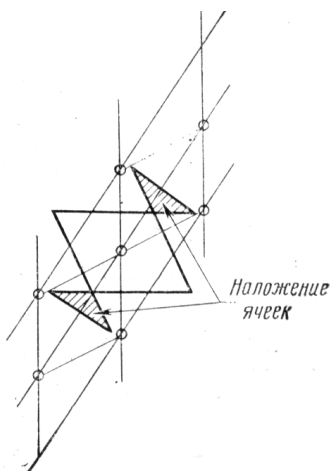


Рис. 4.8. Наложение ячеек Дирихле

Разработано достаточно много различных алгоритмов триангуляции Делоне [7, 16]. Как правило, данные алгоритмы являются рекурсивными, т. е. точки сетки при построении добавляются по одной. На i -м шаге добавляется i -я точка и получается триангуляция Делоне для i точек. При этом предполагается, что предварительно

получена триангуляция Делоне для $i - 1$ точек.

Как правило, используется метод искусственного ограничения [16], состоящий в том, что исходная область Θ (рис. 4.9), на которой предполагается проводить триангуляцию, покрывается искусственной областью Θ_1 , имеющей размеры, значительно превосходящие Θ . Это делается для того, чтобы каждая последующая точка сетки, включая точки, принадлежащие границе области Θ , являлась внутренней, т. е. попадала внутрь области, составленной по предыдущим точкам. Триангуляция Делоне для такой области Θ_1 является исходной для рекурсивного алгоритма.

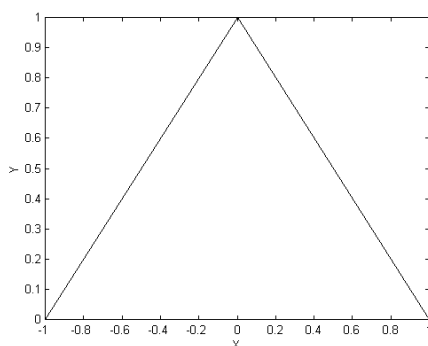


Рис. 4.9. Исходная область для триангуляции

На первом этапе к искусственным угловым четырем точкам границы области Θ_1 добавляются последовательно реальные угловые точки границы области Θ (рис. 4.10).

На втором этапе граничные отрезки области Θ разбиваются на части в соответствии с заданными требованиями к точности (рис. 4.11).

По мере последовательного добавления реальных точек сетки исходная область Θ разобьется на части, как показано на рис. 4.12.

После завершения рекурсивного алгоритма искусственные точки и все связанные с ними отрезки удаляются и получившееся построение считается триангуляцией Делоне для заданного множества точек (рис. 4.13).

В целом рекурсивный алгоритм реализуется в три этапа [7, 16]:

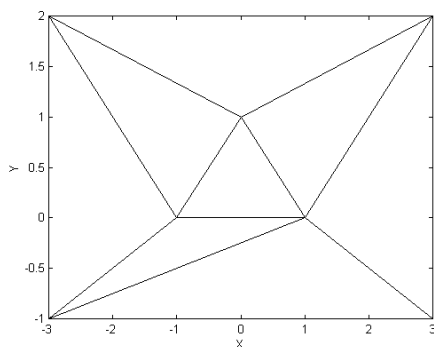


Рис. 4.10. Начальное разбиение искусственной области на элементы с добавлением угловых точек реальной границы

- ранее построенный элемент остается неизменным при добавлении новой точки, если последняя не попадает в описанную около него окружность;
- если новая точка попадает внутрь окружности, описанной около ранее построенного элемента, то новая точка соединяется со всеми его вершинами (при этом ранее построенный элемент расщепляется на части);
- существующая между двумя точками линия сетки устраняется тогда и только тогда, когда новая точка попадает внутрь всех окружностей, описанных около любого ранее построенного элемента, границе которого принадлежит данная линия сетки.

На каждом этапе определяются элементы, подлежащие разбиению, т. е. содержащие новую точку в описанных около них окружностях. Грань, общая для двух таких элементов, отбрасывается, а каждая из оставшихся граней и новая точка определяют новые элементы и линии сетки. Завершает процесс построение линий сетки по полученным элементам [7, 16].

Триангуляцию Делоне можно получить из любой другой триангуляции по тому же множеству точек, последовательно перестраивая пары соседних треугольников ABC и BCD , не удовлетворя-

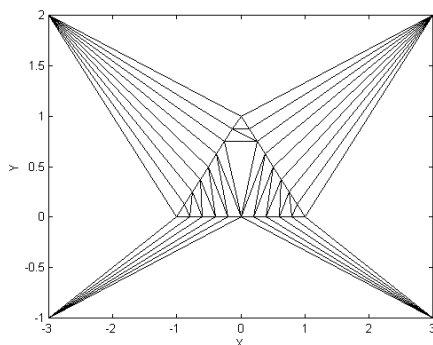


Рис. 4.11. Триангуляция с разбиением отрезков реальной границы на части

ющих условиям Делоне, в пары треугольников ABD и ACD (рис. 4.14) [7, 16].

Такую операцию перестроения часто называют **флипом**. Она позволяет получать триангуляцию Делоне, построив на начальном этапе некоторую произвольную триангуляцию, а затем последовательно улучшая ее в смысле Делоне [7, 16].

При проверке условий Делоне часто используют следующие теоремы [7].

Теорема 1. Триангуляция Делоне обладает максимальной суммой минимальных углов всех своих треугольников среди всех возможных триангуляций.

Теорема 2. Триангуляция Делоне обладает минимальной суммой радиусов окружностей, описанных около треугольников, среди всех возможных триангуляций.

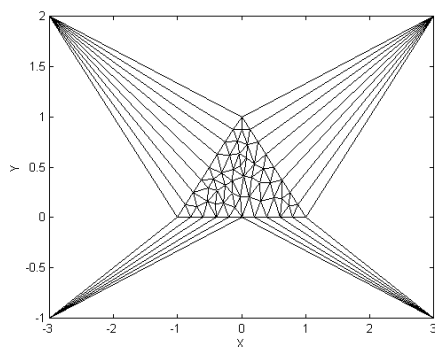


Рис. 4.12. Триангуляция реальной и искусственной областей

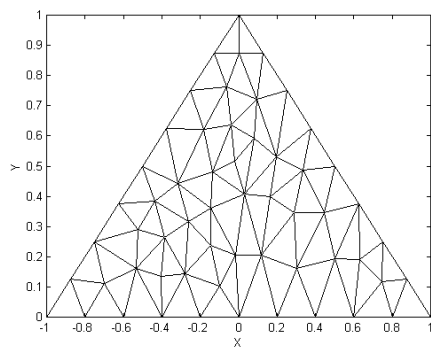


Рис. 4.13. Триангуляция Делоне для заданного множества точек

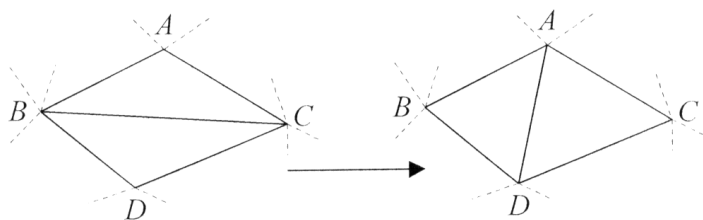


Рис. 4.14. Перестроение треугольников, не удовлетворяющих условиям Делоне

Глава 5

Дискретизация дифференциальных уравнений в частных производных

5.1. Методы дискретизации дифференциальных уравнений

Дискретизация уравнений непрерывной модели представляет собой процедуру преобразования дифференциальных уравнений в частных производных, а также уравнений граничных и начальных условий в системы алгебраических уравнений на координатной и временной сетках. Полученная в результате дискретизации система алгебраических уравнений называется *дискретной моделью*. Данная система алгебраических уравнений может быть *линейной* или *нелинейной* в зависимости от линейности или нелинейности исходной непрерывной модели.

Каждое уравнение непрерывной модели (дифференциальное уравнение, граничное или начальное условие) в процессе дискретизации преобразуется в систему алгебраических уравнений, размерность которой (число уравнений и, соответственно, число искомых

переменных) равно общему числу точек сеток (произведению числа точек по каждому измерению координатной и временной сеток), на которых определено данное уравнение. Например, при дискретизации нестационарной задачи теплопроводности на трехмерной координатной сетке с числом точек $100 \times 50 \times 20$ и сетке по времени с числом точек 10, общее число алгебраических уравнений дискретной модели составит 1 000 000. Если исходная непрерывная модель представляет собой систему дифференциальных уравнений, то число алгебраических уравнений дискретной модели определяется произведением общего числа точек сеток на число искомых функций непрерывной модели.

Существует множество методов дискретизации дифференциальных уравнений: метод конечных разностей, метод конечных элементов, метод взвешенных невязок, метод контрольных объемов и др. Каждый метод имеет свои достоинства и недостатки, определяющие целесообразность его применения для решения конкретной задачи математической физики. Ниже рассмотрены наиболее широко используемые методы: *конечных разностей* и *контрольных объемов*.

5.2. Метод конечных разностей

Метод конечных разностей предполагает поиск решения системы уравнений дискретной модели в виде единой для всей области решения задачи *сеточной функции* — функции, определенной только на конечном множестве точек координатной и временной сеток.

5.2.1. Сеточные функции

В рамках метода конечных разностей введение для области Θ координатной сетки G предполагает, что значения всех переменных и их производных рассматриваются только в узлах этой сетки. С целью выполнения данного требования все переменные задачи заменяются сеточными функциями, а производные любого порядка — конечными разностями [2].

Пусть для некоторой области Θ задана сетка $G = \{(x_i, y_j, z_k) |$

$i = 0, 1, 2, \dots, n, j = 0, 1, 2, \dots, m, k = 0, 1, 2, \dots, l, x_0 = 0, x_n = s_1, y_0 = 0, y_m = s_2, z_0 = 0, z_l = s_3\}$. Тогда функцию $\varphi = \varphi(x_i, y_j, z_k)$, $i = 0, 1, 2, \dots, n, j = 0, 1, 2, \dots, m, k = 0, 1, 2, \dots, l$ дискретного аргумента (x_i, y_j, z_k) называют **сеточной функцией**, определенной на сетке G [2].

Любой непрерывной функции $f(x, y, z)$, заданной в области Θ , можно поставить в соответствие сеточную функцию $\varphi(x_i, y_j, z_k)$ (для удобства обозначают $\varphi_{i,j,k}$), заданную на сетке $G = \{(x_i, y_j, z_k) | i = 0, 1, 2, \dots, n, j = 0, 1, 2, \dots, m, k = 0, 1, 2, \dots, l, x_0 = 0, x_n = s_1, y_0 = 0, y_m = s_2, z_0 = 0, z_l = s_3\}$ (спроектировать функцию $f(x, y, z)$ на сетку G), принимая определенное **правило соответствия**.

Наиболее распространенный вариант правила соответствия выглядит следующим образом:

$$\varphi_{i,j,k} = f(x_i, y_j, z_k). \quad (5.1)$$

В данном случае значения сеточной функции совпадают во всех точках сетки со значениями непрерывной функции.

Иногда используют и более сложные варианты правил соответствия, например:

$$\begin{aligned} \varphi_{i,j,k} = & \frac{1}{(x_{i+1/2} - x_{i-1/2})} \frac{1}{(y_{j+1/2} - y_{j-1/2})} \times \\ & \times \frac{1}{(z_{k+1/2} - z_{k-1/2})} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} \int_{z_{k-1/2}}^{z_{k+1/2}} f(x, y, z) dx dy dz, \end{aligned} \quad (5.2)$$

где $x_{i\pm 1/2}, y_{j\pm 1/2}, z_{k\pm 1/2}$ — координаты серединных точек на соответствующих шагах координатной сетки, определяемые выражениями:

$$x_{i+1/2} = \frac{x_{i+1} + x_i}{2}; \quad x_{i-1/2} = \frac{x_i + x_{i-1}}{2}; \quad (5.3)$$

$$y_{j+1/2} = \frac{y_{j+1} + y_j}{2}; \quad y_{j-1/2} = \frac{y_j + y_{j-1}}{2}; \quad (5.4)$$

$$z_{k+1/2} = \frac{z_{k+1} + z_k}{2}; \quad z_{k-1/2} = \frac{z_k + z_{k-1}}{2}. \quad (5.5)$$

В соответствии с выражением (5.2), значения сеточной функции в узлах сетки задаются средними значениями непрерывной функции в окрестности данного узла, размеры которой определяются серединными точками шагов сетки.

Следует иметь в виду, что одна и та же сеточная функция, заданная на двух различных, но имеющих общие узлы сетках, не обязательно в этих общих узлах будет иметь одни и те же значения [2].

5.2.2. Конечные разности первого порядка

По определению производная функции непрерывного аргумента x в точке x_0 есть предел отношения приращения функции к приращению аргумента, когда последнее стремится к нулю [9]:

$$\frac{df(x)}{dx} = \lim_{(x-x_0) \rightarrow 0} \frac{f(x) - f(x_0)}{x - x_0}. \quad (5.6)$$

Исключая предел из выражения (5.6), производную функции непрерывного аргумента $f(x, y, z)$ можно приближенно заменить (аппроксимировать) разностным выражением, заданным на соответствующей сеточной функции $\varphi(x_i, y_j, z_k)$. Данная аппроксимация может быть осуществлена различными способами [1], например (для первой производной по координате x):

$$\frac{\partial f(x, y, z)}{\partial x} \approx \frac{\varphi_{i+1,j,k} - \varphi_{i,j,k}}{\Delta x_i}; \quad (5.7)$$

$$\frac{\partial f(x, y, z)}{\partial x} \approx \frac{\varphi_{i,j,k} - \varphi_{i-1,j,k}}{\Delta x_{i-1}}; \quad (5.8)$$

$$\frac{\partial f(x, y, z)}{\partial x} \approx \frac{\varphi_{i+1,j,k} - \varphi_{i-1,j,k}}{\Delta x_i + \Delta x_{i-1}}. \quad (5.9)$$

Аналогичным образом аппроксимируются первые производные

по координатам y и z :

$$\frac{\partial f(x, y, z)}{\partial y} \approx \frac{\varphi_{i,j+1,k} - \varphi_{i,j,k}}{\Delta y_j}; \quad (5.10)$$

$$\frac{\partial f(x, y, z)}{\partial y} \approx \frac{\varphi_{i,j,k} - \varphi_{i,j-1,k}}{\Delta y_{j-1}}; \quad (5.11)$$

$$\frac{\partial f(x, y, z)}{\partial y} \approx \frac{\varphi_{i,j+1,k} - \varphi_{i,j-1,k}}{\Delta y_j + \Delta y_{j-1}}; \quad (5.12)$$

$$\frac{\partial f(x, y, z)}{\partial z} \approx \frac{\varphi_{i,j,k+1} - \varphi_{i,j,k}}{\Delta z_k}; \quad (5.13)$$

$$\frac{\partial f(x, y, z)}{\partial z} \approx \frac{\varphi_{i,j,k} - \varphi_{i,j,k-1}}{\Delta z_{k-1}}; \quad (5.14)$$

$$\frac{\partial f(x, y, z)}{\partial z} \approx \frac{\varphi_{i,j,k+1} - \varphi_{i,j,k-1}}{\Delta z_k + \Delta z_{k-1}}, \quad (5.15)$$

где $\Delta x_i, \Delta y_j, \Delta z_k, \Delta x_{i-1}, \Delta y_{j-1}, \Delta z_{k-1}$ — шаги координатной сетки, определяемые выражениями:

$$\Delta x_i = x_{i+1} - x_i; \quad \Delta x_{i-1} = x_i - x_{i-1}; \quad (5.16)$$

$$\Delta y_j = y_{j+1} - y_j; \quad \Delta y_{j-1} = y_j - y_{j-1}; \quad (5.17)$$

$$\Delta z_k = z_{k+1} - z_k; \quad \Delta z_{k-1} = z_k - z_{k-1}. \quad (5.18)$$

Выражения (5.7), (5.10), (5.13) называют правыми разностями, (5.8), (5.11), (5.14) — левыми разностями, (5.9), (5.12), (5.15) — центральными разностями.

Каждому из преобразований свойственна определенная погрешность аппроксимации, стремящаяся к нулю при стремлении к нулю шага сетки.

5.2.3. Конечные разности второго порядка

Производные второго порядка могут быть аппроксимированы следующим образом:

$$\begin{aligned} \frac{\partial^2 f(x, y, z)}{\partial x^2} &= \frac{\frac{\varphi_{i+1,j,k} - \varphi_{i,j,k}}{\Delta x_i} - \frac{\varphi_{i,j,k} - \varphi_{i-1,j,k}}{\Delta x_{i-1}}}{\frac{\Delta x_i + \Delta x_{i-1}}{2}} = \\ &= \frac{2}{\Delta x_i + \Delta x_{i-1}} \left(\frac{\varphi_{i+1,j,k} - \varphi_{i,j,k}}{\Delta x_i} - \frac{\varphi_{i,j,k} - \varphi_{i-1,j,k}}{\Delta x_{i-1}} \right); \end{aligned} \quad (5.19)$$

$$\begin{aligned} \frac{\partial^2 f(x, y, z)}{\partial y^2} &= \\ &= \frac{2}{\Delta y_j + \Delta y_{j-1}} \left(\frac{\varphi_{i,j+1,k} - \varphi_{i,j,k}}{\Delta y_j} - \frac{\varphi_{i,j,k} - \varphi_{i,j-1,k}}{\Delta y_{j-1}} \right); \end{aligned} \quad (5.20)$$

$$\begin{aligned} \frac{\partial^2 f(x, y, z)}{\partial z^2} &= \\ &= \frac{2}{\Delta z_k + \Delta z_{k-1}} \left(\frac{\varphi_{i,j,k+1} - \varphi_{i,j,k}}{\Delta z_k} - \frac{\varphi_{i,j,k} - \varphi_{i,j,k-1}}{\Delta z_{k-1}} \right). \end{aligned} \quad (5.21)$$

В случае равномерной сетки, т.е. при $\Delta x_i = \Delta x_{i-1} = \Delta x$, $\Delta y_j = \Delta y_{j-1} = \Delta y$, $\Delta z_k = \Delta z_{k-1} = \Delta z$, выражения (5.19) – (5.21) примут более простой вид:

$$\frac{\partial^2 f(x, y, z)}{\partial x^2} \approx \frac{\varphi_{i+1,j,k} - 2\varphi_{i,j,k} + \varphi_{i-1,j,k}}{\Delta x^2}; \quad (5.22)$$

$$\frac{\partial^2 f(x, y, z)}{\partial y^2} \approx \frac{\varphi_{i,j+1,k} - 2\varphi_{i,j,k} + \varphi_{i,j-1,k}}{\Delta y^2}; \quad (5.23)$$

$$\frac{\partial^2 f(x, y, z)}{\partial z^2} \approx \frac{\varphi_{i,j,k+1} - 2\varphi_{i,j,k} + \varphi_{i,j,k-1}}{\Delta z^2}. \quad (5.24)$$

5.2.4. Смешанные конечные разности

Аппроксимирующие выражения для смешанных производных могут быть получены следующим образом:

$$\begin{aligned} \frac{\partial^2 f(x, y, z)}{\partial x \partial x} &\approx \frac{\frac{\varphi_{i+1,j+1,k} - \varphi_{i,j+1,k}}{\Delta x_i} - \frac{\varphi_{i+1,j,k} - \varphi_{i,j,k}}{\Delta x_i}}{\Delta y_j} = \\ &= \frac{\varphi_{i+1,j+1,k} - \varphi_{i,j+1,k} - \varphi_{i+1,j,k} + \varphi_{i,j,k}}{\Delta x_i \Delta y_j}; \end{aligned} \quad (5.25)$$

$$\begin{aligned} \frac{\partial^2 f(x, y, z)}{\partial y \partial z} &\approx \frac{\frac{\varphi_{i,j+1,k+1} - \varphi_{i,j,k+1}}{\Delta y_j} - \frac{\varphi_{i,j+1,k} - \varphi_{i,j,k}}{\Delta y_j}}{\Delta z_k} = \\ &= \frac{\varphi_{i,j+1,k+1} - \varphi_{i,j,k+1} - \varphi_{i,j+1,k} + \varphi_{i,j,k}}{\Delta y_j \Delta z_k}; \end{aligned} \quad (5.26)$$

$$\begin{aligned} \frac{\partial^2 f(x, y, z)}{\partial x \partial z} &\approx \frac{\frac{\varphi_{i+1,j,k+1} - \varphi_{i,j,k+1}}{\Delta x_i} - \frac{\varphi_{i+1,j,k} - \varphi_{i,j,k}}{\Delta x_i}}{\Delta z_k} = \\ &= \frac{\varphi_{i+1,j,k+1} - \varphi_{i,j,k+1} - \varphi_{i+1,j,k} + \varphi_{i,j,k}}{\Delta x_i \Delta z_k}. \end{aligned} \quad (5.27)$$

Аналогичным образом могут быть получены аппроксимирующие выражения для производных более высоких порядков [2].

5.2.5. Шаблоны

Для наглядного представления конечно-разностных аппроксимаций используют шаблоны [6].

Шаблон представляет собой граф, символически отображающий участок сетки, на котором производится аппроксимация функций и их производных. Вершины графа символизируют точки сетки с индексами (i, j, k) , $(i \pm 1, j \pm 1, k \pm 1)$, $(i \pm 2, j \pm 2, k \pm 2)$ и т. д. в зависимости от вида аппроксимирующих выражений.

Например, для правой разности

$$\frac{df(x)}{dx} \approx \frac{\varphi_{i+1} - \varphi_i}{\Delta x_i} \quad (5.28)$$

вычислительный шаблон будет иметь вид, представленный на рис. 5.1.

Для левой разности

$$\frac{df(x)}{dx} \approx \frac{\varphi_i - \varphi_{i-1}}{\Delta x_{i-1}} \quad (5.29)$$

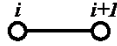


Рис. 5.1. Шаблон для аппроксимации 5.28

вычислительный шаблон будет иметь вид, представленный на рис. 5.2.

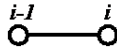


Рис. 5.2. Шаблон для аппроксимации 5.29

Шаблон для разности второго порядка

$$\frac{d^2 f(x)}{dx^2} \approx \frac{2}{\Delta x_i + \Delta x_{i-1}} \left(\frac{\varphi_{i+1} - \varphi_i}{\Delta x_i} - \frac{\varphi_i - \varphi_{i-1}}{\Delta x_{i-1}} \right) \quad (5.30)$$

будет иметь вид, представленный на рис. 5.3.

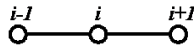


Рис. 5.3. Шаблон для аппроксимации 5.30

Шаблон для аппроксимации (5.25) смешанной производной второго порядка представлен на рис. 5.4.

Шаблон для аппроксимаций производных второго порядка (5.19) – (5.21) представлен на рис. 5.5.

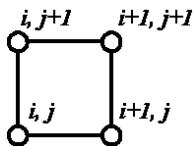


Рис. 5.4. Шаблон для аппроксимации 5.25

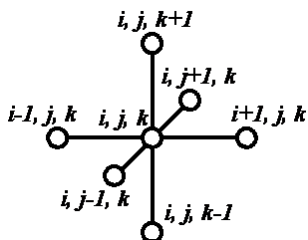


Рис. 5.5. Шаблон для аппроксимации (5.19) – (5.21)

5.2.6. Пример конечно-разностного представления дифференциальных уравнений

Рассмотрим в качестве примера конечно-разностное представление уравнений нестационарной задачи теплопроводности:

$$\rho(x, y, z)C(x, y, z)\frac{\partial T}{\partial t} - \frac{\partial}{\partial x} \left(\lambda(x, y, z)\frac{\partial T}{\partial x} \right) - \frac{\partial}{\partial y} \left(\lambda(x, y, z)\frac{\partial T}{\partial y} \right) - \frac{\partial}{\partial z} \left(\lambda(x, y, z)\frac{\partial T}{\partial z} \right) = f(x, y, z); \quad (5.31)$$

$$T(x_{min}, y, z, t) = g(y, z, t); \quad (5.32)$$

$$T(x_{max}, y, z, t) = q(y, z, t); \quad (5.33)$$

$$T(x, y_{min}, z, t) = w(x, z, t); \quad (5.34)$$

$$T(x, y_{max}, z, t) = r(x, z, t); \quad (5.35)$$

$$\left. \frac{\partial T}{\partial z} \right|_{x, y, z_{min}, t} = h(x, y, t); \quad (5.36)$$

$$\left. \frac{\partial T}{\partial z} \right|_{x, y, z_{max}, t} = v(x, y, t); \quad (5.37)$$

$$T(x, y, z, t_{min}) = p(x, y, z); \quad (5.38)$$

где $\rho(x, y, z)$ — плотность вещества; $C(x, y, z)$ — удельная теплоемкость вещества; $\lambda(x, y, z)$ — коэффициент теплопроводности; x, y, z — координаты; t — время; T — абсолютная температура; $f(x, y, z)$ — плотность мощности источников тепла; $g(y, z, t)$, $q(y, z, t)$, $w(x, z, t)$, $r(x, z, t)$, $h(x, y, t)$, $v(x, y, t)$, $p(x, y, z)$ — известные из условия задачи непрерывные функции координат и времени.

Выражения (5.32) – (5.35) являются граничными условиями первого рода, выражения (5.36), (5.37) — граничными условиями второго рода, выражение (5.38) — начальным условием.

Введем координатную сетку

$$G_{xyz} = \{(x_i, y_i, z_i) | i = \overline{1, I}; j = \overline{1, J}; k = \overline{1, K}\} \quad (5.39)$$

и сетку по времени

$$G_t = \{t_m | m = \overline{1, M}\}, \quad (5.40)$$

где i, j, k, m — индексы узлов сеток; I, J, K, M — число точек сеток по соответствующим измерениям.

Вводя конечно-разностные аппроксимации уравнений (5.31) – (5.38), получим систему линейных алгебраических уравнений:

$$\begin{aligned}
 & \rho_{i,j,k} C_{i,j,k} \frac{T_{i,j,k,m} - T_{i,j,k,m-1}}{\Delta t_{m-1}} - \frac{2}{\Delta x_i + \Delta x_{i-1}} \times \\
 & \times \left(\lambda_{i,j,k} \frac{T_{i+1,j,k,m} - T_{i,j,k,m}}{\Delta x_i} - \lambda_{i-1,j,k} \frac{T_{i,j,k,m} - T_{i-1,j,k,m}}{\Delta x_{i-1}} \right) - \\
 & - \frac{2}{\Delta y_j + \Delta y_{j-1}} \times \\
 & \times \left(\lambda_{i,j,k} \frac{T_{i,j+1,k,m} - T_{i,j,k,m}}{\Delta y_j} - \lambda_{i,j-1,k} \frac{T_{i,j,k,m} - T_{i,j-1,k,m}}{\Delta y_{j-1}} \right) - \\
 & - \frac{2}{\Delta z_k + \Delta z_{k-1}} \times \\
 & \times \left(\lambda_{i,j,k} \frac{T_{i,j,k+1,m} - T_{i,j,k,m}}{\Delta z_k} - \lambda_{i,j,k-1} \frac{T_{i,j,k,m} - T_{i,j,k-1,m}}{\Delta z_{k-1}} \right) = f_{i,j,k}; \\
 & i = \overline{2, I-1}; \quad j = \overline{2, J-1}; \quad k = \overline{2, K-1};
 \end{aligned} \tag{5.41}$$

$$T_{1,j,k,m} = g_{j,k,m}; \quad j = \overline{1, J}; \quad k = \overline{1, K}; \quad m = \overline{2, M}; \tag{5.42}$$

$$T_{I,j,k,m} = q_{j,k,m}; \quad j = \overline{1, J}; \quad k = \overline{1, K}; \quad m = \overline{2, M}; \tag{5.43}$$

$$T_{i,1,k,m} = w_{i,k,m}; \quad i = \overline{2, I-1}; \quad k = \overline{2, K-1}; \quad m = \overline{2, M}; \tag{5.44}$$

$$T_{i,J,k,m} = r_{i,k,m}; \quad i = \overline{2, I-1}; \quad k = \overline{2, K-1}; \quad m = \overline{2, M}; \tag{5.45}$$

$$\begin{aligned}
 & \frac{T_{i,j,2,m} - T_{i,j,1,m}}{\Delta z_1} = h_{i,j,1,m}; \quad i = \overline{2, I-1}; \quad j = \overline{2, J-1}; \\
 & m = \overline{2, M};
 \end{aligned} \tag{5.46}$$

$$\begin{aligned}
 & \frac{T_{i,j,K,m} - T_{i,j,K-1,m}}{\Delta z_{K-1}} = v_{i,j,K,m}; \quad i = \overline{2, I-1}; \quad j = \overline{2, J-1}; \\
 & m = \overline{2, M};
 \end{aligned} \tag{5.47}$$

$$T_{i,j,k,1} = p_{i,j,k}; \quad i = \overline{2, I-1}; \quad j = \overline{2, J-1}; \quad k = \overline{2, K-1}. \tag{5.48}$$

После каждого из конечно-разностных выражений (5.41) – (5.48) приведены индексы точек сеток, на которых определены данные выражения. Общее число уравнений в системе (5.41) – (5.48) составляет $I \times J \times K \times M$.

5.3. Метод контрольных объемов

Основная идея *метода контрольных объемов* поддается прямой физической интерпретации и заключается в следующем. Расчетную область разбивают на некоторое число непересекающихся подобластей (*контрольных объемов*) таким образом, что каждая узловая точка содержится в одном контрольном объеме. Дифференциальное уравнение интегрируют по каждому контрольному объему. Для вычисления интегралов используют кусочные профили, которые описывают изменение функций, входящих в состав уравнений, между узловыми точками. В результате находят дискретный аналог дифференциального уравнения, в который входят значения функций в нескольких узловых точках [?].

Полученный подобным образом дискретный аналог выражает закон сохранения искомых функций для конечного контрольного объема точно так же, как дифференциальное уравнение выражает закон сохранения для бесконечно малого контрольного объема. Одним из важных свойств метода контрольных объемов является то, что в нем заложено точное интегральное сохранение таких величин, как масса, количество движения и энергия на любой группе контрольных объемов и, следовательно, на всей расчетной области. Это свойство проявляется при любом числе узловых точек, а не только в предельном случае очень большого их числа. Таким образом, даже решение на достаточно грубой сетке удовлетворяет условию интегрального баланса [?].

Применим метод контрольных объемов для дискретизации уравнения Пуассона на двухмерной триангулярной сетке.

5.3.1. Пример дискретизации дифференциального уравнения с использованием метода контрольных объемов на триангулярной координатной сетке

Для дискретизации уравнений математической физики, прежде всего, в соответствии с условиями задачи строится множество точек координатной сетки, проводится триангуляция Делоне и разбиение Дирихле. Дальнейшие шаги рассмотрим на примере задачи о распределении электростатического поля в области Θ объемом

V , ограниченной замкнутой поверхностью Ω с площадью S , непроводящей среды при наличии электрических зарядов, описываемой уравнением

$$\nabla (\varepsilon(x, y, z) \nabla \varphi) = -\frac{1}{\varepsilon_0} \rho(x, y, z), \quad (5.49)$$

где φ – электростатический потенциал; ρ – объемная плотность электрических зарядов; $\varepsilon(x, y, z)$ – относительная диэлектрическая проницаемость среды; ε_0 – диэлектрическая проницаемость вакуума.

Интегрируя обе части уравнения (5.49) по объему V , получим

$$\iiint_V \nabla (\varepsilon(x, y, z) \nabla \varphi) dV + \frac{1}{\varepsilon_0} \iiint_V \rho(x, y, z) dV = 0. \quad (5.50)$$

Применив теорему Остроградского – Гаусса, перепишем уравнение (5.50) в виде

$$\oint_S \varepsilon(x, y, z) \nabla \varphi dS + \frac{1}{\varepsilon_0} \iiint_V \rho(x, y, z) dV = 0. \quad (5.51)$$

Уравнение (5.51) представляет собой интегродифференциальную форму уравнения (5.49) и фактически выражает законы сохранения для области решения задачи.

Аппроксимируя уравнение (5.51) для некоторой внутренней ячейки Дирихле Θ_0 с внешней границей D , построенной вокруг точки p_0 сетки (рис. 5.6) в предположении, что все переменные в пределах данной ячейки Дирихле неизменны и триангулярная координатная сетка двумерна (все ячейки разбиения Дирихле представляют собой призмы равной высоты H , которая может быть вынесена за знаки интегралов), тогда получим следующее уравнение:

$$H \oint_D \varepsilon(x, y, z) \nabla \varphi dl + \frac{H}{\varepsilon_0} \iint_{S_0} \rho(x, y, z) dS_0 = 0, \quad (5.52)$$

где dl – элемент контура D ; dS_0 – элемент площади ячейки Дирихле, построенной вокруг точки p_0 (см. рис. 5.6).

Сокращая уравнение (5.52) на H , аппроксимируя интеграл по замкнутому контуру рассматриваемой ячейки Дирихле суммой по

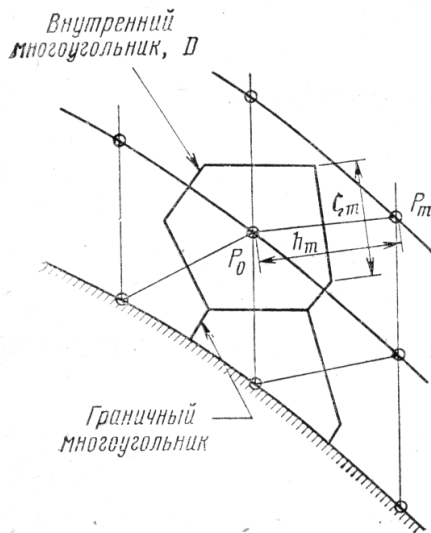


Рис. 5.6. Параметры m -й грани ячейки Дирихле

граням данной ячейки и считая объемную плотность заряда ρ неизменной в пределах ячейки Дирихле (иными словами, вводя ступенчатую аппроксимацию функции объемной плотности заряда), получим

$$\sum_{m=1}^{M_0} \xi_m \frac{\varepsilon_{P0} + \varepsilon_{Pm}}{2} \frac{\varphi_m - \varphi_0}{h_m} + \frac{1}{\varepsilon_0} \rho_0 S_0 = 0, \quad (5.53)$$

где M_0 – число граней ячейки Дирихле с контуром D ; S_0 – площадь ячейки Дирихле, построенной вокруг точки p_0 (см. рис. 5.6); φ_0, φ_m – значения электрического потенциала в точках p_0 и p_m соответственно; $\varepsilon_{P0}, \varepsilon_{Pm}$ – значения диэлектрической проницаемости среды в точках p_0 и p_m соответственно; ρ_0 – значение объемной плотности электрических зарядов в точке p_0 ; h_m – расстояние между точками p_0 и p_m ; ξ_m – длина m -й грани ячейки Дирихле.

Обобщая уравнение (5.53) для всех внутренних ячеек Дирихле области Θ , можно записать

$$\sum_{m=1}^{M_0} \xi_m \frac{\varepsilon_{P0} + \varepsilon_{Pm}}{2} \frac{\varphi_m - \varphi_i}{h_m} + \frac{1}{\varepsilon_0} \rho_i S_i = 0, \quad i = \overline{1, N_{int}}, \quad (5.54)$$

где i – номер ячейки Дирихле; N_{int} – число внутренних ячеек Дирихле.

Аппроксимация уравнений для граничных ячеек Дирихле (см. рис. 5.6) производится аналогичным образом, но с учетом соответствующих граничных условий Дирихле или Неймана. Например, если на границе заданы условия второго рода

$$\frac{\partial \varphi(x, y, z)}{\partial \mathbf{n}} = g(x, y, z), \quad \text{при } (x, y, z) \in \Omega, \quad (5.55)$$

где \mathbf{n} – нормаль к поверхности Ω ; $g(x, y, z)$ – некоторая известная из условия задачи непрерывная функция координат. Выражения для соответствующих граничных ячеек Дирихле могут быть записаны в виде

$$\sum_{m=1}^{M_\Omega} \xi_m \varepsilon_{Pm} g_m + \sum_{m=1+M_\Omega}^{M_i} \xi_m \frac{\varepsilon_{P0} + \varepsilon_{Pm}}{2} \frac{\varphi_m - \varphi_i}{h_m} + \frac{1}{\varepsilon_0} \rho_i S_i = 0, \quad i = \overline{1, N_{ext}}, \quad (5.56)$$

где M_Ω – число граней ячейки Дирихле, совпадающих с поверхностью Ω ; N_{ext} – число граничных ячеек Дирихле, т. е. ячеек, у которых хотя бы одна грань совпадает с поверхностью Ω (см. рис. 5.6); g_m – значение функции $g(x, y, z)$ на m -й грани ячейки Дирихле.

Нетрудно видеть, что система алгебраических уравнений (5.54), (5.56), полученная в результате дискретизации уравнений (5.52), (5.55) на триангулярной координатной сетке с использованием метода контрольных объемов, является линейной. Число уравнений в системе определяется общим числом точек сетки $N = N_{int} + N_{ext}$.

Глава 6

Решение систем алгебраических уравнений

Выбор метода решения системы алгебраических уравнений, полученной в результате дискретизации уравнений математической физики, определяется ее размерностью и характером (линейный или нелинейный). Для решения систем линейных алгебраических уравнений (СЛАУ) широко используют матричный метод, метод исключения Гаусса, метод LU-разложения и др. [1]. Использование данных методов предпочтительно, если размерность системы не слишком велика для имеющегося объема оперативной памяти компьютера, на котором предполагается решать поставленную задачу.

Для решения линейных систем больших размерностей и систем нелинейных алгебраических уравнений используют итерационные методы решения, позволяющие оптимизировать процесс решения в зависимости от имеющихся вычислительных ресурсов компьютера. При этом получается приближенное решение. Максимальная точность ограничивается допустимым временем решения задачи и разрядной сеткой компьютера [2].

6.1. Методы решения систем линейных алгебраических уравнений

6.1.1. Классификация методов решения систем линейных алгебраических уравнений

Существует большое число различных методов решения *систем линейных алгебраических уравнений (СЛАУ)*. Все эти методы можно разделить на две группы:

- *прямые методы*, позволяющие найти решение СЛАУ без методической погрешности. Погрешность решения определяется лишь погрешностью округления результатов на разрядной сетке компьютера. Использование прямых методов затруднительно при решении систем уравнений большой размерности, поскольку требует сравнительно больших ресурсов оперативной памяти;
- *итерационные методы*, обеспечивающие последовательное нахождение элементов решения СЛАУ, что позволяет значительно экономить ресурсы оперативной памяти и благодаря этому решать системы уравнений большой размерности. Помимо погрешности округления результатов на разрядной сетке компьютера, итерационные методы вносят дополнительно определенную методическую погрешность, зависящую от числа итераций.

К прямым методам решения СЛАУ относят *матричный метод, метод исключения Гаусса, метод LU-разложения* и др. [1].

Наиболее широко используемыми итерационными методами решения СЛАУ являются *итерация Якоби* и *итерация Гаусса-Зейделя* [6].

6.1.2. Матричный метод

В общем случае СЛАУ из n уравнений от n переменных имеет вид

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2; \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n. \end{cases} \quad (6.1)$$

или в матричном представлении

$$\mathbf{A}\mathbf{x} = \mathbf{B}, \quad (6.2)$$

где \mathbf{A} – матрица коэффициентов СЛАУ размером $n \times n$; \mathbf{x} – вектор-столбец искомых переменных; \mathbf{B} – вектор-столбец свободных членов.

Решение системы (6.2) может быть найдено матричным методом, т.е. умножением вектор-столбца свободных членов \mathbf{B} на матрицу \mathbf{A}^{-1} , обратную матрице коэффициентов [9]

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{B}. \quad (6.3)$$

Нахождение матрицы, обратной матрице коэффициентов, характеризуется вычислительной сложностью $O(n^3)$ и при решении СЛАУ большой размерности требует значительного объема оперативной памяти компьютера.

6.1.3. Методы прямой и обратной подстановки

Методы прямой и обратной подстановки являются наиболее простыми методами решения СЛАУ. Они характеризуются вычислительной сложностью $O(n)$, но могут быть применены только для систем линейных алгебраических уравнений, приведенных к специальному «треугольному» виду.

Матрица коэффициентов \mathbf{A} называется *верхней треугольной*, если $a_{ij} = 0$ для всех $i > j$:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1; \\ \phantom{a_{11}x_1 +} a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2; \\ \phantom{a_{11}x_1 +} \phantom{a_{22}x_2 +} a_{33}x_3 + \cdots + a_{3n}x_n = b_3; \\ \phantom{a_{11}x_1 +} \phantom{a_{22}x_2 +} \phantom{a_{33}x_3 +} \cdots \\ \phantom{a_{11}x_1 +} \phantom{a_{22}x_2 +} \phantom{a_{33}x_3 +} a_{nn}x_n = b_n. \end{cases} \quad (6.4)$$

В случае, если $a_{ij} = 0$ для всех $i < j$, матрица \mathbf{A} называется нижней треугольной:

$$\begin{cases} a_{11}x_1 & = b_1; \\ a_{21}x_1 + a_{22}x_2 & = b_2; \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 & = b_3; \\ \dots & \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \dots + a_{nn}x_n & = b_n. \end{cases} \quad (6.5)$$

Соответственно, система линейных алгебраических уравнений (6.4) с верхней треугольной матрицей коэффициентов называется *верхней треугольной системой*, а (6.5) – *нижней треугольной системой* [3].

Можно показать, что если $\mathbf{Ax} = \mathbf{B}$ – верхняя (или нижняя) треугольная система и не существует равных нулю коэффициентов главной диагонали матрицы

$$a_{ii} \neq 0, \quad i = 1, 2, \dots, n, \quad (6.6)$$

то система имеет единственное решение [3].

Это решение может быть найдено методом прямой подстановки в случае нижней треугольной системы или обратной подстановки в случае верхней треугольной системы. Например, для верхней треугольной системы (6.4) n -е уравнение имеет лишь одну переменную x_n , которую из него можно легко найти следующим образом:

$$x_n = \frac{b_n}{a_{nn}}. \quad (6.7)$$

Используя (6.7), из $(n-1)$ -го уравнения находим

$$x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}. \quad (6.8)$$

Продолжая делать подобные подстановки далее, получим решение, которое в общем виде может быть записано следующим образом:

$$x_i = \frac{b_i - \sum_{j=i+1}^n a_{ij}x_j}{a_{ii}}, \quad i = n-1, n-2, \dots, 2, 1. \quad (6.9)$$

Аналогично метод прямой подстановки (начиная с 1-го уравнения) позволит найти решения для нижней треугольной системы.

6.1.4. Метод исключения Гаусса

Метод исключения Гаусса предполагает приведение системы линейных уравнений (6.1) к верхней треугольной системе (6.4) или нижней треугольной системе (6.5) и последующее нахождение решения, соответственно, методами обратной или прямой подстановки.

Две системы линейных алгебраических уравнений **эквивалентны**, если они имеют одинаковое множество решений [3].

Приведение СЛАУ к «треугольному» виду в рамках метода исключения Гаусса выполняется с использованием трех операций, преобразующих исходную систему в эквивалентную ей:

- 1) **перестановка** – произвольное изменение порядка уравнений в системе;
- 2) **масштабирование** – умножение левой и правой части любого уравнения системы на любую, отличную от нуля, константу;
- 3) **замещение** – замена любого уравнения системы почленной суммой этого же уравнения и любого другого уравнения системы, умноженного на отличную от нуля константу.

Перечисленные операции выполняются до тех пор, пока полученная система, эквивалентная исходной, не будет треугольной. Вычислительная сложность метода исключения Гаусса составляет $O(n^3)$.

Проиллюстрируем это на следующем примере [3].

Пусть необходимо найти параболу

$$y = A + Bx + Cx^2, \quad (6.10)$$

проходящую через три точки $(1, 1)$, $(2, -1)$, $(3, 1)$.

Последовательно подставив координаты точек в выражение (6.10), найдем для каждой точки уравнение, связывающее значения x и y .

В результате получим систему линейных уравнений, в которой искомыми переменными являются коэффициенты полинома (6.10):

$$\begin{cases} A + B + C = 1; \\ A + 2B + 4C = -1; \\ A + 3B + 9C = 1. \end{cases} \quad (6.11)$$

Производим замещение третьего уравнения разностью третьего и первого уравнений, а второго уравнения – разностью второго и первого уравнений. В результате из второго и третьего уравнений исключается переменная A :

$$\begin{cases} A + B + C = 1; \\ B + 3C = -2; \\ 2B + 8C = 0. \end{cases} \quad (6.12)$$

Производим замещение третьего уравнения системы (6.12) разностью третьего уравнения и второго уравнения, умноженного на 2. При этом из третьего уравнения исключается переменная B :

$$\begin{cases} A + B + C = 1; \\ B + 3C = -2; \\ 2C = 4. \end{cases} \quad (6.13)$$

В результате получили верхнюю треугольную систему (6.13), из которой методом обратной подстановки находим решение: $C = 2$, $B = -8$, $A = 7$. Следовательно, искомая парабола имеет вид

$$y = 7 - 8x + 2x^2. \quad (6.14)$$

6.1.5. Метод LU-разложения

Пусть необходимо решить систему линейных алгебраических уравнений (6.1), записанную в матричном виде (6.1). Допустим, матрица коэффициентов \mathbf{A} размером $n \times n$ невырожденная¹. Тогда ее можно представить в виде произведения двух матриц

$$\mathbf{A} = \mathbf{LU}, \quad (6.15)$$

где \mathbf{L} – нижняя треугольная матрица размером $n \times n$, у которой все элементы главной диагонали – единицы; \mathbf{U} – верхняя треугольная матрица размером $n \times n$.

При этом система (6.2) будет представлена в виде

$$\mathbf{LUx} = \mathbf{B}. \quad (6.16)$$

¹Матрица размером $n \times n$ называется невырожденной, если ее определитель отличен от нуля [3]

Введем вектор-столбец \mathbf{Y} , определяемый выражением

$$\mathbf{U}\mathbf{x} = \mathbf{Y}. \quad (6.17)$$

Подставляя (6.17) в (6.16), получим систему

$$\mathbf{L}\mathbf{Y} = \mathbf{B}. \quad (6.18)$$

При этом решение системы (6.16) можно получить последовательным решением линейных систем (6.18) и (6.17), причем каждая из них является треугольной и легко решается методами прямой и обратной подстановки [3].

Сама процедура разложения матрицы коэффициентов СЛАУ на две треугольные матрицы выполняется с использованием метода исключения Гаусса, т.е. посредством операций перестановки, масштабирования и замещения, что и определяет вычислительную сложность метода $O(n^3)$. Покажем это на простом примере [3]. Пусть невырожденная матрица коэффициентов СЛАУ имеет вид

$$\mathbf{A} = \begin{bmatrix} 4 & 3 & -1 \\ -2 & -4 & 5 \\ 1 & 2 & 6 \end{bmatrix}, \quad (6.19)$$

$$\mathbf{A} = \mathbf{E}\mathbf{A}. \quad (6.20)$$

где \mathbf{E} – единичная матрица, все элементы главной диагонали которой единицы, а остальные – нули. Учитывая (6.20), первый шаг разложения на треугольные матрицы можно записать в виде

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 4 & 3 & -1 \\ -2 & -4 & 5 \\ 1 & 2 & 6 \end{bmatrix}. \quad (6.21)$$

Применяя метод исключения Гаусса к правой матрице и занося константы замещения в левую, получим следующую последовательность операций:

- замещение второй строки правой матрицы разностью второй строки и первой строки, умноженной на константу -0.5 , которая будет элементом l_{21} левой матрицы

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 4 & 3 & -1 \\ 0 & -2.5 & 4.5 \\ 1 & 2 & 6 \end{bmatrix}; \quad (6.22)$$

- замещение третьей строки правой матрицы разностью третьей строки и первой строки, умноженной на константу 0.25, которая будет элементом l_{31} левой матрицы

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0.25 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 4 & 3 & -1 \\ 0 & -2.5 & 4.5 \\ 0 & 1.25 & 6.25 \end{bmatrix}; \quad (6.23)$$

- замещение третьей строки правой матрицы разностью третьей строки и второй строки, умноженной на константу -0.5 , которая будет элементом l_{32} левой матрицы

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0.25 & -0.5 & 1 \end{bmatrix} \times \begin{bmatrix} 4 & 3 & -1 \\ 0 & -2.5 & 4.5 \\ 0 & 0 & 8.5 \end{bmatrix}. \quad (6.24)$$

В результате получили треугольные матрицы разложения в виде

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0.25 & -0.5 & 1 \end{bmatrix}; \quad (6.25)$$

$$\mathbf{U} = \begin{bmatrix} 4 & 3 & -1 \\ 0 & -2.5 & 4.5 \\ 0 & 0 & 8.5 \end{bmatrix}. \quad (6.26)$$

Как видно из приведенного примера, при решении СЛАУ методом LU-разложения основные вычислительные затраты приходятся на разложение матрицы коэффициентов на треугольные матрицы.

Следует отметить, что вычислительная сложность методов исключения Гаусса и метода LU-разложения одинакова [3]. Однако, если необходимо решить несколько систем с одинаковыми матрицами коэффициентов, но различными векторами свободных членов, то метод LU-разложения окажется предпочтительным, так как в этом случае нет необходимости производить разложение матрицы коэффициентов многократно. Достаточно лишь сохранить полученные треугольные матрицы в памяти компьютера и, подставляя

различные векторы свободных членов, получать решения методами прямой и обратной подстановки. Это позволит значительно сократить объем вычислений по сравнению с методом исключения Гаусса.

6.1.6. Итерационные методы решения систем линейных алгебраических уравнений

Итерационные методы используются, как правило, для решения систем линейных алгебраических уравнений больших размерностей. В частности, при решении многих задач математической физики дискретизация дифференциальных уравнений в частных производных приводит к СЛАУ, содержащим десятки и сотни тысяч уравнений и более [3].

Поскольку вычислительная сложность методов исключения Гаусса и LU-разложения составляет (n^3) [3], решение линейных систем таких размерностей данными методами представляет серьезную проблему, решить которую позволяют итерационные *методы Якоби, Гаусса – Зейделя* и др.

Традиционно итерационные методы принято разделять на стационарные и нестационарные. Стационарные методы более просты для понимания и реализации, но обычно менее эффективны. Нестационарные методы более современные и эффективные, но сложнее для понимания и реализации. Ниже приведена классификация с кратким описанием некоторых из них [12].

- Стационарные методы:

- Итерация Якоби.

Метод основан на расчете каждого неизвестного члена вектора \mathbf{x} , выраженного через остальные неизвестные члены, полученные на предыдущем шаге. На каждой итерации расчет происходит единожды для каждого члена вектора. Метод прост в реализации, но сходится медленно. Условием сходимости является диагональное преобладание.

- Итерация Гаусса-Зейделя.

Данный метод подобен методу Якоби, но лишь с той разницей, что данные для расчета текущего неизвестного

члена дополняются решением, полученным на текущем шаге. Метод сходится немного быстрее метода Якоби.

- Метод верхней релаксации (SOR метод).

Метод может быть получен из метода Гаусса-Зейделя за счет введения параметра экстраполяции ω . Скорость сходимости метода может быть увеличена на порядок по сравнению с методом Гаусса-Зейделя, за счет оптимального подбора параметра ω .

- Нестационарные методы:

- Метод сопряженных градиентов (CG метод).

Данный метод основан на получении ряда сопряженных (ортогональных) векторов, обеспечивающих нахождение локального минимума функционала, на основе информации о его значениях и градиенте и является методом Крыловского типа. Сопряженные векторы вычисляются на каждой итерации и являются градиентом квадратичного функционала, минимизация которого приводит к решению СЛАУ. Метод сопряженных градиентов очень эффективен для симметричных матриц.

- Метод минимальных невязок (MINRES, SYMMLQ).
 - Обобщенный метод минимизации невязок (GMRES).
 - Метод бисопряженных градиентов (BiCG).
 - Метод квазимиимальных невязок (QMR).
 - Квадратный метод сопряженных градиентов (CGS).
 - Метод итераций Чебышева.

Стационарные итерационные методы можно представить в виде

$$\mathbf{x}^{(k)} = \mathbf{B}\mathbf{x}^{(k-1)} + \mathbf{c}, \quad (6.27)$$

где \mathbf{B} и \mathbf{c} не зависят от шага итерации.

Итерации Якоби

Пусть требуется решить систему линейных уравнений (6.1). Перепишем ее в следующем виде:

$$\left\{ \begin{array}{l} x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3 - \dots - a_{1n}x_n}{a_{11}}; \\ x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3 - \dots - a_{2n}x_n}{a_{22}}; \\ \vdots \\ x_i = \frac{b_i - a_{i1}x_1 - \dots - a_{i,i-1}x_{i-1} - a_{i,i+1}x_{i+1} - \dots - a_{in}x_n}{a_{ii}}; \\ \vdots \\ x_n = \frac{b_n - a_{n1}x_1 - a_{n2}x_2 - \dots - a_{n,n-1}x_{n-1}}{a_{nn}}. \end{array} \right. \quad (6.28)$$

Для решения системы (6.28) может быть использован следующий итерационный метод, получивший название *итерации Якоби*:

- 1) задание допустимой невязки решения δ . Допустимая невязка задается, как правило, в относительных единицах или процентах. Но в некоторых случаях удобнее задавать ее в абсолютных единицах;
- 2) задание начального приближения $x^{(0)} = [x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}]^T$. Начальное приближение к решению может быть задано произвольным образом либо из определенных соображений. Например, если известна окрестность решения, т.е. область в n -мерном пространстве, в которой гарантированно находится решение, в качестве начального приближения может быть выбрана центральная точка этой области;
- 3) нахождение следующего приближения к решению $x^{(k)} = [x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}]$ путем подстановки текущего приближения $x^{(k-1)} = [x_1^{(k-1)}, x_2^{(k-1)}, \dots, x_n^{(k-1)}]$ в правую часть системы (6.28)

$$x_i^{(k)} = (b_i - a_{i1}x_1^{(k-1)} - \dots - a_{i,i-1}x_{i-1}^{(k-1)} - a_{i,i+1}x_{i+1}^{(k-1)} - \dots - a_{in}x_n^{(k-1)})/a_{ii}, \quad (6.29)$$

²Верхний индекс в скобках указывает на номер итерации

где $i = 1, 2, \dots, n$;

- 4) определение невязки k -го приближения ε . Например, невязку в относительных единицах часто находят в соответствии с выражением

$$\varepsilon = \frac{\max \left(|x_1^{(k)} - x_1^{(k-1)}|, |x_2^{(k)} - x_2^{(k-1)}|, \dots, |x_n^{(k)} - x_n^{(k-1)}| \right)}{\max \left(|x_1^{(k-1)}|, |x_2^{(k-1)}|, \dots, |x_n^{(k-1)}| \right)}, \quad (6.30)$$

- 5) если выполняется неравенство

$$\varepsilon \leq \delta, \quad (6.31)$$

то найденное приближение к решению удовлетворяет заданной точности и итерационный процесс завершается выводом полученного результата. В противном случае найденное приближение к решению считается текущим и осуществляется переход к п. 3 и выполняется новая итерация.

Ниже приведен листинг программы в псевдокоде:

```

Выбор начального приближения  $x^0$ 
for  $k = 1, 2, \dots$ 
    for  $i = 1, 2, \dots, n$ 
         $s = 0$ 
        for  $j = 1, 2, \dots, i - 1, i + 1, \dots, n$ 
             $s = s + a_{ij}x_j^{(k-1)}$ 
        end
         $\bar{x}_i = (b_i - s)/a_{ii}$ 
    end
     $\mathbf{x}^k = \bar{\mathbf{x}}$ 
    Расчет невязки и контроль сходимости
end

```

Листинг программы, реализующий данный алгоритм на языке C++, приведен в приложениях **Д** и **Е**.

При правильной реализации вычислительного процесса каждое последующее приближение к решению должно быть точнее предыдущего. Естественно, в этом случае погрешность ε с каждой итерацией уменьшается. Такой итерационный процесс называют *сходящимся*.

Следует отметить, что в некоторых случаях итерационный процесс с каждой итерацией может не приближать, а наоборот, удалять от истинного решения. При этом погрешность ε с каждой итерацией увеличивается, а итерационный процесс называют расходящимся. Такая ситуация возникает при невыполнении определенных условий, которые называют условиями сходимости. Они зависят от особенностей организации итерационного процесса и характера системы алгебраических уравнений и будут рассмотрены ниже.

Следует иметь в виду, что в случае, когда знаменатель выражения (6.30) будет стремиться к нулю, невязка может возрасти при фактическом приближении к решению. Иными словами, в данном случае будет сделан неверный вывод о расходимости фактически сходящегося итерационного процесса.

Для решения данной проблемы используют отличные от (6.30) выражения для определения невязки, например,

$$\varepsilon = \frac{\max \left(|x_1^{(k)} - x_1^{(k-1)}|, |x_2^{(k)} - x_2^{(k-1)}|, \dots, |x_n^{(k)} - x_n^{(k-1)}| \right)}{\max \left(|x_1^{(0)}|, |x_2^{(0)}|, \dots, |x_n^{(0)}| \right)}, \quad (6.32)$$

если начальное приближение не приводит знаменатель к нулю.

Метод Гаусса-Зейделя

Иногда итерационный процесс можно ускорить.

Согласно итерационной формуле Якоби (6.29), для получения элементов приближения $x_i^{(k)}$ используются все, за исключением i -го, элементы $x_j^{(k-1)}$ предыдущего приближения. Но, поскольку на момент определения $x_i^{(k)}$ все элементы с индексами $j < i$ уже определены, их можно использовать для определения $x_i^{(k)}$. При этом возникает итерационный процесс, названный *итерацией Гаусса-Зейделя* [3]:

- 1) задание допустимой погрешности решения δ ;
- 2) задание начального приближения $x^{(0)} = [x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}]$;
- 3) нахождение следующего приближения к решению $x^{(k)} = [x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}]$ в соответствии с итерационной формулой

$$x_i^{(k)} = (b_i - a_{i1}x_1^{(k)} - \dots - a_{i,i-1}x_{i-1}^{(k)} - a_{i,i+1}x_{i+1}^{(k-1)} - \dots - a_{in}x_n^{(k-1)})/a_{ii}, \quad (6.33)$$

где $i = 1, 2, \dots, n$;

- 4) определение невязки k -го приближения ε ;
- 5) если выполняется неравенство

$$\varepsilon \leq \delta, \quad (6.34)$$

то найденное приближение к решению удовлетворяет заданной точности и итерационный процесс завершается выводом полученного результата. В противном случае найденное приближение к решению считается текущим и осуществляется переход к п. 3 и выполняется новая итерация.

Ниже приведен листинг программы в псевдокоде:

```

Выбор начального приближения  $x^0$ 
  for  $k = 1, 2, \dots$ 
    for  $i = 1, 2, \dots, n$ 
       $s = 0$ 
      for  $j = 1, 2, \dots, i - 1$ 
         $s = s + a_{ij}x_j^{(k-1)}$ 
      end
      for  $j = i + 1, \dots, n$ 
         $s = s + a_{ij}x_j^{(k-1)}$ 
      end
       $\bar{x}_i = (b_i - s)/a_{ii}$ 
    end
     $\mathbf{x}^k = \bar{\mathbf{x}}$ 
    Расчет невязки и контроль сходимости
  end

```

Как видно из приведенных выше описаний, метод Гаусса – Зейделя отличается от метода Якоби лишь незначительным изменением, внесенным в итерационную формулу (6.33), но это во многих случаях позволяет ускорить итерационный процесс.

Подтвердим сказанное простым примером.

Решим линейную систему

$$\begin{aligned}
 2x_1 - x_2 + x_3 &= 2; \\
 4x_1 - 6x_2 + x_3 &= -4; \\
 -2x_1 + x_2 + 8x_3 &= 16
 \end{aligned}
 \tag{6.35}$$

методом Якоби с использованием начального приближения $x^{(0)} = [1, 2, 2]$.

Для этого перепишем систему (6.35) в виде

$$\begin{aligned}
 x_1 &= \frac{2 + x_2 - x_3}{2}; \\
 x_2 &= \frac{-4 - 4x_1 - x_3}{-6}; \\
 x_3 &= \frac{16 + 2x_1 - x_2}{8},
 \end{aligned}
 \tag{6.36}$$

после чего решим систему (6.36) в среде MATLAB.

Функция для решения СЛАУ произвольной размерности методом Якоби в среде MATLAB может иметь следующий вид:

```
function X=yakobi(A,B,X0,delta,Imax)

% Итерация Якоби.
% A      - невырожденная матрица коэффициентов
%          размером n x n;
% B      - вектор-столбец свободных членов;
% X0     - вектор-столбец начального приближения;
% delta  - допустимое значение невязки;
% Imax   - максимальное число итераций;
% X      - приближенное решение линейной системы  $AX = B$ .

n=length(B);
err=5*delta;
X=X0;
ct=0;
while err>delta
    Xp=X;
    for i=1:n
        X(i)=(B(i)-A(i,[1:i-1,i+1:n])*...
            Xp([1:i-1,i+1:n]))/A(i,i);
    end
    err=max(abs(X-Xp))/max(abs(Xp));
    ct=ct+1;
    Xe(ct,:)= [X' err];
    if ct>Imax
        error('Требуемое число итераций слишком велико')
        break
    end
end
Xe
ct
```

Получить информацию о функциях и синтаксисе написания программ в системе инженерных и научных расчетов MATLAB можно в [17, 18].

Для запуска описанного выше итерационного процесса с допустимой невязкой 10^{-5} и максимальным числом итераций не более 100 необходимо войти в систему MATLAB, создать файл функции и сохранить его под именем `yakobi.m`, после чего в командной строке последовательно набрать команды, приведенные ниже.

```
A=[ 2 -1 1; 4 -6 1; -2 1 8];
B=[2; -4; 16];
X0=[1; 2; 2];
X=yakobi(A,B,X0,1e-5,1e2)
```

Первая из них задает матрицу коэффициентов **A** размером 3×3 , вторая – вектор-столбец свободных членов **B** размером 3×1 , третья – вектор-столбец начального приближения x_0 размером 3×1 и четвертая осуществляет запуск итерационного процесса Якоби с заданными параметрами.

Полученные результаты будут представлены на экране монитора в следующем виде:

$x_e =$

1.0000	1.6667	2.0000	0.1667
0.8333	1.6667	2.0417	0.0833
0.8125	1.5625	2.0000	0.0521
0.7813	1.5417	2.0078	0.0156
0.7669	1.5221	2.0026	0.0098
0.7598	1.5117	2.0015	0.0052
0.7551	1.5068	2.0010	0.0025
0.7529	1.5036	2.0004	0.0016
0.7516	1.5020	2.0003	0.0008
0.7509	1.5011	2.0001	0.0005
0.7505	1.5006	2.0001	0.0002
0.7503	1.5003	2.0000	0.0001
0.7501	1.5002	2.0000	0.0001
0.7501	1.5001	2.0000	0.0000
0.7500	1.5001	2.0000	0.0000
0.7500	1.5000	2.0000	0.0000
0.7500	1.5000	2.0000	0.0000

ct =

17

X =

0.7500

1.5000

2.0000,

где первые три столбца матрицы $\mathbf{X_e}$ представляют собой приближения переменных x_1, x_2, x_3 к решению, четвертый столбец – погрешности приближений, ct – число итераций, \mathbf{X} – вектор результата.

Таким образом, решение системы (6.35) с невязкой 10^{-5} и с использованием начального приближения $x^{(0)} = [1, 2, 2]$ методом Якоби было получено за 17 итераций.

Функция для решения СЛАУ произвольной размерности методом Гаусса – Зейделя в среде MATLAB может иметь следующий вид:

```
function X=zeydel(A,B,X0,delta,Imax)
```

```
% Итерация Гаусса - Зейделя.
```

```
% A      - невырожденная матрица коэффициентов
```

```
%          размером n x n;
```

```
% B      - вектор-столбец свободных членов;
```

```
% X0     - вектор-столбец начального приближения;
```

```
% delta  - допустимое значение невязки;
```

```
% Imax   - максимальное число итераций;
```

```
% X      - приближенное решение линейной системы  $\mathbf{AX} = \mathbf{B}$ .
```

```
n=length(B);
```

```
err=5*delta;
```

```
X=X0;
```

```
ct=0;
```

```
while err>delta
```

```
    Xp=X;
```

```
    for i=1:n
```

```
        X(i)=(B(i)-A(i,[1:i-1,i+1:n]))*...
```

```

        X([1:i-1,i+1:n]))/A(i,i);
    end
    err=max(abs(X-Xp))/max(abs(Xp));
    ct=ct+1;
    Xe(ct,:)= [X' err];
    if ct>Imax
        error('Требуемое число итераций слишком велико')
        break
    end
end
Xe
ct

```

Запуск итерационного процесса Гаусса – Зейделя с допустимой невязкой результата 10^{-5} и максимальным числом итераций не более 100 осуществляется аналогично итерации Якоби. Создаваемый файл функции сохраняется под именем **zeydel.m**, после чего в командной строке последовательно набираются команды, приведенные ниже.

```

A=[ 2 -1 1; 4 -6 1; -2 1 8];
B=[2; -4; 16];
X0=[1; 2; 2];
X=zeydel(A,B,X0,1e-5,1e2)

```

Полученные результаты будут представлены на экране монитора в следующем виде:

Xe =

1.0000	1.6667	2.0417	0.1667
0.8125	1.5486	2.0095	0.0937
0.7695	1.5146	2.0031	0.0215
0.7558	1.5044	2.0009	0.0069
0.7517	1.5013	2.0003	0.0020
0.7505	1.5004	2.0001	0.0006
0.7502	1.5001	2.0000	0.0002
0.7500	1.5000	2.0000	0.0001
0.7500	1.5000	2.0000	0.0000
0.7500	1.5000	2.0000	0.0000

ct =

10

X =

0.7500

1.5000

2.0000.

Таким образом, решение системы (6.35) с невязкой 10^{-5} и с использованием начального приближения $x^{(0)} = [1, 2, 2]$ методом Гаусса – Зейделя было получено за 10 итераций. Таким образом, в данном случае число итераций в процессе Гаусса – Зейделя оказалось в 1,7 раза меньше, чем при решении методом Якоби.

Критерий сходимости итераций Якоби и Гаусса-Зейделя

Перед тем, как сформулировать критерий сходимости итераций Якоби и Гаусса – Зейделя для систем линейных алгебраических уравнений, дадим одно определение.

Матрица $\mathbf{A} = [a_{ij}]$ размером $n \times n$ называется *строго диагонально доминирующей*, если выполняется условие

$$|a_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}|, \quad (6.37)$$

т.е. если в каждой строке матрицы модуль элемента главной диагонали больше суммы модулей всех остальных элементов строки [3].

Критерий сходимости итераций Якоби и Гаусса – Зейделя формулируется следующим образом.

Если \mathbf{A} – строго диагонально доминирующая матрица, то линейная система $\mathbf{Ax} = \mathbf{B}$ имеет единственное решение $\mathbf{x} = \mathbf{P}$ и итерационные процессы Якоби и Гаусса – Зейделя порождают последовательность векторов $\{x^{(k)}\}$, которые сходятся к \mathbf{P} для любого выбора вектора начального приближения $x^{(0)}$ [3].

Данный критерий является достаточным условием сходимости, но не является необходимым. Если матрица коэффициентов не является строго диагонально доминирующей, итерационные процессы Якоби и Гаусса – Зейделя могут быть расходящимися.

В качестве примера достаточно в системе (6.35) поменять местами первое и второе уравнения. При этом матрица коэффициентов будет иметь вид

$$\mathbf{A} = \begin{bmatrix} 4 & -6 & 1 \\ 2 & -1 & 1 \\ -2 & 1 & 8 \end{bmatrix}, \quad (6.38)$$

т.е. не будет строго диагонально доминирующей, поскольку для первых двух строк условие (6.37) не выполняется.

Результаты, полученные для первых 15 итераций Якоби, в этом случае будут иметь вид ³

$\mathbf{X}_e =$

1.0e+003 *

0.0015	0.0020	0.0020	0.0003
0.0015	0.0030	0.0021	0.0005
0.0030	0.0031	0.0020	0.0007
0.0032	0.0059	0.0024	0.0014
0.0073	0.0067	0.0021	0.0021
0.0086	0.0147	0.0030	0.0040
0.0203	0.0181	0.0023	0.0059
0.0256	0.0409	0.0048	0.0114
0.0591	0.0541	0.0033	0.0167
0.0793	0.1196	0.0100	0.0327
0.1758	0.1666	0.0069	0.0483
0.2472	0.3565	0.0251	0.0950
0.5275	0.5175	0.0192	0.1402
0.7704	1.0723	0.0692	0.2774
1.5901	1.6080	0.0606	0.4098
2.3959	3.2388	0.1985	0.8154

³ Данная форма представления матрицы \mathbf{X}_e означает, что 1.0e+003 (10^3) является множителем для каждого элемента матрицы.

??? Error using ==> yakobi

Требуемое число итераций слишком велико

Аналогичные результаты для итераций Гаусса – Зейделя будут выглядеть следующим образом:

$x_e =$

1.0e+007 *

0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000
0.0001	0.0001	0.0000	0.0000
0.0002	0.0003	0.0000	0.0001
0.0005	0.0010	0.0000	0.0003
0.0015	0.0030	0.0000	0.0010
0.0044	0.0089	0.0000	0.0030
0.0133	0.0266	0.0000	0.0089
0.0399	0.0797	0.0000	0.0266
0.1196	0.2391	0.0000	0.0797
0.3587	0.7174	0.0000	0.2391
1.0762	2.1523	0.0000	0.7174

??? Error using ==> zeydel

Требуемое число итераций слишком велико

Обе последовательности расходятся. Из приведенных примеров видно, что более быстрая сходимость метода Гаусса – Зейделя при условии строго диагонально доминирующей матрицы коэффициентов СЛАУ оборачивается более быстрой расходимостью при невыполнении условия (6.37).

Метод верхней релаксации

Метод верхней релаксации экстраполирован из метода Гаусса-Зейделя. Текущий шаг вычисляется прибавлением разницы между

текущим шагом, посчитанным методом Гаусса-Зейделя, и предыдущим шагом, умноженной на весовой коэффициент. Выражение для расчета текущего приближения будет иметь вид [12]:

$$x_i^{(k)} = w\bar{x}_i^{(k)} + (1 - w)x_i^{(k-1)}. \quad (6.39)$$

Ниже приведен листинг программы в псевдокоде:

```

Выбор начального приближения  $x^0$ 
for  $k = 1, 2, \dots$ 
  for  $i = 1, 2, \dots, n$ 
     $s = 0$ 
    for  $j = 1, 2, \dots, i - 1$ 
       $s = s + a_{ij}x_j^{(k-1)}$ 
    end
    for  $j = i + 1, \dots, n$ 
       $s = s + a_{ij}x_j^{(k-1)}$ 
    end
     $\bar{x}_i = (b_i - s)/a_{ii}$ 
     $x_i^{(k)} = x_i^{(k-1)} + w(\bar{x}_i - x_i^{(k-1)})$ 
  end
  Расчет невязки и контроль сходимости
end

```

Выбор весового коэффициента

При выборе весового коэффициента $w = 1$ метод верхней релаксации становится методом Гаусса-Зейделя. В соответствии с теоремой Кахана [12] данный метод расходится, если w лежит вне диапазона $(0, 2)$.

В основном невозможно рассчитать оптимальное значение w , а если и возможно, то вычислительные затраты становятся слишком велики. Как правило, используют эвристическую оценку $w = 2 - O(h)$, где h – сеточный шаг дискретизации.

Если матрица \mathbf{A} симметричная и положительно определенная, то метод верхней релаксации сходится во всем диапазоне значений $w \in (0, 2)$, хотя выбор значения w может значительно повысить скорость схождения к решению.

Метод сопряженных градиентов

Нестационарные итерационные методы отличаются от стационарных тем, что на каждом итерационном шаге вычисляется новое направление и коэффициенты.

Метод сопряженных градиентов очень эффективен для симметричных положительно определенных систем. Это один из старейших и хорошо изученных методов [12].

На каждом шаге расчета вектор $x^{(k)}$ модифицируется величиной шага α_k и рассчитывается направляющий вектор $p^{(k)}$:

$$x^{(k)} = x^{(k-1)} + \alpha_k p^{(k)}. \quad (6.40)$$

Соответственно невязку $r^{(k)} = b - Ax^{(k)}$ можно представить в виде

$$r^{(k)} = r^{(k-1)} - \alpha_k q^{(k)}, \quad (6.41)$$

где $q^{(k)} = Ap^{(k)}$.

```

Расчет для начального приближения  $x^0$ 
 $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}x^{(0)}$ 
 $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$ 
for  $k = 1, 2, \dots$ 
     $\mathbf{w}^k = \mathbf{A} \cdot \mathbf{p}^{(k-1)}$ 
     $\alpha_k = \frac{(\mathbf{r}^{(k-1)}, \mathbf{p}^{(k-1)})}{(\mathbf{p}^{(k-1)}, \mathbf{w}^{(k)})}$ 
     $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{p}^{(k-1)}$ 
     $\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} - \alpha_k \mathbf{w}^{(k)}$ 
     $\beta_k = \frac{(\mathbf{r}^{(k)}, \mathbf{w}^{(k)})}{(\mathbf{p}^{(k-1)}, \mathbf{w}^{(k)})}$ 
     $\mathbf{p}^{(k)} = \mathbf{r}^{(k)} + \beta_k \mathbf{p}^{(k-1)}$ 
    Расчет невязки и контроль сходимости
end

```

Листинг программы, реализующий данный алгоритм на языке C++, приведен в приложении E.

Принцип метода сопряженных градиентов заключается в следующем.

Градиентом дифференцируемой функции $f(x)$ в точке x_0 называется n -мерный вектор $f'(x_0)$, компоненты которого являются частными производными функции $f(x)$, вычисленными в точке x_0 , т. е. [13]

$$f'(x_0) = \frac{\partial f(x_0)}{\partial x_1}, \dots, \frac{\partial f(x_0)}{\partial x_n}. \quad (6.42)$$

Этот вектор перпендикулярен к плоскости, проведенной через точку x_0 , и касательной к поверхности уровня функции $f(x)$, проходящей через точку x_0 . В каждой точке такой поверхности функция $f(x)$ принимает одинаковое значение. Приравнивая функцию различным постоянным величинам C_0, C_1, \dots , получим серию поверхностей, характеризующих ее топологию (рис. 6.1).

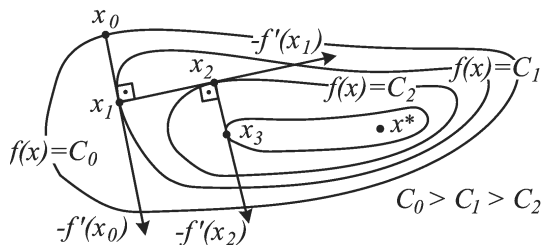


Рис. 6.1. Серия поверхностей, характеризующих функцию $f(x)$ и направление наискорейшего убывания функции, определяемое антиградиентом

Вектор-градиент направлен в сторону наискорейшего возрастания функции в данной точке. Вектор, противоположный градиенту $(-f'(x_0))$, называется антиградиентом (рис. 6.2) и направлен в сторону наискорейшего убывания функции. В точке минимума градиент функции равен нулю. На свойствах градиента основаны методы первого порядка, называемые также градиентными, и методы минимизации. Использование этих методов в общем случае позволяет определить точку локального минимума функции.

Очевидно, что если нет дополнительной информации, то из начальной точки x_0 разумно перейти в точку x_1 , лежащую в на-

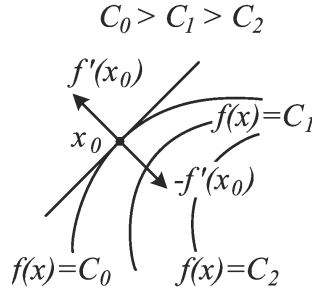


Рис. 6.2. Направление градиента ($f'(x_0)$) и антиградиента ($-f'(x_0)$) функции в точке x_0

правлении антиградиента – наискорейшего убывания функции (см. рис. 6.1). Выбирая в качестве направления спуска $p^{(k)}$ антиградиент $-f'(x^{(k)})$, получаем итерационный процесс вида 6.40.

В координатной форме этот процесс записывается следующим образом:

$$x_i^{(k)} = x_i^{(k-1)} + \alpha_k \frac{\partial f(x^{(k)})}{\partial x_i}; \quad i = 1, \dots, n; k = 0, 1, 2, \dots \quad (6.43)$$

В качестве критерия остановки итерационного процесса используют либо выполнение условия малости приращения аргумента

$$\|x^{(k)} - x^{(k-1)}\| \leq \epsilon, \quad (6.44)$$

либо выполнение условия малости градиента

$$\|f'(x^{(k)})\| \leq \gamma, \quad (6.45)$$

где ϵ, γ – заданные малые величины.

Возможен и комбинированный критерий, состоящий в одновременном выполнении указанных условий. Градиентные методы отличаются друг от друга способами выбора величины шага α_k .

При методе с постоянным шагом для всех итераций выбирается некоторая постоянная величина шага. Достаточно малый шаг α_k обеспечит убывание функции, т. е. выполнение неравенства

$$f(x^{(k)}) = f\left(x^{(k-1)} - \alpha_k f'(x^{(k-1)})\right) < f(x^{(k-1)}). \quad (6.46)$$

Однако это может привести к необходимости проводить неприемлемо большое количество итераций для достижения точки минимума. С другой стороны, слишком большой шаг может вызвать неожиданный рост функции либо привести к колебаниям около точки минимума (зацикливанию). Из-за сложности получения необходимой информации для выбора величины шага методы с постоянным шагом применяются на практике редко [13].

Более экономичны в смысле количества итераций и надежности градиентные методы с переменным шагом, когда в зависимости от результатов вычислений величина шага некоторым образом меняется.

6.2. Методы решения систем нелинейных алгебраических уравнений

В отличие от СЛАУ, прямых методов решения систем нелинейных алгебраических уравнений не существует. Нелинейные системы решаются итерационными методами, которые можно разделить на две группы:

- **методы последовательного решения**, обеспечивающие последовательное нахождение элементов решения систем нелинейных алгебраических уравнений, что позволяет значительно экономить ресурсы оперативной памяти и благодаря этому решать системы уравнений большой размерности;
- **методы одновременного решения**, позволяющие находить все элементы решения нелинейной системы алгебраических уравнений одновременно в результате итерационного выполнения определенных матричных операций, что обеспечивает повышение скорости сходимости процесса решения (уменьшение числа итераций по сравнению с методами последовательного решения), но при этом требует значительных вычислительных ресурсов компьютера.

К методам последовательного решения нелинейных систем относят *итерацию неподвижной точки* и различные модификации данного метода [3].

Из методов одновременного решения нелинейных систем наиболее широкое распространение получил **метод Ньютона-Рафсона** и его модификации [3].

6.2.1. Итерация неподвижной точки

Пусть задана система нелинейных алгебраических уравнений в виде

$$\begin{aligned}x_1 &= f_1(x_1, x_2, \dots, x_n); \\x_2 &= f_2(x_1, x_2, \dots, x_n); \\x_3 &= f_3(x_1, x_2, \dots, x_n); \\&\vdots \\x_n &= f_n(x_1, x_2, \dots, x_n),\end{aligned}\tag{6.47}$$

где f_1, f_2, \dots, f_n – некоторые непрерывные и непрерывно дифференцируемые нелинейные функции от $(x_1, x_2, x_3, \dots, x_n)$.

Неподвижной точкой для системы (6.47) называется такая точка $\mathbf{P} = (p_1, p_2, \dots, p_n)$, для которой выполняется равенство [3]

$$\begin{aligned}p_1 &= f_1(p_1, p_2, \dots, p_n); \\p_2 &= f_2(p_1, p_2, \dots, p_n); \\p_3 &= f_3(p_1, p_2, \dots, p_n); \\&\vdots \\p_n &= f_n(p_1, p_2, \dots, p_n).\end{aligned}\tag{6.48}$$

Иными словами, неподвижная точка фактически является решением системы (6.47).

Итерационный метод решения систем нелинейных алгебраических уравнений, называемый итерацией неподвижной точки, является обобщением итерации Якоби на случай нелинейных систем и состоит в следующем:

- 1) задание допустимого значения невязки δ ;
- 2) задание начального приближения $x^{(0)} = [x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}]$;

- 3) нахождение следующего приближения к решению $x^{(k)} = [x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}]$ путем подстановки текущего приближения $x^{(k-1)} = [x_1^{(k-1)}, x_2^{(k-1)}, \dots, x_n^{(k-1)}]$ в правую часть системы (6.47)

$$x_i^{(k)} = f_i(x_1^{(k-1)}, x_2^{(k-1)}, x_3^{(k-1)}, \dots, x_n^{(k-1)}), \quad (6.49)$$

где $i = 1, 2, \dots, n$;

- 4) определение невязки k -го приближения ε в соответствии с выражением

$$\varepsilon = \frac{\max(|x_1^{(k)} - x_1^{(k-1)}|, |x_2^{(k)} - x_2^{(k-1)}|, \dots, |x_n^{(k)} - x_n^{(k-1)}|)}{\max(|x_1^{(k-1)}|, |x_2^{(k-1)}|, \dots, |x_n^{(k-1)}|)}; \quad (6.50)$$

- 5) если выполняется неравенство

$$\varepsilon \leq \delta, \quad (6.51)$$

то найденное приближение к решению удовлетворяет заданной точности и итерационный процесс завершается выводом полученного результата. В противном случае найденное приближение к решению считается текущим и осуществляется переход к п. 3 и выполняется новая итерация.

Обобщение итерации Гаусса – Зейделя на случай нелинейных систем будет аналогично рассмотренному выше, за исключением итерационной формулы (6.49), которая в данном случае будет иметь вид

$$x_i^{(k)} = f_i(x_1^{(k)}, x_2^{(k)}, \dots, x_{i-1}^{(k)}, x_i^{(k-1)}, \dots, x_n^{(k-1)}), \quad (6.52)$$

где $i = 1, 2, \dots, n$.

6.2.2. Критерий сходимости итерации неподвижной точки

Критерий сходимости итерации неподвижной точки к решению может быть сформулирован следующим образом.

Пусть задана нелинейная система (6.47) и функции f_1, f_2, \dots, f_n и их первые частные производные непрерывны в некоторой области, содержащей неподвижную точку $\mathbf{P} = (p_1, p_2, \dots, p_n)$.

Если начальное приближение $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)})$ находится в некоторой достаточно малой окрестности неподвижной точки и во всех точках данной окрестности выполняются неравенства

$$\begin{aligned} \left| \frac{\partial f_1}{\partial x_1} \right| + \left| \frac{\partial f_1}{\partial x_2} \right| + \left| \frac{\partial f_1}{\partial x_3} \right| + \dots + \left| \frac{\partial f_1}{\partial x_n} \right| &< 1; \\ \left| \frac{\partial f_2}{\partial x_1} \right| + \left| \frac{\partial f_2}{\partial x_2} \right| + \left| \frac{\partial f_2}{\partial x_3} \right| + \dots + \left| \frac{\partial f_2}{\partial x_n} \right| &< 1; \\ \left| \frac{\partial f_3}{\partial x_1} \right| + \left| \frac{\partial f_3}{\partial x_2} \right| + \left| \frac{\partial f_3}{\partial x_3} \right| + \dots + \left| \frac{\partial f_3}{\partial x_n} \right| &< 1; \\ &\vdots \\ \left| \frac{\partial f_n}{\partial x_1} \right| + \left| \frac{\partial f_n}{\partial x_2} \right| + \left| \frac{\partial f_n}{\partial x_3} \right| + \dots + \left| \frac{\partial f_n}{\partial x_n} \right| &< 1, \end{aligned} \quad (6.53)$$

то итерация, заданная итерационными формулами (6.49) или (6.52), сходится к неподвижной точке.

Критерий (6.53) является достаточным условием сходимости итерации неподвижной точки, но не является необходимым. Если условие (6.53) не выполняется, итерационный процесс может (но не обязательно) расходиться. Обычно это происходит, если сумма модулей частных производных в окрестности неподвижной точки существенно больше 1 [3].

Критерий сходимости для итерации неподвижной точки можно наглядно проиллюстрировать графически на примере итерационного решения нелинейного уравнения одной переменной

$$x = g(x), \quad (6.54)$$

где $g(x)$ – некоторая нелинейная функция аргумента x .

Нетрудно видеть, что уравнение (6.54) может быть представлено в виде системы двух уравнений:

$$\begin{cases} y = g(x); \\ y = x. \end{cases} \quad (6.55)$$

Если систему (6.55) представить графически, то ее решением, а значит и решением уравнения (6.54), будет точка **P** пересечения графиков $y = g(x)$ и $y = x$ (рис. 6.3).

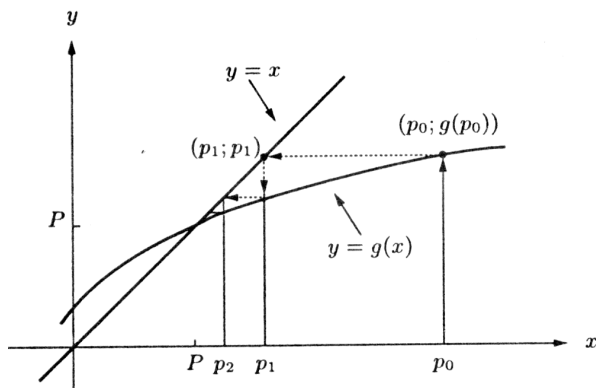


Рис. 6.3. Монотонно сходящаяся итерация неподвижной точки

Выберем некоторое начальное приближение $x = p_0$. Подставив это значение в уравнение $y = g(x)$, найдем соответствующее значение y , подставив которое в уравнение $y = x$, получим новое приближение к решению $x = p_1$ (см. рис.6.3).

Далее, проведя аналогичные операции, получим следующее приближение $x = p_2$, и так далее до тех пор, пока очередное приближение не будет удовлетворять заданной точности.

Из приведенных графиков видно, что в данном примере в рассматриваемой окрестности точки P выполняется неравенство

$$\left| \frac{dg(x)}{dx} \right| < 1, \quad (6.56)$$

что является достаточным условием сходимости. Каждое последующее приближение ближе к решению, чем предыдущее. Шаг между k -м и $(k - 1)$ -м приближениями с каждой итерацией уменьшается, что приводит к уменьшению невязки, определяемой выражением (6.50).

В приведенном примере модуль разности между текущим приближением и решением P с каждой итерацией уменьшается, а знак

этой разности остается неизменным. Такая сходимость называется монотонной [3].

На рис. 6.4 приведен пример нелинейного уравнения вида (6.54), для которого в окрестности решения P тоже выполняется условие сходимости (6.56). При этом модуль разности между текущим приближением и решением P с каждой итерацией также уменьшается, но знак этой разности с каждой последующей итерацией изменяется на противоположный. Такая сходимость называется колеблющейся [3].

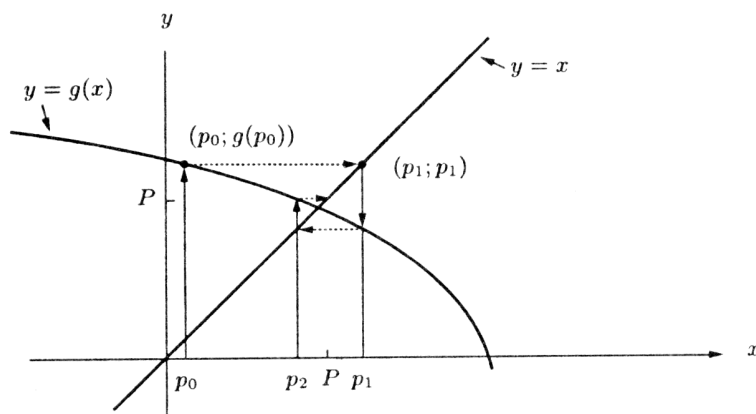


Рис. 6.4. Колеблющаяся сходимость итерации неподвижной точки

Если уравнение (6.54), представленное в виде системы (6.55), таково, что в окрестности решения P условие сходимости (6.56) не выполняется

$$\left| \frac{dg(x)}{dx} \right| > 1, \quad (6.57)$$

то, даже если начальное приближение $x = p_0$ достаточно близко к решению P , итерация неподвижной точки дает расходящуюся последовательность. При этом в зависимости от знака производной $dg(x)/dx$ в окрестности решения будет наблюдаться монотонная расходимость (в случае $dg(x)/dx > 1$, рис. 6.5) или расходящиеся колебания (в случае $dg(x)/dx < -1$, рис. 6.6) [3].

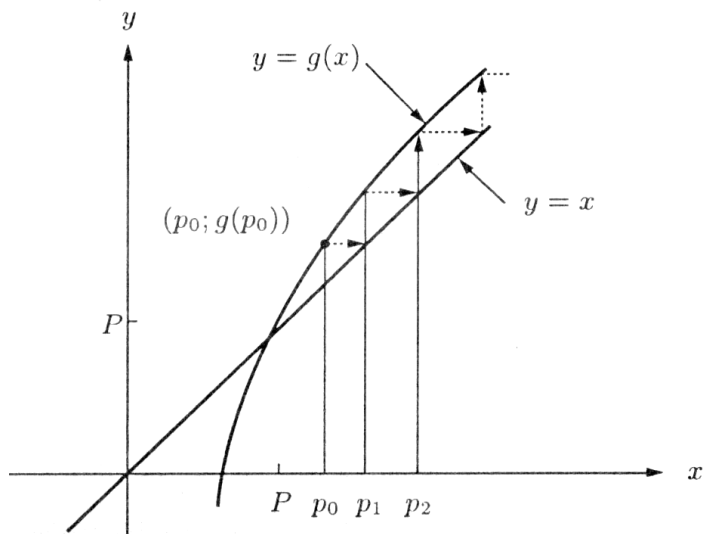


Рис. 6.5. Монотонная расходимость итерации неподвижной точки

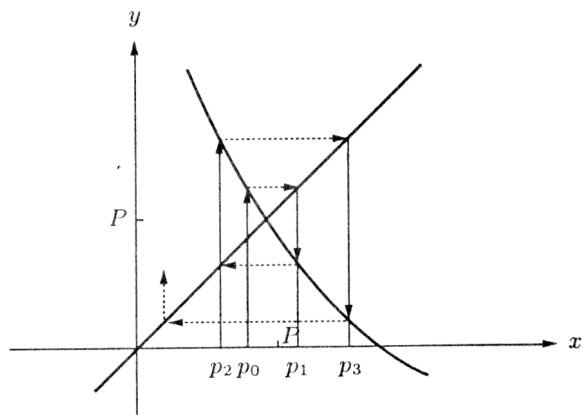


Рис. 6.6. Расходящиеся колебания в процессе итерации неподвижной точки

6.2.3. Метод Ньютона – Рафсона

Пусть требуется решить нелинейное уравнение (6.54). Перепишем это уравнение в виде

$$f(x) = 0. \quad (6.58)$$

Если функция $f(x)$, ее первая и вторая производные непрерывны в окрестности решения P , то для решения уравнения (6.58) может быть использован **метод Ньютона-Рафсона**, который характеризуется более быстрой сходимостью, чем итерация неподвижной точки [1, 2, 3].

Если представить уравнение (6.58) графически, то его решением будет точка P пересечения графика $y = f(x)$ с осью x (рис. 6.7).

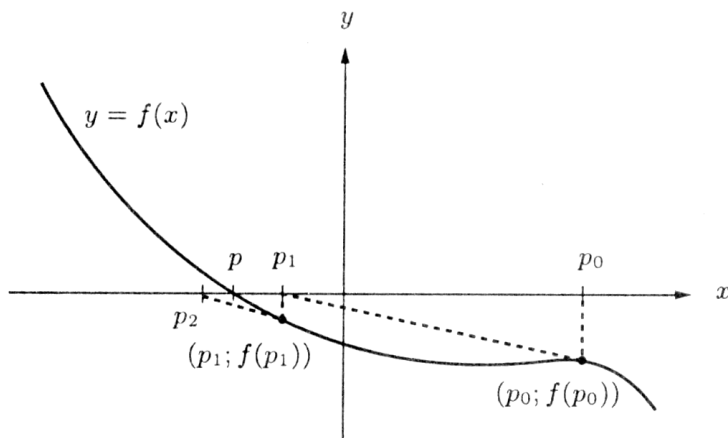


Рис. 6.7. Графическая интерпретация метода Ньютона – Рафсона

Пусть выбранное начальное приближение $x = p_0$ достаточно близко к решению. Определим следующее приближение $x = p_1$ как точку пересечения с осью x касательной к графику $y = f(x)$ в точке $x = p_0$. Из рис. 6.7 видно, что точка $x = p_1$ ближе к решению, чем $x = p_0$.

Соотношение, связывающее p_0 и p_1 , можно получить из выражения для тангенса угла наклона касательной к графику $y = f(x)$,

проходящей через точки $x = p_0$ и $x = p_1$:

$$\left. \frac{df(x)}{dx} \right|_{x=p_0} = \frac{f(p_1) - f(p_0)}{p_1 - p_0}. \quad (6.59)$$

Учитывая, что график касательной пересекает ось x в точке p_1 , $f(p_1) = 0$ (см. рис. 6.7), получим

$$p_1 = p_0 - \frac{f(p_0)}{\left. \frac{df(x)}{dx} \right|_{x=p_0}}. \quad (6.60)$$

Обобщая выражение (6.60) для k -й итерации, получим

$$p_1 = p_0 - \frac{f(p_0)}{\left. \frac{df(x)}{dx} \right|_{x=p_0}}. \quad (6.61)$$

Пусть требуется решить систему нелинейных алгебраических уравнений размерностью n , записанную в виде

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0; \\ f_2(x_1, x_2, \dots, x_n) &= 0; \\ f_3(x_1, x_2, \dots, x_n) &= 0; \\ &\vdots \\ f_n(x_1, x_2, \dots, x_n) &= 0. \end{aligned} \quad (6.62)$$

В матричном представлении система (6.62) может быть записана как

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}, \quad (6.63)$$

где $\mathbf{x} = [x_1, x_2, \dots, x_n]$ – вектор-столбец переменных; $\mathbf{0}$ в правой части – вектор-столбец размерностью n , все элементы которого равны 0.

Обобщенный метод Ньютона – Рафсона на случай нелинейных систем произвольной размерности n состоит в следующем:

- 1) задание допустимой невязки решения δ ;
- 2) задание начального приближения $\mathbf{x}^{(0)} = [x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}]$;

- 3) нахождение следующего приближения к решению $\mathbf{x}^{(k)} = [x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}]$ путем подстановки текущего приближения $\mathbf{x}^{(k-1)} = [x_1^{(k-1)}, x_2^{(k-1)}, \dots, x_n^{(k-1)}]$ в итерационную формулу, аналогичную (6.61):

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - \left[\frac{\partial \mathbf{F}(\mathbf{x})}{\partial \mathbf{x}} \Big|_{\mathbf{x}^{(k-1)}} \right]^{-1} \times \mathbf{F}(\mathbf{x}^{(k-1)}), \quad (6.64)$$

где $\left[\frac{\partial \mathbf{F}(\mathbf{x})}{\partial \mathbf{x}} \Big|_{\mathbf{x}^{(k-1)}} \right]^{-1}$ – матрица размером $n \times n$, обратная матрице

$$\left[\frac{\partial \mathbf{F}(\mathbf{x})}{\partial \mathbf{x}} \right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}, \quad (6.65)$$

называемой матрицей Якоби [3], в точке $\mathbf{x}^{(k-1)} = [x_1^{(k-1)}, x_2^{(k-1)}, \dots, x_n^{(k-1)}]$;

- 4) определение погрешности k -го приближения ε в соответствии с выражением

$$\varepsilon = \frac{\max \left(|x_1^{(k)} - x_1^{(k-1)}|, |x_2^{(k)} - x_2^{(k-1)}|, \dots, |x_n^{(k)} - x_n^{(k-1)}| \right)}{\max \left(|x_1^{(k-1)}|, |x_2^{(k-1)}|, \dots, |x_n^{(k-1)}| \right)}, \quad (6.66)$$

- 5) если выполняется неравенство

$$\varepsilon \leq \delta, \quad (6.67)$$

то найденное приближение к решению удовлетворяет заданной точности и итерационный процесс завершается выводом полученного результата. В противном случае найденное приближение к решению считается текущим, осуществляется переход к п. 3 и выполняется новая итерация.

В учебных пособиях по математическому анализу [1, 2] показано, что итерация неподвижной точки характеризуется линейной сходимостью, а метод Ньютона – Рафсона – квадратичной сходимостью. Это значит, что при условии сходимости итерации неподвижной точки невязка в каждой последующей итерации убывает пропорционально невязке в предыдущей итерации. А при условии сходимости метода Ньютона – Рафсона невязка в каждой последующей итерации убывает пропорционально квадрату невязки в предыдущей итерации.

6.2.4. Критерий сходимости метода Ньютона – Рафсона

Так же, как и итерация неподвижной точки, метод Ньютона – Рафсона может давать расходящиеся последовательности приближений к решению, один из примеров которой показан на рис. 6.8 [3].

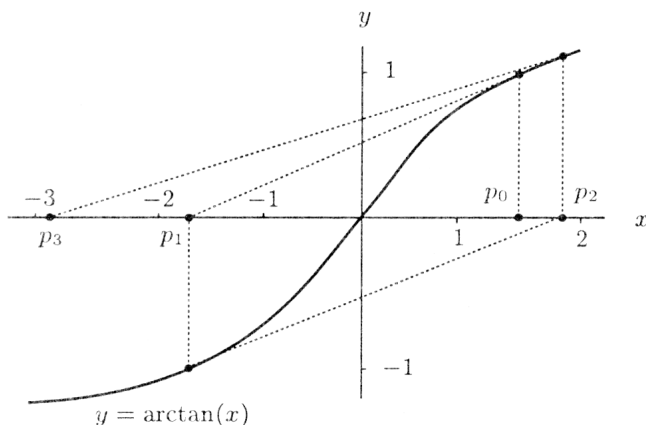


Рис. 6.8. Расходящаяся последовательность приближений к решению при использовании метода Ньютона – Рафсона

Кроме того, возможны так называемые циклические последовательности, когда получаемые приближения к решению циклически повторяются или почти повторяются (рис. 6.9) [3].

Критерий сходимости метода Ньютона – Рафсона может быть получен из следующих рассуждений [3]. Пусть задана система нелинейных алгебраических уравнений вида (6.63). Введем матричную функцию

$$\mathbf{G}(\mathbf{x}) = \mathbf{x} - \left[\frac{\partial \mathbf{F}(\mathbf{x})}{\partial \mathbf{x}} \right]^{-1} \times \mathbf{F}(\mathbf{x}), \quad (6.68)$$

называемую интерполяционной функцией Ньютона.

Пусть $P = (p_1, p_2, \dots, p_n)$ – решение системы (6.63). Тогда

$$\mathbf{F}(\mathbf{P}) = \mathbf{0}. \quad (6.69)$$

Подставляя (6.69) в интерполяционную функцию Ньютона (6.68), получим

$$\mathbf{G}(\mathbf{P}) = \mathbf{P}. \quad (6.70)$$

Иными словами, метод Ньютона – Рафсона фактически является итерацией неподвижной точки для функции (6.68), откуда следует, что *критерий сходимости метода Ньютона – Рафсона* может быть сформулирован следующим образом.

Если начальное приближение $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)})$ находится в некоторой достаточно малой окрестности решения задачи и во

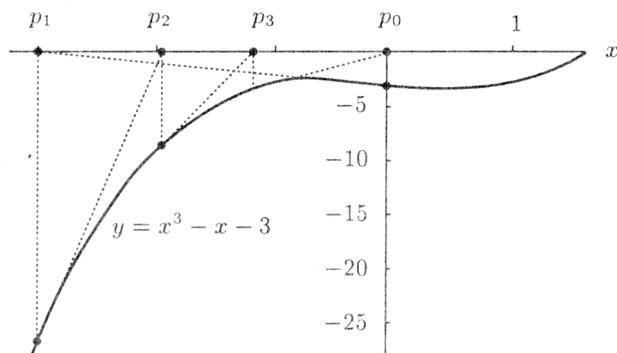


Рис. 6.9. Циклическая последовательность приближений к решению при использовании метода Ньютона – Рафсона

всех точках данной окрестности для интерполяционных функций Ньютона (6.68) выполняются неравенства

$$\begin{aligned} \left| \frac{\partial g_1}{x_1} \right| + \left| \frac{\partial g_1}{x_2} \right| + \cdots + \left| \frac{\partial g_1}{x_n} \right| &< 1; \\ \left| \frac{\partial g_2}{x_1} \right| + \left| \frac{\partial g_2}{x_2} \right| + \cdots + \left| \frac{\partial g_2}{x_n} \right| &< 1; \\ \left| \frac{\partial g_3}{x_1} \right| + \left| \frac{\partial g_3}{x_2} \right| + \cdots + \left| \frac{\partial g_3}{x_n} \right| &< 1; \\ &\vdots \\ \left| \frac{\partial g_n}{x_1} \right| + \left| \frac{\partial g_n}{x_2} \right| + \cdots + \left| \frac{\partial g_n}{x_n} \right| &< 1, \end{aligned} \tag{6.71}$$

метод Ньютона-Рафсона будет давать сходящуюся последовательность приближений к решению.

Глава 7

Классификация погрешности

Погрешности численного решения задачи, обусловленные различными причинами разделяют на [15]:

- 1) **неустранимую погрешность** – вызванную неточностью задания числовых данных, входящих в математическое описание задачи и несоответствием математического описания задачи реальности, последнюю часто называют погрешностью математической модели;
- 2) **погрешность метода** – вызванной методом решения, который часто не является точным или требует слишком большого числа арифметических операций;
- 3) **вычислительную погрешность** – вызванную тем, что при вводе, выводе данных в машину и при выполнении арифметических операций производятся округления.

При решении большинства задач нет смысла применять метод решения задачи с погрешностью, меньшей, чем величина неустранимой погрешности. Поэтому имея представления о величине неустранимой погрешности, можно разумно сформулировать требования к точности результатов численного решения задачи [15].

При требовании к точности численного решения задачи следует обратить внимание на следующие соображения:

- насколько груба математическая модель;
- насколько точно могут быть определены параметры модели;
- насколько точно может быть изготовлен данный прибор и проведены реальные измерения.

7.1. Абсолютная и относительная погрешности

Если x – точное значение искомой величины, а x^* – известное приближение к нему, то *абсолютной погрешностью* приближенного значения x^* называют величину $\Delta(x^*)$, про которую известно, что [15]

$$|x^* - x| \leq \Delta(x^*). \quad (7.1)$$

Относительной погрешностью приближенного значения называют некоторую величину $\delta(x^*)$, про которую известно, что [15]

$$\left| \frac{x^* - x}{x^*} \right| \leq \delta(x^*). \quad (7.2)$$

Относительную погрешность часто выражают в процентах.

Информацию о том, что x^* является приближенным значением с абсолютной погрешностью $\Delta(x^*)$, записывают в виде [15]

$$x = x^* \pm \Delta(x^*); \quad (7.3)$$

числа x^* и Δx^* принято записывать с одинаковым числом знаков после запятой, например

$$x = 1,123 \pm 0,004, \quad x = 1,123 \pm 4 \cdot 10^{-3}.$$

Информацию о том, что x^* является приближенным значением с относительной погрешностью $\delta(x^*)$, записывают в виде [15]

$$x = x^* (1 \pm \delta(x^*)), \quad (7.4)$$

например

$$x = 1,123 (1 \pm 0,003), \quad x = 1,123 (1 \pm 3 \cdot 10^{-3}), \quad x = 1,123 (1 \pm 0,3\%).$$

Абсолютную или относительную погрешность обычно записывают в виде числа, содержащего одну или две значащие цифры.

Заключение

Круг задач, описываемых уравнениями математической физики, чрезвычайно широк. В учебном пособии рассмотрены лишь отдельные уравнения математической физики, наиболее широко используемые в процессе создания элементной базы микросхем и микросистем, основные этапы численного решения задач, а также особенности задания граничных и начальных условий, методы дискретизации дифференциальных уравнений в частных производных, методы решения систем алгебраических уравнений.

Рассмотренные методы решения уравнений проиллюстрированы примерами исходных текстов MATLAB и C++ с комментариями и рекомендациями, позволяющими составить представление об основных правилах и приемах разработки компьютерных программ для решения уравнений математической физики.

Приложение А

Примеры решения задач математической физики в системе MATLAB

В данном разделе приведены примеры решения некоторых задач математической физики в системе инженерных и научных расчетов MATLAB 5.x.

А.1. Примеры решения уравнения Пуассона

Как было показано выше, многие задачи матфизики описываются уравнением Пуассона. К ним относятся задачи о стационарном распределении тепла, о диффузии вещества, о распределении электростатического поля в непроводящей среде при наличии электрических зарядов и многие другие.

А.1.1. Решение одномерного уравнения Пуассона методом конечных разностей

Решим одномерное уравнение Пуассона вида

$$\frac{\partial^2 u}{\partial x^2} = f(x), \quad (\text{А.1})$$

где x – координата; $u(x)$ – искомая функция; $f(x)$ – некоторая непрерывная функция на отрезке $[x_{min}, x_{max}]$ с граничными условиями Дирихле или Неймана в точках $x = x_{min}$, $x = x_{max}$.

Зададим на отрезке $[x_{min}, x_{max}]$ равномерную координатную сетку с шагом Δx :

$$\mathbf{x} = \{x_i, |i = 1, 2, \dots, n\}. \quad (\text{A.2})$$

Граничные условия первого рода (Дирихле) для рассматриваемой задачи могут быть представлены в виде

$$u(x_1) = g_1; \quad (\text{A.3})$$

$$u(x_n) = g_2, \quad (\text{A.4})$$

где x_1 , x_n – координаты граничных точек области $[x_{min}, x_{max}]$; g_1 , g_2 – некоторые константы.

Граничные условия второго рода (Неймана) для рассматриваемой задачи могут быть представлены в виде

$$\left. \frac{du}{dx} \right|_{x_1} = g_1; \quad (\text{A.5})$$

$$\left. \frac{du}{dx} \right|_{x_n} = g_2. \quad (\text{A.6})$$

Проводя дискретизацию граничных условий Дирихле на равномерной координатной сетке (A.2) с использованием метода конечных разностей, получим

$$u_1 = g_1; \quad (\text{A.7})$$

$$u_n = g_2, \quad (\text{A.8})$$

где u_1 , u_n – значения функции $u(x)$ в точках x_1 , x_n соответственно.

Проводя дискретизацию граничных условий Неймана на сетке (A.2), получим

$$\frac{u_2 - u_1}{\Delta x} = g_1; \quad (\text{A.9})$$

$$\frac{u_n - u_{n-1}}{\Delta x} = g_2. \quad (\text{A.10})$$

Проводя дискретизацию уравнения (A.1) для внутренних точек сетки, получим

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} = f_i, \quad i = 2, \dots, n-1, \quad (\text{A.11})$$

где f_i – значение функции $f(x)$ в точке сетки с координатой x_i .

Таким образом, в результате дискретизации получим систему линейных алгебраических уравнений размерностью n , содержащую $n-2$ уравнений вида (A.11) для внутренних точек области и уравнения (A.7) или (A.9) и (A.8) или (A.10) для двух граничных точек.

Ниже приведен один из вариантов функции для численного решения уравнения (A.1) с граничными условиями (A.3) – (A.6) на координатной сетке (A.2).

% Функция решения одномерного уравнения Пуассона

% $d^2u/dx^2=f(x,y)$

% с граничными условиями Дирихле и/или Неймана

```
function[x,u]=puass_1d(x0,xn,n,f,v1,g1,v2,g2)
```

% Входные параметры:

% x0 – начальная координата области решения;

% xn – конечная координата области решения;

% n – число точек координатной сетки;

% f – функция в правой части уравнения Пуассона,

% задаваемая строкой символов, заключенных

% в одинарные кавычки, например, 'exp(-x)+exp(-y)';

% v1 – параметр, значение которого определяет

% тип граничного условия на первой границе

% области $x = x(1)$ (1 – ГУ Дирихле, 2 – ГУ Неймана);

% g1 – граничное условие на первой границе в виде

% значения в одинарных кавычках, например, '0';

% v2 – параметр, значение которого определяет

% тип граничного условия на второй границе

% области $x = x(n)$ (1 – ГУ Дирихле, 2 – ГУ Неймана);

% g2 – граничное условие на второй границе в виде

% значения в одинарных кавычках.

% Выходные параметры:

```
% x - вектор-строка координатной сетки по оси x
%      размерностью 1 x n;
% U - матрица результирующих значений функции U
%      в узлах координатной сетки размерностью 1 x n.
```

```
% Функции и переменные по умолчанию
```

```
if exist('x0')==0
    x0=0;
end
if exist('xn')==0
    xn=5;
end
if exist('n')==0
    n=50;
end
if exist('f')==0
    f='2*sin(x^2)+10*cos(x^2)';
end
if exist('v1')==0
    v1=1;
end
if exist('g1')==0
    g1='0';
end
if exist('v2')==0
    v2=2;
end
if exist('g2')==0
    g2='-0.5';
end
```

```
% Задание равномерной координатной сетки
```

```
x=x0:(xn-x0)/(n-1):xn; dx=x(2)-x(1);
```

```
% Вычисление значений функций, заданных символьно,
```

% в узлах координатной сетки

```
F=inline(f,'x');  
G1=inline(g1,'x');  
G2=inline(g2,'x');
```

% Задание матрицы коэффициентов СЛАУ размерностью $n \times n$,
% все элементы которой равны 0

```
a=zeros(n,n);
```

% Задание матрицы-строки свободных членов СЛАУ
% размерностью $1 \times n$,
% все элементы которой равны 0

```
b=zeros(1,n);
```

% Определение коэффициентов и свободных членов СЛАУ,
% соответствующих граничным условиям и проверка
% корректности значений параметров v_1 , v_2

```
b(1)=G1(x(1));  
if v1==1  
    a(1,1)=1;  
elseif v1==2  
    a(1,1)=-1/dx;  
    a(1,2)=1/dx;  
else  
    error('Parameter v1 have incorrect value');  
end  
b(n)=G2(x(n));  
if v2==1  
    a(n,n)=1;  
elseif v2==2  
    a(n,n)=1/dx;  
    a(n,n-1)=-1/dx;  
else
```



```
error('Parameter v2 have incorrect value');
end

% Определение коэффициентов и свободных членов СЛАУ,
% соответствующих внутренним точкам области

for i=2:n-1
    a(i,i)=-2/dx^2;
    a(i,i+1)=1/dx^2;
    a(i,i-1)=1/dx^2;
    b(i)=F(x(i));
end

% Решение СЛАУ

u=b/a';

% Построение графика искомой функции u(x,y)

plot(x,u)
xlabel('x')
ylabel('u')
grid on
```

Строки, начинающиеся символом %, являются комментариями. Приведенное описание необходимо сохранить в виде текстового файла с именем `puass_1d.m` и поместить его в каталог `/WORK`, находящийся в корневом каталоге системы MATLAB. Вызов функции `puass_1d` может осуществляться следующими командами:

```
puass_1d;
x=puass_1d;
[x,u]=puass_1d;
[x,u]=puass_1d(x0,xn,n,f,v1,g1,v2,g2) .
```

При использовании первой, второй или третьей команд функция будет выводить графически решение задачи при входных параметрах, принятых по умолчанию.

При вызове функции с помощью первой или второй команд без символа «;» после текста команды, на экран монитора, помимо графического представления решения, будет выводиться вектор координат узлов сетки \mathbf{x} . При использовании символа «;» вектор координат узлов сетки выводиться не будет.

При использовании третьей или четвертой команд без символа «;» после текста команды, на экран монитора будет выводиться график решения, вектор координат узлов сетки \mathbf{x} и вектор значений искомой функции в узлах сетки \mathbf{u} .

Например, при вызове функции командой

```
puass_1d;
```

на экране появится график решения задачи для функции $f(x)$ в правой части уравнения (A.1), определяемой выражением

$$f(x) = 2 \sin(x^2) + 10 \cos(x^2), \quad (\text{A.12})$$

с граничными условиями

$$u(x_{min}) = 0; \quad (\text{A.13})$$

$$\left. \frac{du}{dx} \right|_{x_{max}} = 0.5, \quad (\text{A.14})$$

на границах $x_{min} = 0$, $x_{max} = 5$ на равномерной координатной сетке, содержащей 50 точек (рис. A.1).

При вызове функции командой

```
[x,u]=puass_1d(-1,6,100,'sin(x)',1,'1',2,'-0.5');
```

на экране появится график решения задачи для функции $f()$ в правой части уравнения (A.1),

$$f(x) = \sin(x), \quad (\text{A.15})$$

с граничными условиями

$$u(x_{min}) = 1; \quad (\text{A.16})$$

$$\left. \frac{du}{dx} \right|_{x_{max}} = -0.5, \quad (\text{A.17})$$

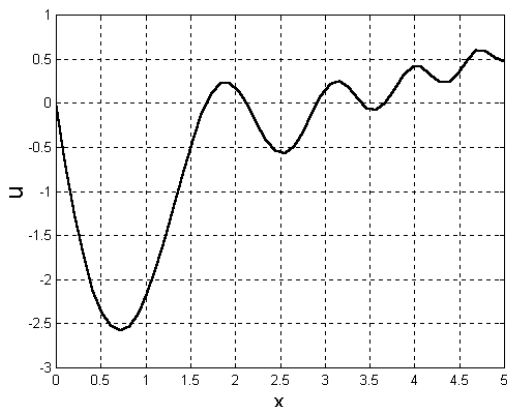


Рис. А.1. Решение одномерного уравнения Пуассона при помощи функции `puass_1d` с параметрами по умолчанию

на границах $x_{min} = -1$, $x_{max} = 6$ на равномерной координатной сетке, содержащей 100 точек (рис. А.2).

При вызове функции `puass_1d` со списком входных параметров (в круглых скобках) этот список не обязательно должен быть полным. При отсутствии некоторых параметров функция использует их значения по умолчанию.

А.1.2. Решение двухмерного уравнения Пуассона методом конечных разностей

Решим двумерное уравнение Пуассона вида

$$\frac{\partial^2 u}{\partial x^2} = f(x), \quad (\text{А.18})$$

где x, y – координаты; $u(x, y)$ – искомая функция; $f(x, y)$ – некоторая непрерывная функция на прямоугольной области с граничными условиями Дирихле или Неймана на границах $x = x_{min}$, $x = x_{max}$, $y = y_{min}$, $y = y_{max}$.

Зададим на отрезке $[x_{min}, x_{max}]$ равномерную координатную сетку с шагом Δ :

$$\mathbf{x} = \{x_i, |i = 1, 2, \dots, n\}, \quad (\text{А.19})$$

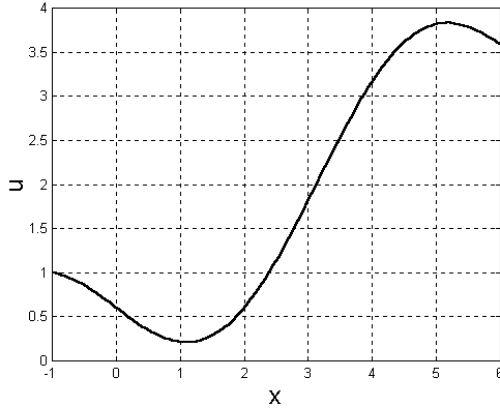


Рис. А.2. Решение одномерного уравнения Пуассона при помощи функции `puass_1d` с параметрами, заданными в командной строке

а на отрезке $[y_{min}, y_{max}]$ – равномерную координатную сетку с шагом Δy :

$$\mathbf{y} = \{y_i, |i = 1, 2, \dots, n\}. \quad (\text{A.20})$$

Векторы, заданные выражениями (A.19) и (A.20), определяют на прямоугольной области двухмерную равномерную сетку:

$$G = \{(x_i = i\Delta, y_j = j\Delta y), |i = 1, 2, \dots, n, j = 1, 2, \dots, m\}. \quad (\text{A.21})$$

Граничные условия первого рода (Дирихле) для рассматриваемой задачи могут быть представлены в виде

$$u(x_1, y) = g_1(y); \quad (\text{A.22})$$

$$u(x_n, y) = g_2(y); \quad (\text{A.23})$$

$$u(x, y_1) = g_3(x); \quad (\text{A.24})$$

$$u(x, y_m) = g_4(x), \quad (\text{A.25})$$

где x_1, x_n – координаты граничных точек области x_{min}, x_{max} ; y_1, y_m – координаты граничных точек области y_{min}, y_{max} ; $g_1(y), g_2(y), g_3(x), g_4(x)$ – некоторые непрерывные функции соответствующих координат.

Граничные условия второго рода (Неймана) для рассматриваемой задачи могут быть представлены в виде

$$\left. \frac{du}{dx} \right|_{x_1, y} = g_1(y); \quad (\text{A.26})$$

$$\left. \frac{du}{dx} \right|_{x_n, y} = g_2(y); \quad (\text{A.27})$$

$$\left. \frac{du}{dx} \right|_{x, y_1} = g_3(y); \quad (\text{A.28})$$

$$\left. \frac{du}{dx} \right|_{x, y_m} = g_4(y). \quad (\text{A.29})$$

$$(\text{A.30})$$

Проводя дискретизацию граничных условий Дирихле на равномерной координатной сетке (A.21) с использованием метода конечных разностей, получим

$$u_{1,j} = g_1(y_j); \quad (\text{A.31})$$

$$u_{n,j} = g_2(y_j); \quad (\text{A.32})$$

$$u_{i,1} = g_3(x_i); \quad (\text{A.33})$$

$$u_{i,m} = g_4(x_i), \quad (\text{A.34})$$

где $u_{1,j}$, $u_{n,j}$, $u_{i,1}$, $u_{i,m}$ – значения функции $u(x, y)$ в точках (x_1, y_j) , (x_n, y_j) , (x_i, y_1) , (x_i, y_m) соответственно.

Проводя дискретизацию граничных условий Неймана на сетке (A.21), получим

$$\frac{u_{2,j} - u_{1,j}}{\Delta x} = g_1(y_j); \quad (\text{A.35})$$

$$\frac{u_{n,j} - u_{n-1,j}}{\Delta x} = g_2(y_j); \quad (\text{A.36})$$

$$\frac{u_{i,2} - u_{i,1}}{\Delta y} = g_3(x_j); \quad (\text{A.37})$$

$$\frac{u_{i,m} - u_{i,m-1}}{\Delta y} = g_4(x_j). \quad (\text{A.38})$$

Проводя дискретизацию уравнения (A.18) для внутренних точек

сетки, получим

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = f_{i,j} \quad (\text{A.39})$$

$$i = 2, \dots, n-1; j = 2, \dots, m-1, \quad (\text{A.40})$$

где $f_{i,j}$ – значение функции $f(x, y)$ в точке сетки с координатой (x_i, y_j) .

Таким образом, в результате дискретизации получим систему линейных алгебраических уравнений размерностью $n \times m$, содержащую $(n-2)(m-2)$ уравнений вида (A.40) для внутренних точек области и $2n + 2(m-2)$ уравнений вида (A.31) или (A.35), (A.32) или (A.36), (A.33) или (A.37) и (A.34) или (A.38) для граничных точек.

Ниже приведен один из вариантов функции для численного решения уравнения (A.18) с граничными условиями (A.22) – (A.29) на сетке (A.21).

% Функция решения двухмерного уравнения Пуассона
 % $d^2u/dx^2 + d^2u/dy^2 = f(x, y)$
 % на прямоугольной области с граничными условиями
 % Дирихле и/или Неймана

```
function[x,y,U]=...
puass_2d(x0,xn,n,y0,ym,m,f,v1,g1,v2,g2,v3,g3,v4,g4)
% Входные параметры:
% x0 - начальная координата области решения по оси x;
% xn - конечная координата области решения по оси x;
% y0 - начальная координата области решения по оси y;
% ym - конечная координата области решения по оси y;
% n - число точек координатной сетки вдоль оси x;
% m - число точек координатной сетки вдоль оси y;
% f - функция в правой части уравнения Пуассона,
% задаваемая строкой символов, заключенных
% в одинарные кавычки, например, 'exp(-x)+exp(-y)';
% v1 - параметр, значение которого определяет
% тип граничного условия на первой границе
% области x = x(1) (1 - ГУ Дирихле, 2 - ГУ Неймана);
% g1 - функция в правой части граничного условия
```

```
%      на первой границе,  
%      задаваемая строкой символов, заключенных  
%      в одинарные кавычки;  
% v2 - параметр, значение которого определяет  
%      тип граничного условия на второй границе  
%      области  $x = x(n)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);  
% g2 - функция в правой части граничного условия  
%      на второй границе,  
%      задаваемая строкой символов, заключенных  
%      в одинарные кавычки;  
% v3 - параметр, значение которого определяет  
%      тип граничного условия на третьей границе  
%      области  $y = y(1)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);  
% g3 - функция в правой части граничного условия  
%      на третьей границе,  
%      задаваемая строкой символов, заключенных  
%      в одинарные кавычки;  
% v4 - параметр, значение которого определяет  
%      тип граничного условия на четвертой границе  
%      области  $y = y(m)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);  
% g4 - функция в правой части граничного условия  
%      на четвертой границе,  
%      задаваемая строкой символов, заключенных  
%      в одинарные кавычки.
```

```
% Выходные параметры:
```

```
% x - вектор-строка координатной сетки по оси x  
%      размерностью 1 x n;  
% y - вектор-строка координатной сетки по оси y  
%      размерностью 1 x m;  
% U - матрица результирующих значений функции U  
%      в узлах координатной сетки размерностью n x m.
```

```
% Функции и переменные по умолчанию
```

```
if exist('x0')==0  
    x0=0;
```

```
end
if exist('xn')==0
xn=1;
end
if exist('n')==0
n=10;
end
if exist('y0')==0
y0=0;
end
if exist('ym')==0
ym=2;
end
if exist('m')==0
m=20;
end
if exist('f')==0
f='exp(-x)+exp(-y)';
end
if exist('v1')==0
v1=1;
end
if exist('g1')==0
g1='sin(y^2)';
end
if exist('v2')==0
v2=1;
end
if exist('g2')==0
g2='cos(3*y)';
end
if exist('v3')==0
v3=2;
end
if exist('g3')==0
g3='10*sin(x^2)';
end
```



```
if exist('v4')==0
v4=2;
end
if exist('g4')==0
g4='10*sin(6*x)';
end
```

% Задание равномерной координатной сетки

```
x=x0:(xn-x0)/(n-1):xn; dx=x(2)-x(1);
y=y0:(ym-y0)/(m-1):ym; dy=y(2)-y(1);
```

% Вычисление значений функций, заданных символьно,
% в узлах координатной сетки

```
F=inline(f,'x','y');
G1=inline(g1,'y');
G2=inline(g2,'y');
G3=inline(g3,'x');
G4=inline(g4,'x');
```

% Определение размерности СЛАУ

```
N=n*m;
```

% Задание матрицы коэффициентов СЛАУ размерностью N x N,
% все элементы которой равны 0

```
a=zeros(N,N);
```

% Задание матрицы-строки свободных членов СЛАУ
% размерностью 1 x N, все элементы которой равны 0

```
b=zeros(1,N);
```

% Определение коэффициентов и свободных членов СЛАУ,
% соответствующих граничным условиям, и проверка

% корректности значений параметров v1, v2, v3, v4

```
for j=1:m
    b(j)=G1(y(j));
    if v1==1
        a(j,j)=1;
    elseif v1==2
        a(j,j)=-1/dx;
        a(j,m+j)=1/dx;
    else
        error('Parameter v1 have incorrect value');
    end
    b(m*(n-1)+j)=G2(y(j));
    if v2==1
        a(m*(n-1)+j,m*(n-1)+j)=1;
    elseif v2==2
        a(m*(n-1)+j,m*(n-1)+j)=1/dx;
        a(m*(n-1)+j,m*(n-2)+j)=-1/dx;
    else
        error('Parameter v2 have incorrect value');
    end
end
for i=2:n-1
    b(m*(i-1)+1)=G3(x(i));
    if v3==1
        a(m*(i-1)+1,m*(i-1)+1)=1;
    elseif v3==2
        a(m*(i-1)+1,m*(i-1)+1)=-1/dy;
        a(m*(i-1)+1,m*(i-1)+2)=1/dy;
    else
        error('Parameter v3 have incorrect value');
    end
    b(m*(i-1)+m)=G4(x(i));
    if v4==1
        a(m*(i-1)+m,m*(i-1)+m)=1;
    elseif v4==2
        a(m*(i-1)+m,m*(i-1)+m)=1/dy;
```

```
a(m*(i-1)+m,m*(i-1)+m-1)=-1/dy;
else
    error('Parameter v4 have incorrect value');
end
end

% Определение коэффициентов и свободных членов СЛАУ,
% соответствующих внутренним точкам области

for i=2:n-1
    for j=2:m-1
        a(m*(i-1)+j,m*(i-1)+j)=-2/dx^2-2/dy^2;
        a(m*(i-1)+j,m*(i)+j)=1/dx^2;
        a(m*(i-1)+j,m*(i-2)+j)=1/dx^2;
        a(m*(i-1)+j,m*(i-1)+j+1)=1/dy^2;
        a(m*(i-1)+j,m*(i-1)+j-1)=1/dy^2;
        b(m*(i-1)+j)=F(x(i),y(j));
    end
end

% Решение СЛАУ

u=b/a';

% Преобразование вектора-строки значений искомой функции
% в узлах координатной сетки в матрицу размерностью
% n x m, удобную для представления результатов
% в графическом виде

for i=1:n
    for j=1:m
        U(i,j)=u(m*(i-1)+j);
    end
end

% Построение графика искомой функции U(x,y)
```

```
surf(y,x,U)
xlabel('y')
ylabel('x')
zlabel('U')
grid on
```

Приведенное описание необходимо сохранить в виде текстового файла с именем `puass_2d.m` и поместить его в каталог `/WORK`, находящийся в корневом каталоге системы MATLAB.

Вызов функции `puass_2d` может осуществляться следующими командами:

```
puass_2d;
x=puass_2d;
[x,y]=puass_2d;
[x,y,U]=puass_2d;
[x,y,U]=puass_2d(x0,xn,n,y0,ym,m,f,v1,g1,v2,g2,v3,g3,v4,g4).
```

При использовании первой, второй, третьей или четвертой команд функция будет выводить графически решение задачи при входных параметрах, принятых по умолчанию.

При вызове функции с помощью первой или второй команд без символа «;» после текста команды, на экран монитора, помимо графического представления решения, будет выводиться вектор координат узлов сетки **x**. При использовании символа «;» вектор координат узлов сетки выводиться не будет.

При использовании третьей команды без символа «;» после текста команды, на экран монитора будет выводиться график решения, вектор координат узлов сетки **x** и вектор координат узлов сетки **y**. При использовании третьей или четвертой команд без символа «;» после текста команды на экран монитора будет выводиться график решения, вектор координат узлов сетки **x**, вектор координат узлов сетки **y** и вектор значений искомой функции в узлах сетки **u**.

Например, при вызове функции командой

```
puass_2d;
```

на экране появится график решения задачи для функции $f(y)$ в правой части уравнения (A.18), определяемой выражением

$$f(x, y) = \exp(-x) + \exp(-y), \quad (\text{A.41})$$

с граничными условиями

$$u(x_{min}, y) = \sin(y^2); \quad (\text{A.42})$$

$$u(x_{max}, y) = \cos(3y); \quad (\text{A.43})$$

$$\left. \frac{\partial u}{\partial y} \right|_{x, y_{min}} = 10 \sin(x^2); \quad (\text{A.44})$$

$$\left. \frac{\partial u}{\partial y} \right|_{x, y_{max}} = 10 \sin(6x); \quad (\text{A.45})$$

на границах $x_{min} = 0$, $x_{max} = 1$, $y_{min} = 0$, $y_{max} = 2$ на равномерной координатной сетке, содержащей 200 точек (10×20) (рис. A.3).

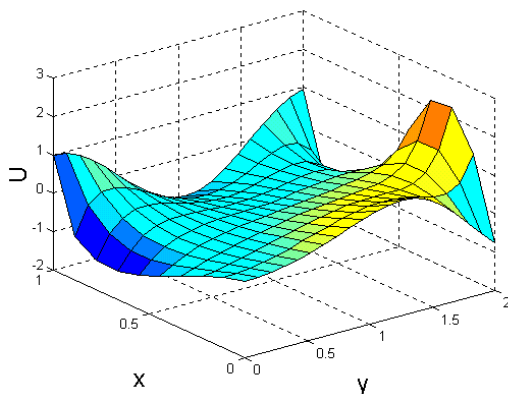


Рис. A.3. Решение двухмерного уравнения Пуассона при помощи функции `puass_2d` с параметрами по умолчанию

При вызове функции командой

```
[x,y,U]=puass_2d(-1,2,20,0.35,5,30,'x^2+3*y',1,...
'2*y*sin(3*y)',1,'3*y*cos(4*y)',2,'2-x',2,'3*x^2');
```

на экране появится график решения задачи для функции $f(x, y)$ в правой части уравнения (A.18), определяемой выражением

$$f(x, y) = x^2 + 3y, \quad (\text{A.46})$$

с граничными условиями

$$u(x_{min}, y) = 2y \sin(3y); \quad (\text{A.47})$$

$$u(x_{max}, y) = 3y \cos(4y); \quad (\text{A.48})$$

$$\left. \frac{\partial u}{\partial y} \right|_{x, y_{min}} = 2 - x; \quad (\text{A.49})$$

$$\left. \frac{\partial u}{\partial y} \right|_{x, y_{max}} = 3x^2; \quad (\text{A.50})$$

на границах $x_{min} = -1$, $x_{max} = 1$, $y_{min} = 0.35$, $y_{max} = 5$ на равномерной координатной сетке, содержащей 600 точек (20×30) (рис. A.4).

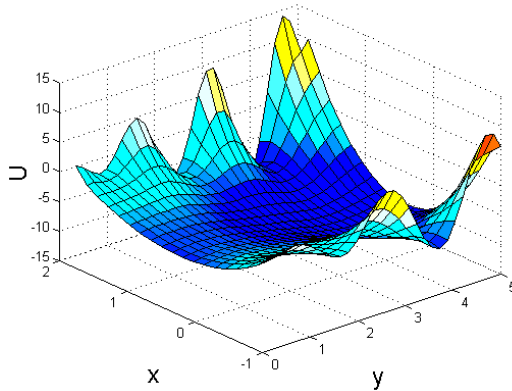


Рис. A.4. Решение двухмерного уравнения Пуассона при помощи функции `puass_2d` с параметрами, заданными в командной строке

При вызове функции `puass_2d` список входных параметров не обязательно должен быть полным. При отсутствии некоторых параметров функция использует их значения по умолчанию.

A.1.3. Решение двухмерного уравнения Пуассона методом конечных элементов

Решим двухмерное уравнение Пуассона (A.18) с правой частью, определяемой выражением (A.41) и граничными условиями (A.42)

– (A.45) на границах $x_{min} = 0$, $x_{max} = 1$, $y_{min} = -1$, $y_{max} = 1$ на триангулярной координатной сетке методом конечных элементов.

Для решения дифференциальных уравнений в частных производных методом конечных элементов в системе MATLAB 5.x воспользуемся приложением `pdetool`.

Запуск приложения осуществляется командой

`pdetool`.

При этом на экране монитора отображается главное окно приложения (рис. A.5).

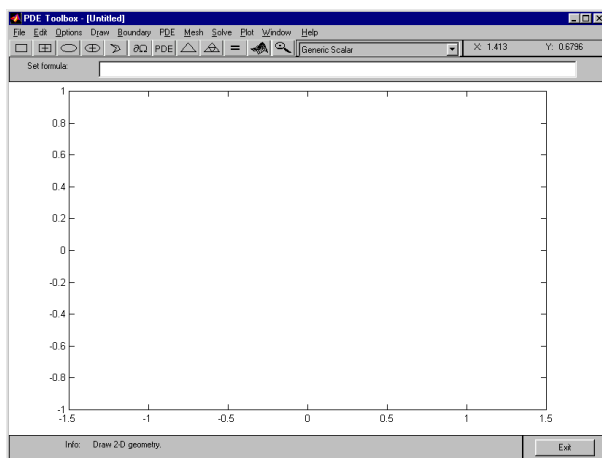


Рис. A.5. Главное окно приложения `pdetool`

Для задания прямоугольной области решения задачи необходимо активизировать манипулятором «мышь» кнопку с символом \square , после чего навести курсор «мыши» на рабочее поле редактора, нажать левую кнопку «мыши» в левом верхнем углу $(0; 1)$ задаваемой прямоугольной области, переместить курсор в правый нижний угол $(1; -1)$ области, удерживая левую кнопку, после чего отпустить ее. Прямоугольная область будет зафиксирована (рис. A.6).

При необходимости корректировки координат и размеров области нужно навести курсор «мыши» на изображение прямоугольника и дважды щелкнуть левой кнопкой. На экране появится окно

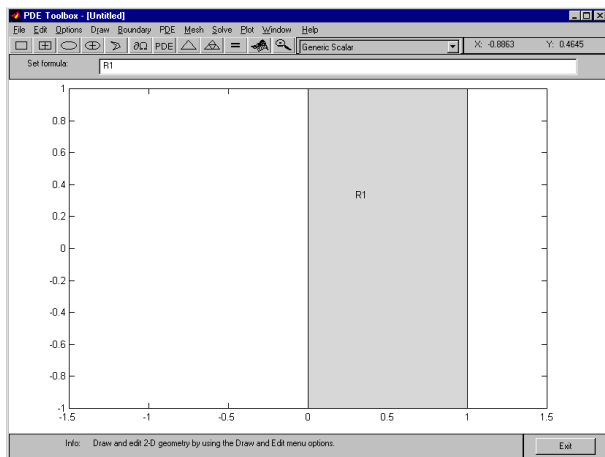


Рис. А.6. Задание прямоугольной области решения задачи в приложении pdetool

редактирования параметров области с соответствующими полями (рис. А.7). В первом и втором полях отображаются координаты левой нижней точки прямоугольника по осям x и y соответственно. В третьем поле – ширина прямоугольника, в четвертом – высота, в пятом – условное обозначение.

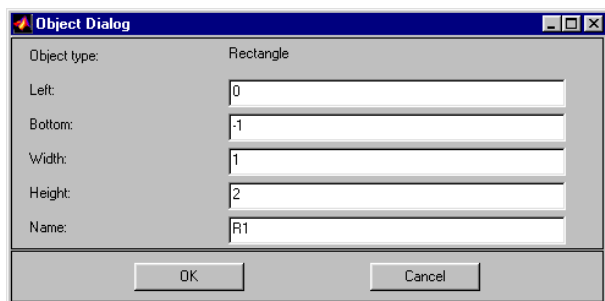


Рис. А.7. Окно редактирования параметров прямоугольной области решения задачи в приложении pdetool

Области решения произвольной формы могут быть заданы ана-

логичным образом с использованием кнопок с изображениями прямоугольников, эллипсов и полигона. При этом результирующая область может быть определена как объединение или разность нескольких областей простой формы. Для этого в поле Set formula указываются условные обозначения областей, связанные знаками «+» в случае объединения или «-» в случае разности (см. рис. A.6).

Для задания граничных условий необходимо активизировать манипулятором «мышь» кнопку с символом $\partial\Omega$, в результате чего окно приложения примет вид, показанный на рис. A.8.

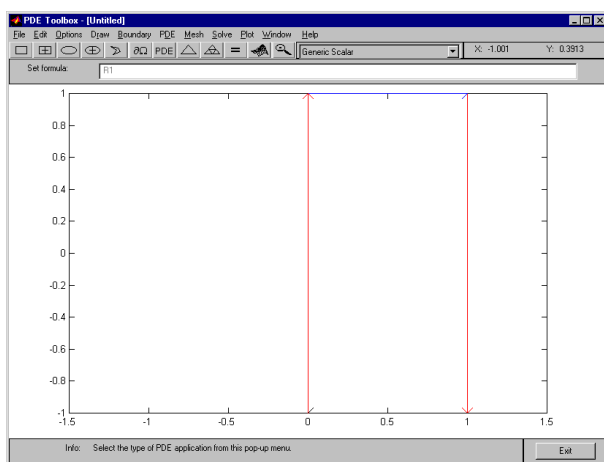


Рис. A.8. Задание граничных условий в приложении pdetool

Все границы области показаны линиями со стрелками, причем по умолчанию на них заданы условия Дирихле (красные линии). Для редактирования граничных условий необходимо дважды щелкнуть левой кнопкой «мыши» на выбранной границе и внести соответствующие изменения в полях окна редактирования граничных условий (рис. A.9 – A.12).

Точки после символов x и y означают, что действие, знак которого стоит после точки, применяется к каждому элементу матрицы x или y .

Для редактирования вида дифференциального уравнения и ввода его функций и коэффициентов необходимо активизировать манипулятором «мышь» кнопку с символами PDE, после чего внести

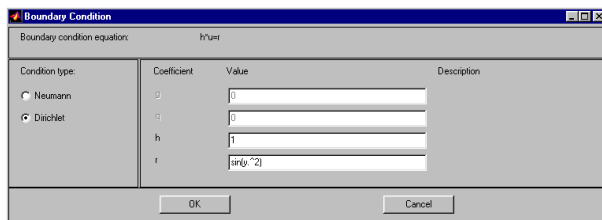


Рис. А.9. Задание граничных условий на левой границе

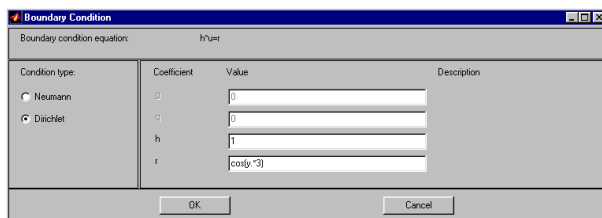


Рис. А.10. Задание граничных условий на правой границе

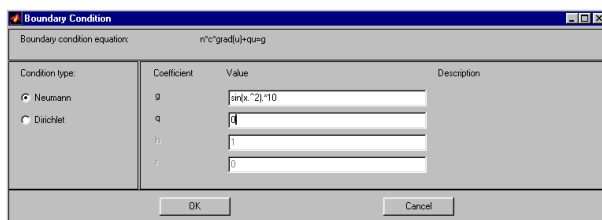



Рис. А.11. Задание граничных условий на нижней границе

соответствующие изменения в полях окна редактирования, показанного на рис. А.13.

Формирование триангулярной сетки с использованием метода Делоне осуществляется активизацией кнопки с символом \triangle (рис.А.14). При необходимости увеличения числа узлов сетки следует активизировать кнопку  (рис.А.15).

Решение задачи осуществляется при активизации кнопки с символом « = ». По умолчанию значения функции решения выделены

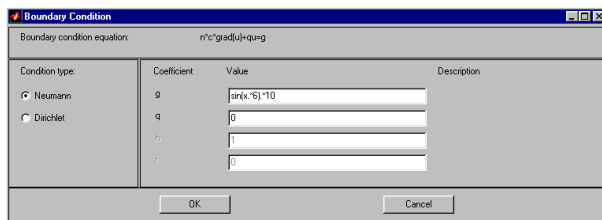


Рис. А.12. Задание граничных условий на верхней границе

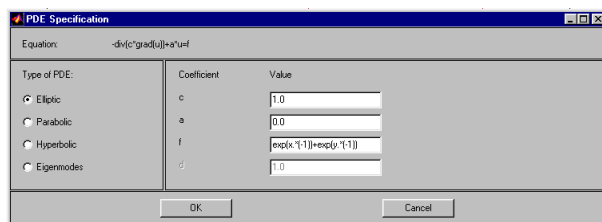



Рис. А.13. Окно редактирования уравнения задачи в приложении pdetool

цветом (рис. А.16).

Для графического вывода решения задачи в виде трехмерного (3D) изображения следует активизировать кнопку , после чего в появившемся окне редактирования параметров изображения внести в соответствующие поля данные, как показано на рис. А.17.

При активизации кнопки Plot на экран будет выведен график, показанный на рис. А.18.

Более подробно с возможностями приложения pdetool можно ознакомиться в справке HELP системы MATLAB или в [17, 18].

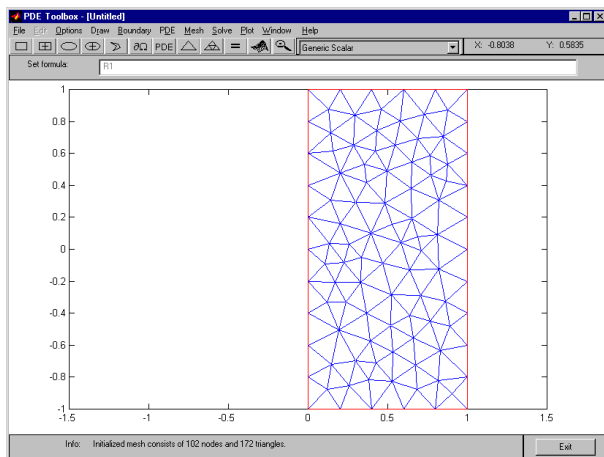


Рис. А.14. Генерация триангулярной сетки с использованием метода Делоне в приложении pde-tool

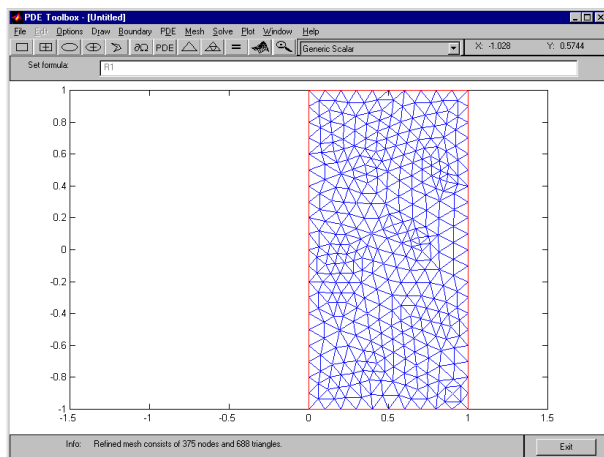


Рис. А.15. Увеличение числа элементов сетки в приложении pde-tool

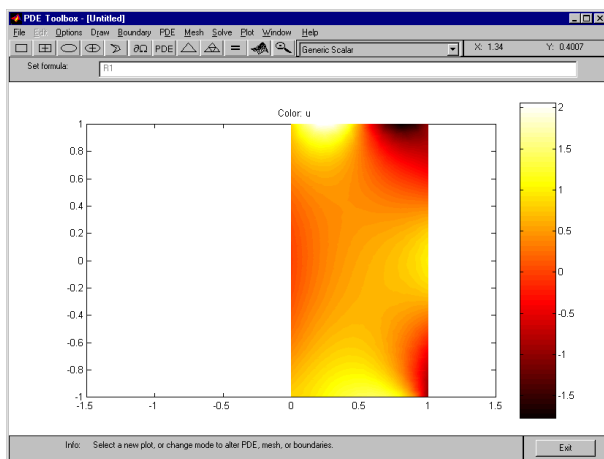


Рис. А.16. Решение уравнения Пуассона в приложении pdeplot

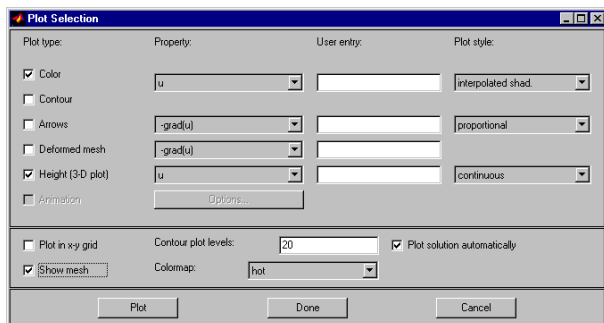


Рис. А.17. Окно редактирования параметров графического представления решения в приложении pdeplot

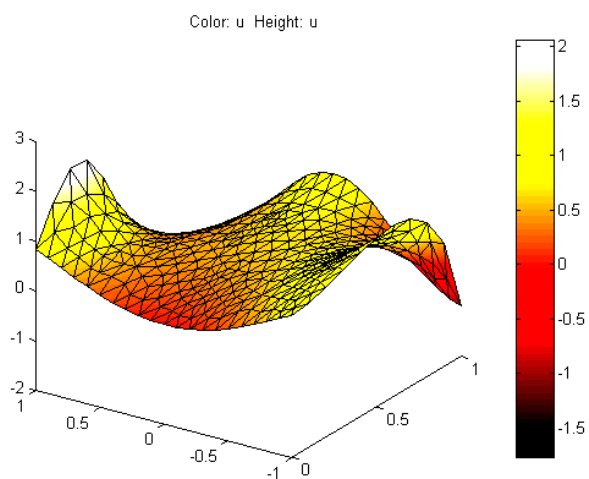


Рис. А.18. Представление решения уравнения Пуассона в виде 3D-изображения

А.2. Примеры решения уравнения теплопроводности

В качестве примера решения уравнений параболического типа рассмотрим нестационарное уравнение теплопроводности

$$\rho(x, y)C(x, y)\frac{\partial T}{\partial t} - \Delta(k(x, y)\Delta T) = f(x, y), \quad (\text{A.51})$$

где t – время; x, y – координаты; $T(x, y)$ – искомая функция распределения абсолютной температуры по координатам; $\rho(x, y)$ – плотность вещества; $C(x, y)$ – удельная теплоемкость вещества; $k(x, y)$ – коэффициент теплопроводности вещества; $f(x, y)$ – плотность мощности источников тепла на прямоугольной области с граничными условиями Дирихле или Неймана на границах $x = x_{\min}$, $x = x_{\max}$, $y = y_{\min}$, $y = y_{\max}$ и начальными условиями первого или второго рода на отрезке времени $[t_{\min}, t_{\max}]$. Зададим на отрезке $[x_{\min}, x_{\max}]$ равномерную координатную сетку с шагом Δx :

$$\mathbf{x} = \{x_i | i = 1, 2, \dots, n\}, \quad (\text{A.52})$$

на отрезке $[y_{\min}, y_{\max}]$ – равномерную координатную сетку с шагом Δy :

$$\mathbf{y} = \{y_j | j = 1, 2, \dots, m\}, \quad (\text{A.53})$$

на отрезке $[t_{\min}, t_{\max}]$ – равномерную сетку с шагом Δt :

$$\mathbf{t} = \{t_l | l = 1, 2, \dots, s\}. \quad (\text{A.54})$$

Векторы, заданные выражениями (A.52) – (A.54), определяют на прямоугольной области равномерную пространственно-временную сетку:

$$G = \{x_i, y_j, t_l | i = 1, 2, \dots, n, j = 1, 2, \dots, m, l = 1, 2, \dots, s\}. \quad (\text{A.55})$$

Граничные условия первого рода (Дирихле) для рассматриваемой задачи могут быть представлены в виде

$$T(x_1, y, t) = g_1(y); \quad (\text{A.56})$$

$$T(x_n, y, t) = g_2(y); \quad (\text{A.57})$$

$$T(x, y_1, t) = g_3(y); \quad (\text{A.58})$$

$$T(x, y_m, t) = g_4(y), \quad (\text{A.59})$$

где x_1, x_n – координаты граничных точек области x_{min}, x_{max} ; y_1, y_m – координаты граничных точек области y_{min}, y_{max} ; $g_1(y), g_2(y), g_3(x), g_4(x)$ – некоторые непрерывные функции соответствующих координат.

Граничные условия второго рода (Неймана) для рассматриваемой задачи могут быть представлены в виде

$$\left. \frac{\partial T}{\partial x} \right|_{x_1, y, t} = g_1(y); \quad (\text{A.60})$$

$$\left. \frac{\partial T}{\partial x} \right|_{x_n, y, t} = g_2(y); \quad (\text{A.61})$$

$$\left. \frac{\partial T}{\partial y} \right|_{x, y_1, t} = g_3(x); \quad (\text{A.62})$$

$$\left. \frac{\partial T}{\partial y} \right|_{x, y_m, t} = g_4(x). \quad (\text{A.63})$$

$$(\text{A.64})$$

Начальные условия первого рода для рассматриваемой задачи могут быть представлены в виде

$$T(x, y, t_1) = g_t(x, y), \quad (\text{A.65})$$

где t_1 – начальный момент времени; $g_t(x, y)$ – некоторая непрерывная функция соответствующих координат.

Начальные условия второго рода для рассматриваемой задачи могут быть представлены в виде

$$\left. \frac{\partial T}{\partial t} \right|_{x, y, t_1} = g_t(x, y). \quad (\text{A.66})$$

$$(\text{A.67})$$

Проводя дискретизацию граничных условий Дирихле на равномерной сетке (A.55) с использованием метода конечных разностей, получим

$$T_{1,j,l} = g_1(y_j); \quad (\text{A.68})$$

$$T_{n,j,l} = g_2(y_j); \quad (\text{A.69})$$

$$T_{i,1,l} = g_3(x_i); \quad (\text{A.70})$$

$$T_{i,m,l} = g_4(x_i), \quad (\text{A.71})$$

где $T_{1,j,l}$, $T_{n,j,l}$, $T_{i,1,l}$, $T_{i,m,l}$ – значения функции $T(x, y, t)$ в точках (x_1, y_j, t_l) , (x_n, y_j, t_l) , (x_i, y_1, t_l) , (x_i, y_m, t_l) соответственно.

Проводя дискретизацию граничных условий Неймана на сетке (A.55), получим

$$\frac{T_{2,j,l} - T_{1,j,l}}{\Delta x} = g_1(y_j); \quad (\text{A.72})$$

$$\frac{T_{n,j,l} - T_{n-1,j,l}}{\Delta x} = g_2(y_j); \quad (\text{A.73})$$

$$\frac{T_{i,2,l} - T_{i,1,l}}{\Delta y} = g_3(x_j); \quad (\text{A.74})$$

$$\frac{T_{i,m,l} - T_{i,m-1,l}}{\Delta y} = g_4(x_j). \quad (\text{A.75})$$

Проводя дискретизацию начальных условий первого рода на равномерной сетке (A.55), получим

$$T_{i,j,1} = g_t(x_i, y_j), \quad (\text{A.76})$$

где $T_{i,j,1}$ – значения функции $T(x, y, t)$ в точке (x_i, y_j, t_1) .

Проводя дискретизацию начальных условий второго рода на сетке (A.55), получим

$$\frac{T_{i,j,2} - T_{i,j,2}}{\Delta t} = g_t(x_i, y_j). \quad (\text{A.77})$$

Проводя дискретизацию уравнения (A.51) для внутренних точек сетки, получим

$$\begin{aligned} & \rho_{i,j,l} C_{i,j,l} \frac{T_{i,j,l} - T_{i,j,l-1}}{\Delta t} - \\ & - \frac{1}{\Delta x^2} (k_{i,j,l} (T_{i+1,j,l} - T_{i,j,l}) - k_{i-1,j,l} (T_{i,j,l} - T_{i-1,j,l})) - \\ & - \frac{1}{\Delta y^2} (k_{i,j,l} (T_{i,j+1,l} - T_{i,j,l}) - k_{i,j-1,l} (T_{i,j,l} - T_{i,j-1,l})) = f_{i,j,l} \\ & i = 2, \dots, n-1; \quad j = 2, \dots, m-1; \quad l = 2, \dots, s, \end{aligned} \quad (\text{A.78})$$

где $f_{i,j,l}$ – значение функции $f(x, y, t)$ в точке сетки с координатами (x_i, y_j, t_l) .

Таким образом, в результате дискретизации получим систему линейных алгебраических уравнений размерностью $n \times m \times s$.

Ниже приведен один из вариантов функции для численного решения уравнения (A.51) с граничными условиями (A.56) – (A.63) и начальными условиями (A.65) или (A.66) на равномерной сетке (A.55) с подробными комментариями.

```
% Функция решения двухмерного нестационарного
% уравнения теплопроводности
%  $r(x,y)C(x,y)dT/dt - d/dx(k(x,y)dT/dx) -$ 
%  $- d/dy(k(x,y)dT/dy) = f(x,y)$ 
% на прямоугольной области с граничными условиями
% Дирихле и/или Неймана

function[x,y,t,T]= ...
termo_2d(t0,ts,s,x0,xn,n,y0,ym,m,r,c,k,f, ...
vt,gt1,v1,g1,v2,g2,v3,g3,v4,g4)

% Входные параметры:
% t0 - начальный момент времени, с;
% ts - конечный момент времени, с;
% x0 - начальная координата области решения по оси x, м;
% xn - конечная координата области решения по оси x, м;
% y0 - начальная координата области решения по оси y, м;
% ym - конечная координата области решения по оси y, м;
% n - число точек координатной сетки вдоль оси x;
% m - число точек координатной сетки вдоль оси y;
% s - число точек сетки вдоль оси времени t;
% r - функция плотности вещества,
% задаваемая строкой символов, заключенных
% в одинарные кавычки, например, 'x+2*y', кг/м^3;
% c - функция теплоемкости вещества,
% задаваемая строкой символов, заключенных
% в одинарные кавычки, Дж/(кг K);
% k - функция теплопроводности вещества,
% задаваемая строкой символов, заключенных
% в одинарные кавычки, Вт/(м K);
% f - функция плотности мощности источников тепла,
```

```
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки, Вт/м^3;
% vt - параметр, значение которого определяет
%      тип начального условия
%      (1 - Дирихле, 2 - Неймана);
% gt1- функция в правой части начального условия,
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки, К или К/с;
% v1 - параметр, значение которого определяет
%      тип граничного условия на первой границе
%      области  $x = x(1)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);
% g1 - функция в правой части граничного условия
%      на первой границе,
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки, К или К/м;
% v2 - параметр, значение которого определяет
%      тип граничного условия на второй границе
%      области  $x = x(n)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);
% g2 - функция в правой части граничного условия
%      на второй границе,
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки, К или К/м;
% v3 - параметр, значение которого определяет
%      тип граничного условия на третьей границе
%      области  $y = y(1)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);
% g3 - функция в правой части граничного условия
%      на третьей границе,
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки, К или К/м;
% v4 - параметр, значение которого определяет
%      тип граничного условия на четвертой границе
%      области  $y = y(m)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);
% g4 - функция в правой части граничного условия
%      на четвертой границе,
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки, К или К/м.
```

```
% Выходные параметры:
% x - вектор-строка координатной сетки по оси x
%     размерностью 1 x n;
% y - вектор-строка координатной сетки по оси y
%     размерностью 1 x m;
% t - вектор-строка сетки по оси времени
%     размерностью 1 x s;
% T - матрица результирующих значений температуры
%     в узлах координатной сетки размерностью n x m x s.

% Функции и переменные по умолчанию

if exist('t0')==0
    t0=0;
end
if exist('ts')==0
    ts=6;
end
if exist('s')==0
    s=6;
end
if exist('x0')==0
    x0=0;
end
if exist('xn')==0
    xn=5e-2;
end
if exist('n')==0
    n=10;
end
if exist('y0')==0
    y0=0;
end
if exist('ym')==0
    ym=8e-2;
end
if exist('m')==0
```

```
m=20;
end
if exist('r')==0
r='2e3';
end
if exist('c')==0
c='100';
end
if exist('k')==0
k='20';
end
if exist('f')==0
f='0';
end
if exist('vt')==0
vt=1;
end
if exist('gt1')==0
gt1='10*(sign(1e2*x-2)-sign(1e2*x-3)+ ...
      sign(1e2*y-3)-sign(1e2*y-5))+300';
end
if exist('v1')==0
v1=1;
end
if exist('g1')==0
g1='300';
end
if exist('v2')==0
v2=1;
end
if exist('g2')==0
g2='300';
end
if exist('v3')==0
v3=1;
end
if exist('g3')==0
```

```
g3='300';
end
if exist('v4')==0
v4=1;
end
if exist('g4')==0
g4='300';
end
```

% Задание равномерной сетки

```
x=x0:(xn-x0)/(n-1):xn; dx=x(2)-x(1);
y=y0:(ym-y0)/(m-1):ym; dy=y(2)-y(1);
t=t0:(ts-t0)/(s-1):ts; dt=t(2)-t(1);
```

% Вычисление значений функций, заданных символьно,
% в узлах координатной сетки

```
F=inline(f,'x','y');
R=inline(r,'x','y');
C=inline(c,'x','y');
K=inline(k,'x','y');
GT=inline(gt1,'x','y');
G1=inline(g1,'y');
G2=inline(g2,'y');
G3=inline(g3,'x');
G4=inline(g4,'x');
```

% Определение размерности СЛАУ

```
N=s*n*m;
```

% Задание матрицы коэффициентов СЛАУ размерностью N x N,
% все элементы которой равны 0

```
a=zeros(N,N);
```

```
% Задание матрицы-строки свободных членов СЛАУ
% размерностью 1 x N, все элементы которой равны 0

b=zeros(1,N);

% Определение коэффициентов и свободных членов СЛАУ,
% соответствующих граничным условиям, и проверка
% корректности значений параметров vt, v1, v2, v3, v4

for i=1:n
    for j=1:m
        b(m*(i-1)+j)=GT(x(i),y(j));
        if vt==1
            a(m*(i-1)+j,m*(i-1)+j)=1;
        elseif vt==2
            a(m*(i-1)+j,m*(i-1)+j)=-1/dt;
            a(m*(i-1)+j,n*m+m*(i-1)+j)=1/dt;
        else
            error('Parameter vt have incorrect value');
        end
    end
end
for l=1:s
    for j=1:m
        b(n*m*(l-1)+j)=G1(y(j));
        if v1==1
            a(n*m*(l-1)+j,n*m*(l-1)+j)=1;
        elseif v1==2
            a(n*m*(l-1)+j,n*m*(l-1)+j)=-1/dx;
            a(n*m*(l-1)+j,n*m*(l-1)+m+j)=1/dx;
        else
            error('Parameter v1 have incorrect value');
        end
    end
    b(n*m*(l-1)+m*(n-1)+j)=G2(y(j));
    if v2==1
        a(n*m*(l-1)+m*(n-1)+j,n*m*(l-1)+m*(n-1)+j)=1;
    elseif v2==2
```

```

        a(n*m*(l-1)+m*(n-1)+j,n*m*(l-1)+m*(n-1)+j)=1/dx;
        a(n*m*(l-1)+m*(n-1)+j,n*m*(l-1)+m*(n-2)+j)= ...
            -1/dx;
    else
        error('Parameter v2 have incorrect value');
    end
end
for i=2:n-1
    b(n*m*(l-1)+m*(i-1)+1)=G3(x(i));
    if v3==1
        a(n*m*(l-1)+m*(i-1)+1,n*m*(l-1)+m*(i-1)+1)=1;
    elseif v3==2
        a(n*m*(l-1)+m*(i-1)+1,n*m*(l-1)+m*(i-1)+1)= ...
            -1/dy;
        a(n*m*(l-1)+m*(i-1)+1,n*m*(l-1)+m*(i-1)+2)=1/dy;
    else
        error('Parameter v3 have incorrect value');
    end
    b(n*m*(l-1)+m*(i-1)+m)=G4(x(i));
    if v4==1
        a(n*m*(l-1)+m*(i-1)+m,n*m*(l-1)+m*(i-1)+m)=1;
    elseif v4==2
        a(n*m*(l-1)+m*(i-1)+m,n*m*(l-1)+m*(i-1)+m)=1/dy;
        a(n*m*(l-1)+m*(i-1)+m,n*m*(l-1)+m*(i-1)+m-1)=...
            -1/dy;
    else
        error('Parameter v4 have incorrect value');
    end
end
end

% Определение коэффициентов и свободных членов СЛАУ,
% соответствующих внутренним точкам области

for l=2:s
    for i=2:n-1
        for j=2:m-1

```



```

        a(n*m*(l-1)+m*(i-1)+j,n*m*(l-1)+m*(i-1)+j)=...
            +R(x(i),y(j))*C(x(i),y(j))/dt+...
            (K(x(i),y(j))+K(x(i-1),y(j)))/dx^2+...
            (K(x(i),y(j))+K(x(i),y(j-1)))/dy^2;
    a(n*m*(l-1)+m*(i-1)+j,n*m*(l-1)+m*i+j)=...
        -K(x(i),y(j))/dx^2;
    a(n*m*(l-1)+m*(i-1)+j,n*m*(l-1)+m*(i-2)+j)=...
        -K(x(i-1),y(j))/dx^2;
    a(n*m*(l-1)+m*(i-1)+j,n*m*(l-1)+m*(i-1)+j+1)=...
        -K(x(i),y(j))/dy^2;
    a(n*m*(l-1)+m*(i-1)+j,n*m*(l-1)+m*(i-1)+j-1)=...
        -K(x(i),y(j-1))/dy^2;
    a(n*m*(l-1)+m*(i-1)+j,n*m*(l-2)+m*(i-1)+j)=...
        -R(x(i),y(j))*C(x(i),y(j))/dt;
    b(n*m*(l-1)+m*(i-1)+j)=F(x(i),y(j));
    end
end
end

% Решение СЛАУ

u=b/a';

% Преобразование вектора-строки значений искомой функции
% в узлах координатной сетки в матрицу размерностью n x m,
% удобную для представления результатов
% в графическом виде

for l=1:s
    for i=1:n
        for j=1:m
            T(i,j,l)=u(n*m*(l-1)+m*(i-1)+j);
        end
    end
end

% Построение графиков искомой функции T(x,y,t)

```

```
for l=1:s
    figure
    surf(y,x,T(:,:,l))
    xlabel('y, м')
    ylabel('x, м')
    zlabel('T, К')
    grid on
    colormap('cool')
    axis([min(y) max(y) min(x) max(x) ...
          min(min(T(:,:,1))) max(max(T(:,:,1)))])
    pause(0.1)
    M(l)=getframe;
end

figure
movie(M,10,3)
```

Приведенное описание необходимо сохранить в виде текстового файла с именем `termo_2d.m` и поместить его в каталог `/WORK`, находящийся в корневом каталоге системы MATLAB.

Вызов функции `termo_2d` может осуществляться следующими командами:

```
termo_2d;
x=termo_2d;
[x,y]=termo_2d;
[x,y,t]=termo_2d;
[x,y,t,T]=termo_2d(t0,ts,s,x0,xn,n,y0,ym,m,r,c,k,f, ...
                   vt,gt1,v1,g1,v2,g2,v3,g3,v4,g4) .
```

При использовании первой, второй, третьей или четвертой команд функция будет выводить графически решение задачи при входных параметрах, принятых по умолчанию.

Графики распределений температуры по координатам в различные моменты времени будут выводиться в отдельных окнах. После вывода всех графиков в новом окне будет показано изменение температуры во всех точках координатной сетки в динамическом режиме (`movie`).

Правила вывода информации при вызове функции `termo_2d` аналогичны соответствующим правилам для функции `puass_2d`.

Например, при вызове функции командой

```
termo_2d;
```

на экране появится график решения задачи для следующих значений параметров, заданных в данной функции по умолчанию:

- начальный момент времени – 0 с;
- конечный момент времени – 6 с;
- число точек сетки по оси времени – 6;
- $x_{min} = 0$;
- $x_{max} = 5\text{см}$;
- число точек сетки по оси x – 10;
- $y_{min} = 0$;
- $y_{max} = 8\text{см}$;
- число точек сетки по оси y – 20;
- плотность вещества – $2\,000\text{ кг/м}^3$;
- удельная теплоемкость вещества – $100\text{ Дж}/(\text{кг} \cdot \text{K})$;
- коэффициент теплопроводности вещества – $20\text{ Вт}/(\text{м} \cdot \text{K})$;
- плотность мощности источников (стоков) тепла – 0 Вт/м^3 ,

с начальными условиями первого рода

$$T(x, y, t_1) = 10[\text{sign}(100x - 2) - \text{sign}(100x - 3) + \text{sign}(100y - 3) - \text{sign}(100y - 5)] + 300 \quad (\text{A.79})$$

и граничными условиями Дирихле

$$T(x_{min}, y, t) = 300\text{ K}; \quad (\text{A.80})$$

$$T(x_{max}, y, t) = 300\text{ K}; \quad (\text{A.81})$$

$$T(x, y_{min}, t) = 300\text{ K}; \quad (\text{A.82})$$

$$T(x, y_{max}, t) = 300\text{ K}. \quad (\text{A.83})$$

на равномерной пространственно-временной сетке, содержащей 1 200 точек (размером $10 \times 20 \times 6$) (рис. A.19).

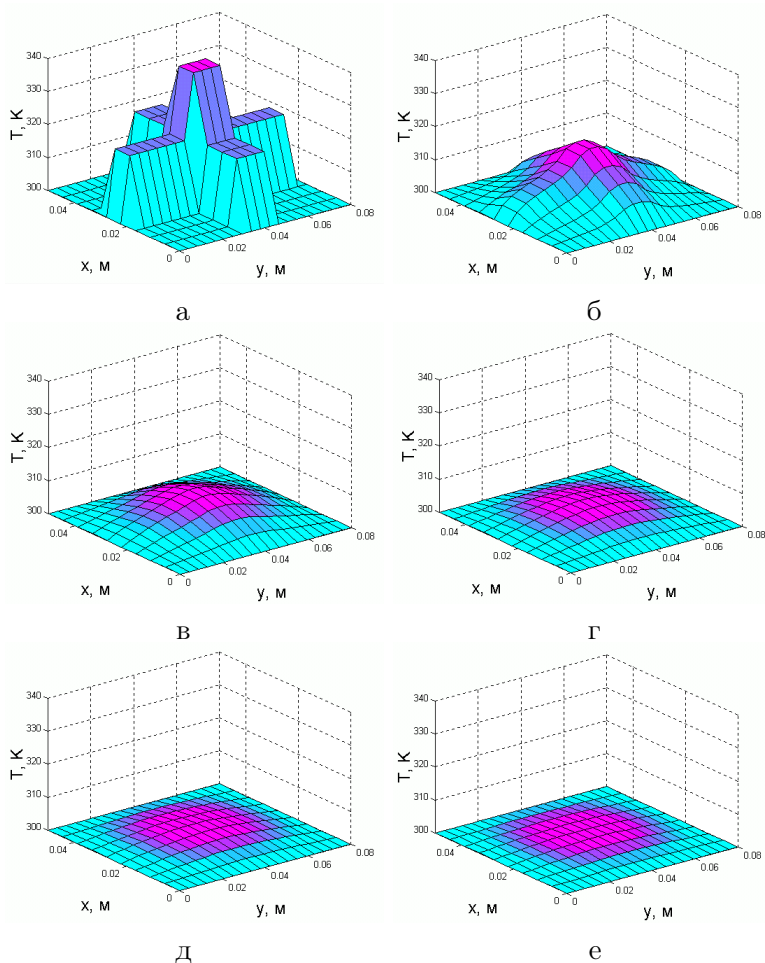


Рис. A.19. Распределение температуры по координатам в различные моменты времени: а) $t = 0$; б) $t = 1 c$; в) $t = 2 c$; г) $t = 3 c$; д) $t = 4 c$; е) $t = 5 c$

Пусть необходимо решить задачу о нестационарном распределении температуры в медном стержне прямоугольного профиля длиной 20 см, шириной 1 см (плотность 8900 кг/м^3 , удельная теплоем-

кость $380 \text{ Дж}/(\text{кг}\cdot\text{K})$, коэффициент теплопроводности $385 \text{ Вт}/(\text{м}\cdot\text{K})$ [17]) на промежутке времени 10 с при условии отсутствия источников и стоков тепла в объеме стержня, если температура на концах стержня равна 300 K и тепловой поток через боковые границы равен нулю для начального распределения температуры, показанного на рис. A.19, а.

Для решения данной задачи с помощью функции `termo_2d` следует использовать следующую командную строку:

```
[x,y,t,T]=termo_2d(0,10,6,0,1e-2,6,0,20e-2,50,'8900',...  
'380','385','0',1,'20*(sign(1e2*y-5)-sign(1e2*y-7))+...  
sign(1e2*y-12)-sign(1e2*y-14))+300',2,'0',2,'0',1,...  
'300',1,'300')
```

или изменить в исходном файле `termo_2d.m` значения соответствующих параметров по умолчанию.

В результате дискретизации задачи на пространственно-временной сетке, содержащей 1800 узлов (размером $6 \times 50 \times 6$) и решения СЛАУ, получим результаты, представленные на рис. A.20.

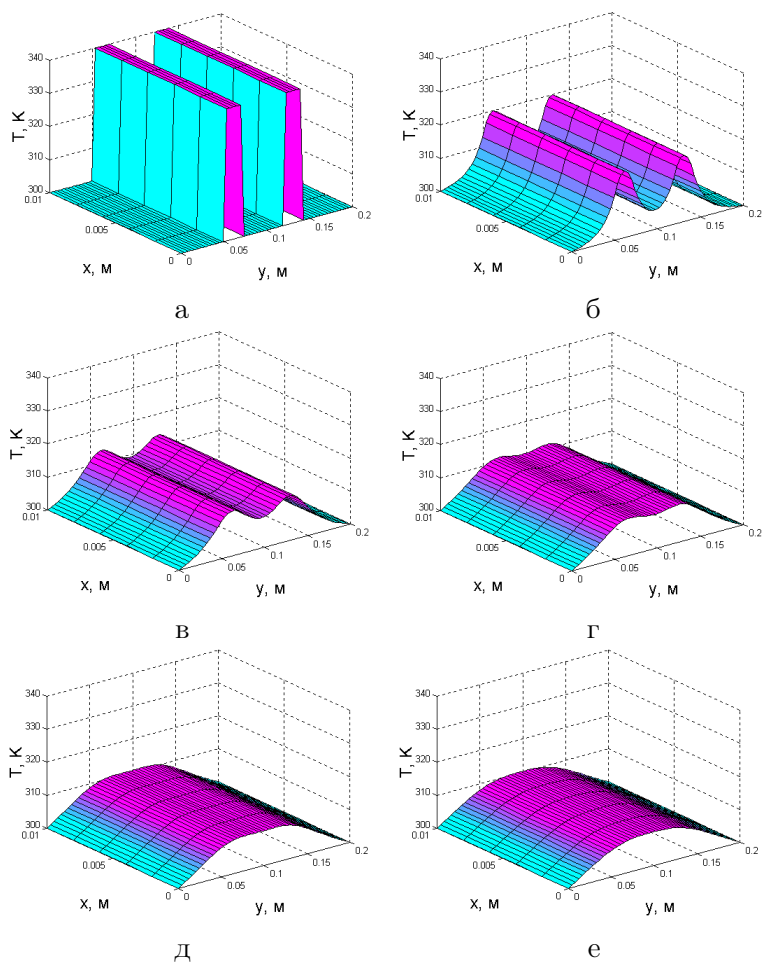


Рис. А.20. Распределение температуры в медном стержне в различные моменты времени: а) $t = 0$; б) $t = 2c$; в) $t = 4c$; г) $t = 6c$; д) $t = 8c$; е) $t = 10c$

А.3. Примеры решения волнового уравнения

В качестве примера решения уравнений гиперболического типа рассмотрим волновое уравнение:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}, \quad (\text{A.84})$$

где t – время; x, y – координаты; $u(x, y, t)$ – искомая функция координат и времени на прямоугольной области с граничными условиями Дирихле или Неймана на границах $x = x_{\min}$, $x = x_{\max}$, $y = y_{\min}$, $y = y_{\max}$ и начальными условиями первого или второго рода на отрезке времени $[t_{\min}, t_{\max}]$.

Зададим на отрезке $[x_{\min}, x_{\max}]$ равномерную координатную сетку с шагом Δx :

$$\mathbf{x} = \{x_i | i = 1, 2, \dots, n\}, \quad (\text{A.85})$$

на отрезке $[y_{\min}, y_{\max}]$ – равномерную координатную сетку с шагом Δy :

$$\mathbf{y} = \{y_j | j = 1, 2, \dots, m\}, \quad (\text{A.86})$$

на отрезке $[t_{\min}, t_{\max}]$ – равномерную сетку с шагом Δt :

$$\mathbf{t} = \{t_l | l = 1, 2, \dots, s\}. \quad (\text{A.87})$$

Векторы, заданные выражениями (A.85) – (A.87), определяют на прямоугольной области равномерную пространственно-временную сетку:

$$G = \{x_i, y_j, t_l | i = 1, 2, \dots, n, j = 1, 2, \dots, m, l = 1, 2, \dots, s\}. \quad (\text{A.88})$$

Граничные условия первого рода (Дирихле) для рассматриваемой задачи могут быть представлены в виде

$$u(x_1, y, t) = g_1(y); \quad (\text{A.89})$$

$$u(x_n, y, t) = g_2(y); \quad (\text{A.90})$$

$$u(x, y_1, t) = g_3(y); \quad (\text{A.91})$$

$$u(x, y_m, t) = g_4(y), \quad (\text{A.92})$$

где x_1, x_n – координаты граничных точек области x_{\min}, x_{\max} ; y_1, y_m – координаты граничных точек области y_{\min}, y_{\max} ; $g_1(y), g_2(y)$,

$g_3(x)$, $g_4(x)$ – некоторые непрерывные функции соответствующих координат.

Граничные условия второго рода (Неймана) для рассматриваемой задачи могут быть представлены в виде

$$\left. \frac{\partial u}{\partial x} \right|_{x_1, y, t} = g_1(y); \quad (\text{A.93})$$

$$\left. \frac{\partial u}{\partial x} \right|_{x_n, y, t} = g_2(y); \quad (\text{A.94})$$

$$\left. \frac{\partial u}{\partial y} \right|_{x, y_1, t} = g_3(x); \quad (\text{A.95})$$

$$\left. \frac{\partial u}{\partial y} \right|_{x, y_m, t} = g_4(x). \quad (\text{A.96})$$

$$(\text{A.97})$$

Начальные условия первого рода для рассматриваемой задачи могут быть представлены в виде

$$u(x, y, t_1) = g_{t1}(x, y); u(x, y, t_s) = g_{ts}(x, y), \quad (\text{A.98})$$

где t_1 – начальный момент времени; t_s – конечный момент времени; $g_{t1}(x, y)$, $g_{ts}(x, y)$ – некоторые непрерывные функции соответствующих координат.

Начальные условия второго рода для рассматриваемой задачи могут быть представлены в виде

$$\left. \frac{\partial u}{\partial t} \right|_{x, y, t_1} = g_{t1}(x, y); \quad (\text{A.99})$$

$$\left. \frac{\partial u}{\partial t} \right|_{x, y, t_s} = g_{ts}(x, y). \quad (\text{A.100})$$

Проводя дискретизацию граничных условий Дирихле на равномерной сетке (A.88) с использованием метода конечных разностей, получим

$$u_{1,j,l} = g_1(y_j); \quad (\text{A.101})$$

$$u_{n,j,l} = g_2(y_j); \quad (\text{A.102})$$

$$u_{i,1,l} = g_3(x_i); \quad (\text{A.103})$$

$$u_{i,m,l} = g_4(x_i), \quad (\text{A.104})$$

где $u_{1,j,l}$, $u_{n,j,l}$, $u_{i,1,l}$, $u_{i,m,l}$ – значения функции $u(x, y, t)$ в точках (x_1, y_j, t_l) , (x_n, y_j, t_l) , (x_i, y_1, t_l) , (x_i, y_m, t_l) соответственно.

Проводя дискретизацию граничных условий Неймана на сетке (A.88), получим

$$\frac{u_{2,j,l} - u_{1,j,l}}{\Delta x} = g_1(y_j); \quad (\text{A.105})$$

$$\frac{u_{n,j,l} - u_{n-1,j,l}}{\Delta x} = g_2(y_j); \quad (\text{A.106})$$

$$\frac{u_{i,2,l} - u_{i,1,l}}{\Delta y} = g_3(x_j); \quad (\text{A.107})$$

$$\frac{u_{i,m,l} - u_{i,m-1,l}}{\Delta y} = g_4(x_j). \quad (\text{A.108})$$

Проводя дискретизацию начальных условий первого рода на равномерной сетке (A.88), получим

$$u_{i,j,1} = g_{t1}(x_i, y_j); \quad (\text{A.109})$$

$$u_{i,j,s} = g_{ts}(x_i, y_j), \quad (\text{A.110})$$

где $u_{i,j,1}$, $u_{i,j,s}$ – значения функции $u(x, y, t)$ в точках (x_i, y_j, t_1) , (x_i, y_j, t_s) .

Проводя дискретизацию начальных условий второго рода на сетке (A.88), получим

$$\frac{u_{i,j,2} - u_{i,j,1}}{\Delta t} = g_{t1}(x_i, y_j); \quad (\text{A.111})$$

$$\frac{u_{i,j,s} - u_{i,j,s-1}}{\Delta t} = g_{ts}(x_i, y_j). \quad (\text{A.112})$$

Проводя дискретизацию уравнения (A.84) для внутренних точек сетки, получим

$$\begin{aligned} & \frac{u_{i,j,l+1} - 2u_{i,j,l} + u_{i,j,l-1}}{\Delta t^2} - \frac{u_{i+1,j,l} - 2u_{i,j,l} + u_{i-1,j,l}}{\Delta x^2} - \\ & - \frac{u_{i,j+1,l} - 2u_{i,j,l} + u_{i,j-1,l}}{\Delta y^2} = 0 \\ & i = 2, \dots, n-1; \quad j = 2, \dots, m-1; \quad l = 2, \dots, s. \end{aligned} \quad (\text{A.113})$$

Таким образом, в результате дискретизации получим систему линейных алгебраических уравнений размерностью $n \times m \times s$.

Ниже приведен один из вариантов функции с комментариями для численного решения уравнения (A.84) с граничными условиями (A.89) – (A.96) и начальными условиями (A.98) или (A.100) на равномерной сетке (A.88).

```
% Функция решения волнового уравнения
%  $d^2U/dt^2 = d^2U/dx^2 + d^2U/dy^2$ 
% на прямоугольной области с граничными условиями
% Дирихле и/или Неймана

function[x,y,t,U]= ...
wave_2d(t0,ts,s,x0,xn,n,y0,ym,m, ...
vt1,gt1,vt2,gt2,v1,g1,v2,g2,v3,g3,v4,g4)

% Входные параметры:
% t0 - начальный момент времени;
% ts - конечный момент времени;
% x0 - начальная координата области решения по оси x;
% xn - конечная координата области решения по оси x;
% y0 - начальная координата области решения по оси y;
% ym - конечная координата области решения по оси y;
% n - число точек координатной сетки вдоль оси x;
% m - число точек координатной сетки вдоль оси y;
% s - число точек сетки вдоль оси времени t;
% vt1- параметр, значение которого определяет
%      тип начального условия в момент времени t(1)
%      (1 - Дирихле, 2 - Неймана);
% gt1- функция в правой части начального условия
%      в момент времени t(1),
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки;
% vt2- параметр, значение которого определяет
%      тип начального условия в момент времени t(s)
%      (1 - Дирихле, 2 - Неймана);
% gt2- функция в правой части начального условия
%      в момент времени t(s),
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки;
```

```
% v1 - параметр, значение которого определяет
%      тип граничного условия на первой границе
%      области  $x = x(1)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);
% g1 - функция в правой части граничного условия
%      на первой границе,
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки;
% v2 - параметр, значение которого определяет
%      тип граничного условия на второй границе
%      области  $x = x(n)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);
% g2 - функция в правой части граничного условия
%      на второй границе,
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки;
% v3 - параметр, значение которого определяет
%      тип граничного условия на третьей границе
%      области  $y = y(1)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);
% g3 - функция в правой части граничного условия
%      на третьей границе,
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки;
% v4 - параметр, значение которого определяет
%      тип граничного условия на четвертой границе
%      области  $y = y(m)$  (1 - ГУ Дирихле, 2 - ГУ Неймана);
% g4 - функция в правой части граничного условия
%      на четвертой границе,
%      задаваемая строкой символов, заключенных
%      в одинарные кавычки.

% Выходные параметры:
% x - вектор-строка координатной сетки по оси x
%      размерностью 1 x n;
% y - вектор-строка координатной сетки по оси y
%      размерностью 1 x m;
% t - вектор-строка сетки по оси времени
%      размерностью 1 x s;
```

% U - матрица значений результирующей функции
% в узлах координатной сетки размерностью n x m x s.

% Функции и переменные по умолчанию

```
if exist('t0')==0
    t0=0;
end
if exist('ts')==0
    ts=0.2;
end
if exist('s')==0
    s=6;
end
if exist('x0')==0
    x0=-1;
end
if exist('xn')==0
    xn=1;
end
if exist('n')==0
    n=18;
end
if exist('y0')==0
    y0=-1;
end
if exist('ym')==0
    ym=1;
end
if exist('m')==0
    m=18;
end
if exist('vt1')==0
    vt1=1;
end
if exist('gt1')==0
    gt1='sin(4*x)-sin(4*y)';
```

```
end
if exist('vt2')==0
vt2=1;
end
if exist('gt2')==0
gt2='sin(4*y)-sin(4*x)';
end
if exist('v1')==0
v1=2;
end
if exist('g1')==0
g1='0';
end
if exist('v2')==0
v2=2;
end
if exist('g2')==0
g2='0';
end
if exist('v3')==0
v3=2;
end
if exist('g3')==0
g3='0';
end
if exist('v4')==0
v4=2;
end
if exist('g4')==0
g4='0';
end
```

% Задание равномерной сетки

```
x=x0:(xn-x0)/(n-1):xn; dx=x(2)-x(1);
y=y0:(ym-y0)/(m-1):ym; dy=y(2)-y(1);
t=t0:(ts-t0)/(s-1):ts; dt=t(2)-t(1);
```

```
% Вычисление значений функций, заданных символьно,  
% в узлах координатной сетки  
  
GT1=inline(gt1,'x','y');  
GT2=inline(gt2,'x','y');  
G1=inline(g1,'y');  
G2=inline(g2,'y');  
G3=inline(g3,'x');  
G4=inline(g4,'x');  
  
% Определение размерности СЛАУ  
  
N=s*n*m;  
  
% Задание матрицы коэффициентов СЛАУ размерностью N x N,  
% все элементы которой равны 0  
  
a=zeros(N,N);  
  
% Задание матрицы-строки свободных членов СЛАУ размерностью 1xN,  
% все элементы которой равны 0  
  
b=zeros(1,N);  
  
% Определение коэффициентов и свободных членов СЛАУ,  
% соответствующих начальным и граничным условиям,  
% и проверка корректности  
% значений параметров vt1, vt2, v1, v2, v3, v4  
  
for i=1:n  
    for j=1:m  
        b(m*(i-1)+j)=GT1(x(i),y(j));  
        if vt1==1  
            a(m*(i-1)+j,m*(i-1)+j)=1;  
        elseif vt1==2  
            a(m*(i-1)+j,m*(i-1)+j)=-1/dt;
```

```
        a(m*(i-1)+j,n*m+m*(i-1)+j)=1/dt;
    else
        error('Parameter vt1 have incorrect value');
    end
    b(n*m*(s-1)+m*(i-1)+j)=GT2(x(i),y(j));
    if vt2==1
        a(n*m*(s-1)+m*(i-1)+j,n*m*(s-1)+m*(i-1)+j)=1;
    elseif vt2==2
        a(n*m*(s-1)+m*(i-1)+j,n*m*(s-1)+m*(i-1)+j)=1/dt;
        a(n*m*(s-1)+m*(i-1)+j,n*m*(s-2)+m*(i-1)+j)= ...
            -1/dt;
    else
        error('Parameter vt2 have incorrect value');
    end
end
end
for l=1:s
    for j=1:m
        b(n*m*(l-1)+j)=G1(y(j));
        if v1==1
            a(n*m*(l-1)+j,n*m*(l-1)+j)=1;
        elseif v1==2
            a(n*m*(l-1)+j,n*m*(l-1)+j)=-1/dx;
            a(n*m*(l-1)+j,n*m*(l-1)+m+j)=1/dx;
        else
            error('Parameter v1 have incorrect value');
        end
        b(n*m*(l-1)+m*(n-1)+j)=G2(y(j));
        if v2==1
            a(n*m*(l-1)+m*(n-1)+j,n*m*(l-1)+m*(n-1)+j)=1;
        elseif v2==2
            a(n*m*(l-1)+m*(n-1)+j,n*m*(l-1)+m*(n-1)+j)=1/dx;
            a(n*m*(l-1)+m*(n-1)+j,n*m*(l-1)+m*(n-2)+j)= ...
                -1/dx;
        else
            error('Parameter v2 have incorrect value');
        end
    end
end
```

```
end
for i=2:n-1
    b(n*m*(l-1)+m*(i-1)+1)=G3(x(i));
    if v3==1
        a(n*m*(l-1)+m*(i-1)+1,n*m*(l-1)+m*(i-1)+1)=1;
    elseif v3==2
        a(n*m*(l-1)+m*(i-1)+1,n*m*(l-1)+m*(i-1)+1)= ...
            -1/dy;
        a(n*m*(l-1)+m*(i-1)+1,n*m*(l-1)+m*(i-1)+2)=1/dy;
    else
        error('Parameter v3 have incorrect value');
    end
    b(n*m*(l-1)+m*(i-1)+m)=G4(x(i));
    if v4==1
        a(n*m*(l-1)+m*(i-1)+m,n*m*(l-1)+m*(i-1)+m)=1;
    elseif v4==2
        a(n*m*(l-1)+m*(i-1)+m,n*m*(l-1)+m*(i-1)+m)=1/dy;
        a(n*m*(l-1)+m*(i-1)+m,n*m*(l-1)+m*(i-1)+m-1)=...
            -1/dy;
    else
        error('Parameter v4 have incorrect value');
    end
end
end
end
```

% Определение коэффициентов и свободных членов СЛАУ,
% соответствующих внутренним точкам области

```
for l=2:s-1
    for i=2:n-1
        for j=2:m-1
            a(n*m*(l-1)+m*(i-1)+j,n*m*(l-1)+m*(i-1)+j)=...
                -2/dt^2+2/dx^2+2/dy^2;
            a(n*m*(l-1)+m*(i-1)+j,n*m*(l-1)+m*i+j)=-1/dx^2;
            a(n*m*(l-1)+m*(i-1)+j,n*m*(l-1)+m*(i-2)+j)=...
                -1/dx^2;
            a(n*m*(l-1)+m*(i-1)+j,n*m*(l-1)+m*(i-1)+j+1)=...
```



```
-1/dy^2;
a(n*m*(l-1)+m*(i-1)+j,n*m*(l-1)+m*(i-1)+j-1)=...
-1/dy^2;
a(n*m*(l-1)+m*(i-1)+j,n*m*l+m*(i-1)+j)=1/dt^2;
a(n*m*(l-1)+m*(i-1)+j,n*m*(l-2)+m*(i-1)+j)=...
1/dt^2;
end
end
end

% Решение СЛАУ

u=b/a';

% Преобразование вектора-строки значений искомой функции
% в узлах координатной сетки в матрицу
% размерностью n x m x s, удобную для представления
% результатов в графическом виде

for l=1:s
    for i=1:n
        for j=1:m
            U(i,j,l)=u(n*m*(l-1)+m*(i-1)+j);
        end
    end
end

% Построение графика искомой функции U(x,y,t)

for l=1:s
    figure
    surf(y,x,U(:, :, l))
    xlabel('y')
    ylabel('x')
    zlabel('U')
    grid on
    colormap('cool')
```

```

axis([min(y) max(y) min(x) max(x) min(min(min(U))) ...
      max(max(max(U)))])
pause(0.1)
M(l)=getframe;
end

```

% Отображение волнового процесса в динамическом режиме

```

figure
movie(M,20,10)

```

Приведенное описание необходимо сохранить в виде текстового файла с именем `wave_2d.m` и поместить его в каталог `/WORK`, находящийся в корневом каталоге системы MATLAB.

Вызов функции `wave_2d` осуществляется аналогично функциям `puass_2d` и `termo_2d` следующими командами:

```

wave_2d;
x=wave_2d;
[x,y]=wave_2d;
[x,y,t]=wave_2d;
[x,y,t,U]=wave_2d(t0,ts,s,x0,xn,n,y0,ym,m,...
    vt1,gt1,vt2,gt2,v1,g1,v2,g2,v3,g3,v4,g4) .

```

Графики распределений искомой функции по координатам в различные моменты времени будут выводиться в отдельных окнах. После вывода всех графиков в новом окне будет показан волновой процесс в динамическом режиме.

При использовании первой, второй, третьей или четвертой команд функция будет выводить графически решение задачи при входных параметрах, принятых по умолчанию:

- начальный момент времени – 0;
- конечный момент времени – 0,2;
- число точек сетки по оси времени – 6;
- $x_{min} = -1$;
- $x_{max} = 1$;

- число точек сетки по оси x – 18;
- $y_{min} = -1$;
- $y_{max} = 1$;
- число точек сетки по оси y – 18.

При этом заданы начальные условия первого рода

$$u(x, y, t_1) = \sin(4x) - \sin(4y); \quad (\text{A.114})$$

$$u(x, y, t_1) = \sin(4y) - \sin(4x); \quad (\text{A.115})$$

и граничные условия Неймана

$$\left. \frac{\partial u}{\partial x} \right|_{x_{min}, y, t} = 0; \quad (\text{A.116})$$

$$\left. \frac{\partial u}{\partial x} \right|_{x_{max}, y, t} = 0; \quad (\text{A.117})$$

$$\left. \frac{\partial u}{\partial x} \right|_{x, y_{min}, t} = 0; \quad (\text{A.118})$$

$$\left. \frac{\partial u}{\partial x} \right|_{x, y_{max}, t} = 0 \quad (\text{A.119})$$

на равномерной пространственно-временной сетке, содержащей 1 944 узла (размером $18 \times 18 \times 6$) (рис. A.21).

Функция `wave_2d` выводит результаты в нормированных единицах. При умножении на соответствующие коэффициенты нормировки можно получить графическое отображение волнового процесса изменения конкретных физических величин на прямоугольной области с размерами, выраженными в удобных единицах измерения, в моменты времени, выраженные в секундах.

Для этого необходимо осуществить вызов функции `wave_2d` при помощи одной из следующих команд:

```
[x,y,t,U]=wave_2d;
```

```
[x,y,t,U]=wave_2d(t0,ts,s,x0,xn,n,y0,ym,m,...
```

```
vt1,gt1,vt2,gt2,v1,g1,v2,g2,v3,g3,v4,g4) .
```

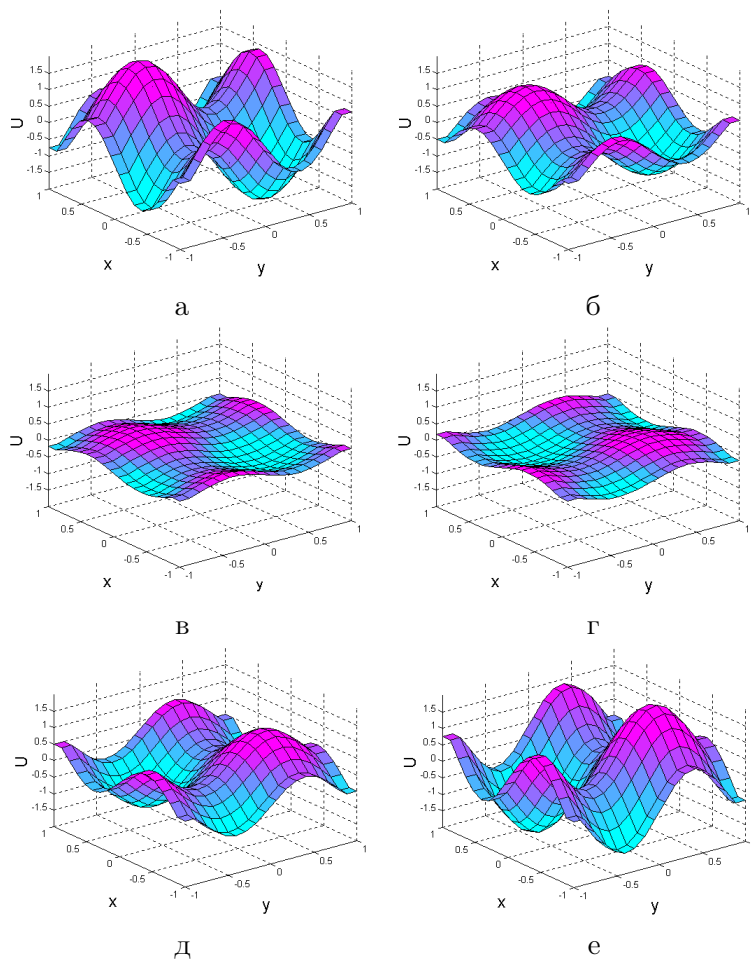


Рис. А.21. Волновой процесс $u(x, y, t)$ в различные моменты времени: а) $t = 0$; б) $t = 0,04$; в) $t = 0,08$; г) $t = 0,12$; д) $t = 0,16$; е) $t = 0,2$

В этом случае при выходе из функции матрицы точек пространственно-временной сетки \mathbf{x} , \mathbf{y} , \mathbf{t} и трехмерная матрица \mathbf{U} результирующих значений функции $u(x, y, t)$ будут сохранены в оперативной памяти и доступны для операций умножения на коэффициенты нормировки. При этом в первом случае будут использованы вход-

ные параметры по умолчанию.

Приложение Б

Справочник MatLab

Б.1. Введение

Б.1.1. Алфавит языка программирования

В MATLAB, как и в других системах, используются все буквы латинского алфавита от A до Z и арабские цифры от 0 до 9. Как и в C++, большие и малые буквы это разные переменные и константы. Кроме букв латинского алфавита используются все специальные символы клавиатуры компьютера.

Б.1.2. Арифметические и логические операторы

Число арифметических операторов в MATLAB включает в себя матричные и арифметические операции. В таблице 2.1 приводится список арифметических операторов.

В математических выражениях операторы имеют определенный приоритет исполнения. В MATLAB приоритет логических операторов выше, чем арифметических, приоритет возведения в степень выше приоритетов умножения и деления, приоритет умножения и деления выше сложения и вычитания. Для повышения приоритета операций нужно использовать круглые скобки. Степень вложения скобок не ограничивается.

Операторы отношения служат для сравнения двух величин, векторов или матриц, все операторы отношения имеют две сравниваемые величины и записываются, как показано в таблице 2.2.

Таблица 2.1. Арифметические операторы

Функция	Обозначение (синтаксис)
Сложение	$+$ (M1+M2)
Вычитание	$-$ (M1-M2)
Матричное умножение	$*$ (M1*M2)
Поэлементное умножение	$.*$ (M1.*M2)
Деление матриц слева направо	$/$ (M1/M2)
Поэлементное деление массивов слева направо	$./$ (M1./M2)
Деление матриц справа налево	\backslash (M1\M2)
Поэлементное деление	$.\backslash$ (M1.\M2)

Таблица 2.2. Арифметические операторы

Функция	Обозначение (синтаксис)
Равно	$=$ (x = y)
Не равно	\sim (x ~ y)
Меньше	$<$ (x < y)
Больше	$>$ (x > y)
Меньше или равно	$<=$ (x <= y)
Больше или равно	$>=$ (x >= y)

Таблица 2.3. Арифметические операторы

Функция	Синтаксис
$ x $ – модуль	<code>abs(x)</code>
e^x – экспонента	<code>exp(x)</code>
$\ln x$ – натуральный логарифм	<code>log(x)</code>
$\log_2 x$ – логарифм по основанию 2	<code>log2(x)</code>
$\log_{10} x$ – десятичный логарифм	<code>log10(x)</code>
2^x – 2 в степени	<code>pow(x)</code>
\sqrt{x} – квадратный корень	<code>sqrt(x)</code>
$\cos x$ – косинус	<code>cos(x)</code>
$\sin x$ – синус	<code>sin(x)</code>
$\tan x$ – тангенс	<code>tan(x)</code>

Б.1.3. Элементарные функции

Набор элементарных функций и их описание приведен в таблице 2.3.

В тригонометрических функциях *углы измеряются в радианах*. Следует помнить, что все элементарные функции должны записываться в программах малыми буквами.

Б.1.4. Понятие о файлах-сценариях и файлах-функциях

При загрузке системы MATLAB на мониторе появляется основное окно системы, в котором можно выделить окно команд (Command Window). Система готова к проведению вычислений и созданию программ в командном режиме. Для этого можно на языке MATLAB

записывать программы. Операторы заканчиваются символом ; - точка с запятой. Одновременно точка с запятой блокирует вывод численного значения результата этого оператора в окне команд. В одной строке можно записать несколько операторов, а сами строки автоматически нумеруются при нажатии клавиши Enter. Если программа полностью записана и выходные величины не имеют символа ;, то после нажатия клавиши Enter она выполняется. Ниже программы появляется ее результат. В таком режиме выполнять решения задач нецелесообразно, т.к. исправить возможные ошибки после нажатия клавиши Enter уже нельзя. Поэтому записывать программы, их редактировать и отлаживать необходимо в так называемых М-файлах. М-файл создается при выполнении команды New меню File. Для ускорения этой команды выведена специальная пиктограмма в виде белой странички с загнутым уголком на панели инструментов. Щелкнув по пиктограмме стрелкой мышки, получаем окно М-файла, на котором можно записывать, редактировать и отлаживать любые программы решения научных и инженерных задач. Данный М-файл по умолчанию имеет название Untitled (Безымянный). Чтобы дать ему имя, необходимо в меню этого окна File выполнить команду Save as и в другом окне указать папку и имя этого файла. После указания имени и сохранения М-файла он готов для выполнения записанной программы. Для этого необходимо щелкнуть мышкой по пиктограмме Выполнить. Она выполнена в виде страницы со стрелкой, направленной вниз. Результат выполнения программы или сообщения об ошибках появится в окне команд. Описанный процесс называется созданием М-файла сценария сессии. Файл-сценарий, именуемый также Script-файлом, имеет весьма простую структуру:

% Комментарии, если необходимо.

Тело программы с любыми выражениями.

Важными являются следующие свойства файлов-сценариев:

1. Они не имеют входных и выходных аргументов.
2. Работают с данными из рабочей области.
3. В процессе выполнения не компилируются.

4. Представляют собой последовательность операций, аналогичную той, что используется в сессии.

Кроме М-файла сценария, в MATLAB существует М-файл функция. Отличие М-файла функции от сценария состоит в том, что он является аналогом подпрограммы типа `function` в языке Pascal.

Структура М-файла функции с одним выходным параметром имеет вид:

```
function var = f_name (Список параметров)
```

Тело программы с любыми выражениями.

```
var = выражение
```

М-файл функция обладает такими свойствами:

1. Он начинается с ключевого слова `function`, после которого указывается имя переменной `var` – выходного параметра, имя самой функции `f_name` и список ее входных параметров, отделенных запятой.
2. Результат выполнения М-файла функции присваивается имени функции, которое может использоваться в математических выражениях подобно функциям `sin(x)`, `log(x)` и т.п.
3. Все переменные, используемые в файле-функции, являются локальными, т.е. действуют только в пределах тела функции.
4. Файл-функция является самостоятельным программным модулем, который связан с другими модулями и головной программой через входные и выходные параметры.
5. При обнаружении файла-функции он компилируется и затем исполняется.

Внимание: Имя М-файла функции должно совпадать с самой `f_name` (именем самой функции).

Б.2. Основы программирования

Б.2.1. Оператор присваивания

Программирование, т.е. создание определенного набора команд, в системе MATLAB является средством ее расширения и использовании в решении специфических проблем. Отдельные вопросы программирования изложены выше, здесь рассмотрим правила, дополняющие синтаксис языка MATLAB.

Программы оперируют с *переменными* и *константами*. Переменные – это имеющие имена объекты, способные хранить разные по значению данные. В зависимости от этих данных переменные могут быть числовыми или символьными, векторными или матричными.

Для задания переменным определенных значений используется *оператор присваивания*, вводимый знаком равенства =

`Имя_переменной = Выражение;`

Типы переменных заранее не декларируются. Они определяются выражением, значение которого присваивается переменной. Имя переменной может содержать сколько угодно символов, но идентифицируется только 31 начальный символ. Имя любой переменной должно быть уникальным. Имя должно начинаться с буквы, может содержать буквы, цифры и символ подчеркивания `_`. Недопустимо включать в имена пробелы и специальные знаки.

Б.2.2. Перенос строки

Если математическое выражение выходит за размер экрана монитора, то целесообразно перенести его часть на следующую строку. Для этого используется символ многоточие `...` – три и более точки. В командном режиме число возможных символов в одной строке – 4096, в М-файле – не ограничено, но с такими длинными строками работать неудобно. Поэтому применение в файлах-сценариях символа переноса строки улучшает наглядность программ.

Б.2.3. Ввод и вывод данных

В языке MATLAB нет явных операторов ввода вывода данных. Эта проблема решается для ввода данных оператором при-

сваивания и использованием системных констант. Вывод данных осуществляется еще проще. Для этого необходимо после математического выражения не ставить символ ; – точку с запятой. К системным константам относятся:

π = 3,1415 – число «ПИ»;

i или j – мнимые единицы;

NaN – неопределенность в виде $\frac{0}{0}$;

Inf – бесконечность типа $\frac{a}{0}$;

ans – результат последней операции и др.

Б.2.4. Форматы чисел

При вычислениях в MATLAB используется режим двойной точности. Однако, при выводе результатов, по умолчанию выдаются числа с 4 цифрами после десятичной точки в действительной форме. Чтобы изменить данную форму вывода, необходимо в программе перед выводимой величиной использовать команду **format name**, где **name** – имя формата. Для числовых данных **name** может быть следующим сообщением:

short – короткое представление в фиксированном формате (5 знаков);

short e – короткое представление в экспоненциальной форме (5 знаков мантиисы и 3 знака порядка);

long – длинное представление в фиксированном формате (15 знаков);

long e – длинное представление в экспоненциальной форме (15 знаков мантиисы и 3 знака порядка).

Б.2.5. Формирование векторов и матриц

Описанные правила вычислений распространяются и на более сложные вычисления, которые при использовании обычных языков программирования (типа Pascal, Fortran, C++ и др.) требуют составления специальных программ. MATLAB специально предназначен для проведения сложных вычислений с векторами и матрицами. При этом по умолчанию предполагается, что каждая переменная – это вектор или матрица. Например, если задано $x = 1$,

то это значит, что x – это вектор с одним элементом, равным 1. Если надо задать вектор из трех элементов, то их значения надо перечислить в квадратных скобках, разделяя пробелами.

```
>>V = [1 2 3]
V =
```

```
1      2      3
```

В данном случае задан вектор-строка. Если разделить элементы точкой с запятой, то получим вектор-столбец.

```
>>V = [1; 2; 3]
V =
```

```
1
2
3
```

Задание матрицы требует указания несколько строк. Для разграничения строк используется символ ; (точка с запятой).

```
>>T = [1 2 3; 4 5 6; 7 8 9]
T =
```

```
1      2      3
4      5      6
7      8      9
```

Для указания отдельного элемента вектора или матрицы используются выражения вида $V(i)$ или $T(i, j)$ \verb. Например:

```
>>T (3,2)
ans =
```

```
8
```

Если элементу $T(i, j)$ нужно присвоить новое значение x , то используют оператор присваивания

```
T (3,2) = x;
```

Выражение $T(i)$ с одним индексом дает доступ к элементам матрицы, развернутым в один столбец. Такая матрица образуется из исходной, если подряд выписать ее столбцы. Например:

```
>>T (3)
```

```
ans =
```

```
7
```

```
>>T (8)
```

```
ans =
```

```
6
```

Наряду с операциями над отдельными элементами матриц и векторов MATLAB позволяет производить арифметические операции сразу над всеми элементами. Для этого перед знаком операции ставится точка.

Б.2.6. Оператор двоеточие :

Весьма часто необходимо выполнить формирование упорядоченных числовых последовательностей. Такие последовательности нужны для создания векторов или значений аргументов x при построении графиков. В MATLAB для этого используется оператор двоеточие `:`, который представляется следующим образом:

```
x = Начальное _ значение : Шаг : Конечное _ значение ;
```

Эта конструкция создает возрастающую последовательность чисел, которая начинается с начального значения, изменяется на заданный шаг и завершается конечным значением. Если шаг не задан, то он принимает значение 1. Если конечное значение указано меньшим, чем начальное значение, – то выдается сообщение об ошибке.

Примеры:

```
>> x = 0 : 5
```

```
x =
```

```

      0      1      2      3      4      5

>> cos(x)

ans =

      1.0000      0.5403     -0.4161     -0.9900     -0.6536      0.2837

>> x = 1 : - 0.2 : 0

x =

      1.0000      0.8000      0.6000      0.4000      0.2000           0

```

Б.2.7. Формирование массивов специального вида

ZEROS, ONES – формирование массива нулей, или единиц.

Синтаксис:

$Y = \text{zeros}(n)$ – формирует массив нулей размера $n \times n$.

$Y = \text{zeros}(m, n)$ – формирует массив нулей размера $m \times n$.

$Y = \text{zeros}(\text{size}(A))$ – формирует массив нулей соразмерный с массивом A .

Примеры:

```

>> a = ones (3, 2)

a =

```

```

      1      1      1
      1      1      1

```

Linspace – формирование линейного массива равноотстоящих узлов.

Синтаксис:

$x = \text{linspace}(x1, x2)$ – формирует линейный массив размера 1×100 , начальным и конечным элементами которого являются точки $x1$ и $x2$.

`x = linspace(x1, x2, n)` – формирует линейный массив размера $1 \times n$, начальным и конечным элементами которого являются точки $x1$ и $x2$.

LOGSPACE – формирование узлов логарифмической сетки.

Синтаксис:

`x = logspace(d1, d2)` – формирует вектор-строку, содержащую 50 равноотстоящих в логарифмическом масштабе точек, которые покрывают диапазон от 10^{d1} до 10^{d2} .

`x = logspace(d1, d2, n)` – формирует вектор-строку, содержащую n равноотстоящих в логарифмическом масштабе точек, которые покрывают диапазон от 10^{d1} до 10^{d2} .

MESHGRID – формирование узлов двумерной и трехмерной сеток.

Синтаксис:

`[X, Y] = meshgrid(x, y)` – формирует массивы X и Y , которые определяют координаты узлов прямоугольника, задаваемого векторами x и y . Этот прямоугольник задает область определения функции от двух переменных, которую можно построить в виде 3D-поверхности.

`[X, Y] = meshgrid(x)` – является сокращенной формой записи функции `[X, Y] = meshgrid(x, x)`.

`[X, Y, Z] = meshgrid(x, y, z)` – формирует массивы X , Y и Z , которые определяют координаты узлов параллелепипеда, задаваемого векторами x , y и z . Этот параллелепипед задает область определения для вычисления функции от трех переменных и построения 3D-параметрических поверхностей.

Б.2.8. Оператор разветвления if

Условный оператор `if` в MATLAB записывается в общем виде так:

```
if (Логическое условие)
    Оператор 1
elseif (Логическое условие)
    Оператор 2
else
    Оператор 3
```


end

Эта конструкция имеет несколько частных вариантов:

```
if (Логическое условие)
    Оператор 1
end
```

```
if (Логическое условие)
    Оператор 1
else
    Оператор 2
end
```

В качестве операторов отношения используются операторы: $=$, $<$, $>$, $<=$, $>=$, $=$. Если логическое условие принимает значение 1 (true – истина), то выполняются соответствующие операторы. Если логическое условие принимает значение 0 (false – ложь), то операторы, следующие за логическим условием, не выполняются. Оператор **end** указывает на конец условного оператора **if**. В понятие **Оператор 1** входят один или несколько операторов. В последнем случае они разделяются символами ,(запятой) или ;(точкой с запятой).

Как и в других алгоритмических языках, оператор **if** позволяет осуществить разветвление процесса вычислений в зависимости от какого-либо условия.

Б.2.9. Операторы циклов

В MATLAB существует 3 типа операторов цикла. С оператором : (двоеточие) мы познакомились в п. [Б.2.6.](#). Следующий оператор **for ... end** используется для организации цикла с фиксированным числом повторений. Он имеет вид:

```
for var = Выражение
    Операторы
end
```

Здесь **var** – счетчик цикла – любая переменная, обычно это *i*, *j*, *k*, *l*, *m* и т. д. Выражение записывается в виде **s : d : e**, где

s – начальное значение счетчика цикла **var**, d – шаг изменения и – конечное значение **var**. Возможна и запись в виде **s : e**, тогда $d = 1$. Список операторов завершается ключевым словом **end**. Оператор **continue** передает управление в следующую итерацию цикла, пропуская операции, которые записаны за ним. Оператор **break** используется для досрочного прерывания цикла. Возможны вложенные циклы.

```
>> for i = 1 : 3
      for j = 1 : 3
          a(i, j) = i * j ;
      end
  end
```

В результате выполнения этого цикла формируется матрица a

```
>> a
```

```
a =
```

1	2	3
2	4	6
3	6	9

Циклы типа **while ... end** выполняются до тех пор, пока выполняется заданное условие. Оператор записывается в виде:

```
while (Логическое условие)
    Операторы
end
```

Б.3. Работа с массивами

Б.3.1. Решение систем линейных алгебраических уравнений

К решению систем линейных уравнений сводятся многочисленные практические задачи, например различные краевые задачи для обыкновенных и в частных производных дифференциальных уравнений. Можно с полным основанием утверждать, что данная проблема является одной из самых распространенных и важных задач вычислительной математики.

Пусть задана система n линейных алгебраических уравнений с n неизвестными:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2; \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n. \end{cases} \quad (\text{Б.1})$$

Система уравнений (Б.1) в матричной форме представляется следующим образом:

$$\mathbf{Ax} = \mathbf{B}, \quad (\text{Б.2})$$

где \mathbf{A} – матрица коэффициентов СЛАУ размером $n \times n$; \mathbf{x} – вектор-столбец неизвестных; \mathbf{B} – вектор-столбец свободных членов.

Систему уравнений (Б.2) можно решить различными методами. Один из наиболее простых и эффективных методов является метод исключения Гаусса и его модификации. В MATLAB имеется обширный арсенал методов решения систем уравнений (Б.2) методом исключения Гаусса. Для этого применяются следующие операторы

/ – правое деление;

\ – левое деление;

~ -1 – возведение в степень -1 ;

inv(A) – обращение матрицы A .

Выражения

$$\mathbf{X} = \mathbf{B}/\mathbf{A}$$

$$\mathbf{X} = \mathbf{B} * \mathbf{A}^{-1}$$

```
X = B*inv(A)
```

```
X = A\B
```

дают решения системы линейных уравнений $\mathbf{Ax} = \mathbf{B}$, где \mathbf{A} – матрица размером $n \times n$, \mathbf{B} – матрица размером $n \times 1$.

Б.3.2. Основные операции над массивами

MAX – определение максимальных элементов массива.

Синтаксис:

$\mathbf{Y} = \max(\mathbf{X})$ – в случае одномерного массива возвращает наибольший элемент; в случае двумерного массива – это вектор-строка, содержащая максимальные элементы каждого столбца. Таким образом, $\max(\max(\mathbf{X}))$ – это наибольший элемент массива.

$[\mathbf{Y}, \mathbf{I}] = \max(\mathbf{X})$ – кроме самих максимальных элементов возвращает вектор-строку индексов этих элементов в данном столбце.

$\mathbf{C} = \max(\mathbf{A}, \mathbf{B})$ возвращает массив \mathbf{C} тех же размеров, какие имеют массивы \mathbf{A} и \mathbf{B} , каждый элемент которого есть максимальный из соответствующих элементов этих массивов.

Если анализируемый массив содержит комплексные элементы, то максимальные элементы определяются из условия $\max(\text{abs}(\mathbf{X}))$. Если массив содержит один или несколько элементов типа NaN, то результатом операции будет NaN.

Пример:

```
>> M = magic(3)
```

```
M =
```

8	1	6
3	5	7
4	9	2

```
>> y = max(M)
```

```
y =
```

```
      8      9      7
```

```
>> [y, I] = max(M)
```

```
y =
```

```
      8      9      7
```

```
I =
```

```
      1      3      2
```

```
>> max(max(M))
```

```
ans =
```

```
      9
```

MIN – определение минимальных элементов массива.

$Y = \min(X)$ в случае одномерного массива возвращает наименьший элемент; в случае двумерного массива – это вектор-строка, содержащая минимальные элементы каждого столбца. Таким образом, $\min(\min(X))$ – это наименьший элемент массива.

$[Y, I] = \min(X)$ кроме самих минимальных элементов возвращает вектор-строку индексов этих элементов в данном столбце.

$C = \min(A, B)$ возвращает массив **C** тех же размеров, какие имеют массивы **A** и **B**, каждый элемент которого есть минимальный из соответствующих элементов этих массивов.

Если анализируемый массив содержит комплексные элементы, то минимальные элементы определяются из условия $\min(\text{abs}(X))$. Если массив содержит один или несколько элементов типа NaN, то результатом операции \min будет NaN.

Пример:

```
>> M = magic(3)
```

```
M =
```

```
      8      1      6
      3      5      7
      4      9      2
```

```
>> y = min(M)
```

```
y =
```

```
      3      1      2
```

```
>> [y, I] = min(M)
```

```
y =
```

```
      3      1      2
```

```
I =
```

```
      2      1      3
```

```
>> min(min(M))
```

```
ans =
```

```
      1
```

GRADIENT – конечные разности и приближенное вычисление градиента функции от двух переменных.

`[px, py] = gradient(F)\verb` – вычисляет конечные разности функции $F(x, y)$, заданной на двумерной сетке и представляющей собой массив чисел.

`[px, py] = gradient(F, dx, dy)` – возвращает численные зна-

чения производных функции F в виде массивов $px = dF/dx$ и $py = dF/dy$, где dx и dy могут быть скалярами, равными шагам разбиения сетки по осям x и y , либо векторами координат узлов сетки при разбиении с переменным шагом.

Пример: Рассмотрим расчет и построение поля направлений для функции $F(x, y) = x \exp(-x^2 - y^2)$ с использованием функции `gradient`. Результаты представлены на рис. Б.1

```
[x, y] = meshgrid(-2:.2:2, -2:.2:2);  
  
z = x .* exp(-x.^2 - y.^2);  
  
[px, py] = gradient(z, .2, .2);  
  
contour(z), hold on, quiver(px, py), hold off
```

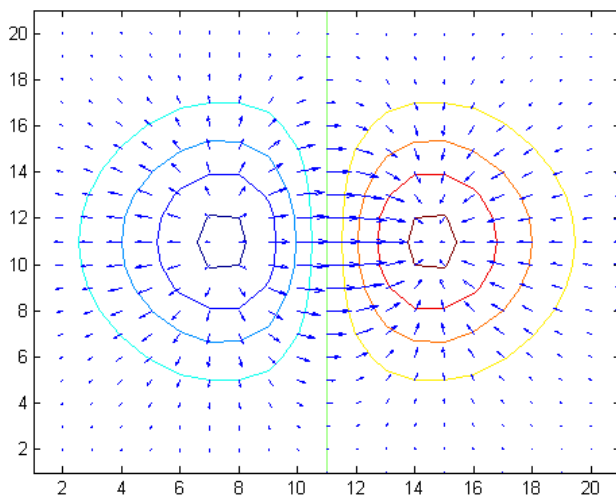


Рис. Б.1. Поле направлений для функции $F(x, y) = x \exp(-x^2 - y^2)$

Б.4. Графика

Элементарные графические функции системы MATLAB позволяют построить на экране и вывести на печатающее устройство следующие типы графиков: линейный, логарифмический, полулогарифмический, полярный.

Для каждого графика можно задать заголовок, нанести обозначение осей и масштабную сетку.

Б.4.1. Двумерные графики

PLOT – график в линейном масштабе.

```
plot(y)
plot(x, y)
plot(x, y, s)
plot(x1, y1, s1, x2, y2, s2, ...)
```

Команда `plot(y)` строит график элементов одномерного массива **y** в зависимости от номера элемента; если элементы массива **y** комплексные, то строится график `plot(real(y), imag(y))`. Если **Y** – двумерный действительный массив, то строятся графики для столбцов; в случае комплексных элементов их мнимые части игнорируются.

Команда `plot(x, y)` соответствует построению обычной функции, когда одномерный массив **x** соответствует значениям аргумента, а одномерный массив **y** – значениям функции. Когда один из массивов **X** или **Y** либо оба двумерные, реализуются следующие построения:

- если массив **Y** двумерный, а массив **x** одномерный, то строятся графики для столбцов массива **Y** в зависимости от элементов вектора **x**;
- если двумерным является массив **X**, а массив **y** одномерный, то строятся графики столбцов массива **X** в зависимости от элементов вектора **y**;
- если оба массива **X** и **Y** двумерные, то строятся зависимости столбцов массива **Y** от столбцов массива **X**.

Таблица 2.4. Способы отображения линии у функции plot

Тип линии	Тип точки	Цвет
Непрерывная -	Точка .	Желтый y
Штриховая --	Плюс +	Фиолетовый m
Двойной пунктир :	Звездочка *	Голубой c
Штрих-пунктирная -.	Кружок o	Красный r
	Крестик x	Зеленый g
		Синий b
		Белый w
		Черный k

Команда `plot(x, y, s)` позволяет выделить график функции, указав способ отображения линии, способ отображения точек, цвет линий и точек с помощью строковой переменной `s`, которая может включать до трех символов из следующей таблицы:

Если цвет линии не указан, он выбирается по умолчанию из шести первых цветов, с желтого до синего, повторяясь циклически.

Команда `plot(x1, y1, s1, x2, y2, s2, ...)` позволяет объединить на одном графике несколько функций $y_1(x_1)$, $y_2(x_2)$, ..., определив для каждой из них свой способ отображения.

Обращение к командам `plot` вида `plot(x, y, s1, x, y, s2)` позволяет для графика $y(x)$ определить дополнительные свойства, для указания которых применения одной строковой переменной `s1` недостаточно, например при задании разных цветов для линии и для точек на ней.

Примеры:

Построим график функции $y = \sin(x)$ на отрезке $[-\pi : \pi]$ с шагом $\pi/500$:

```
>> x = -pi:pi/500:pi;
```

```
y = sin(x);  
figure  
plot(y, ['--', 'r'])  
figure  
plot(x, y)
```

Ниже приведены 2 графика, на рис. Б.2 отображает значения одномерного массива **y**, состоящего из 1001 элемента, как функцию от номера элемента; на рис.Б.3 отображает значения того же массива как функцию элементов массива **x**.

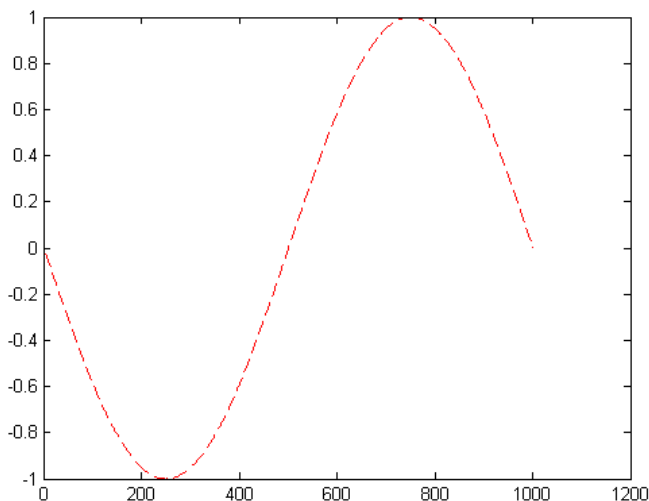
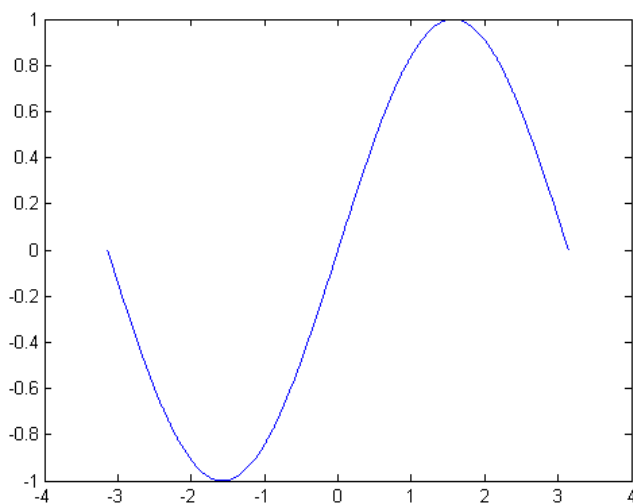


Рис. Б.2. График `plot(y, ['--', 'r'])`

Б.4.2. Трехмерные графики

В системе MATLAB предусмотрено несколько команд и функций для построения трехмерных графиков. Значения элементов числового массива рассматриваются как *z*-координаты точек над плоскостью, определяемой координатами *x* и *y*. Возможно несколько способов соединения этих точек. Первый из них – это соединение

Рис. Б.3. График `plot(x, y)`

точек в сечении (функция `plot3`), второй – построение сетчатых поверхностей (функции `mesh` и `surf`). Поверхность, построенная с помощью функции `mesh` – это сетчатая поверхность, ячейки которой имеют цвет фона, а их границы могут иметь цвет, который определяется свойством `EdgeColor` графического объекта `surface`. Поверхность, построенная с помощью функции `surf` – это сетчатая поверхность, у которой может быть задан цвет не только границы, но и ячейки; последнее управляется свойством `FaceColor` графического объекта `surface`.

PLOT3 – построение линий и точек в трехмерном пространстве.

MESHGRID – формирование двумерных массивов `X` и `Y`.

MESH, MESHZ, MESHZ – трехмерная сетчатая поверхность.

SURF, SURFC – затененная сетчатая поверхность.

GRID – нанесение сетки.

HOLD – управление режимом сохранения текущего графического окна.

SUBPLOT – разбиение графического окна.

CONTOURC – формирование массива описания линий уров-

ня.

CONTOUR – изображение линий уровня для трехмерной поверхности.

CONTOUR3 – изображение трехмерных линий уровня.

plot3(x, y, z), plot3(x, y, z), plot3(x, y, z, s), plot3(x1, y1, z1, s1, x2, y2, z2, s2, ...)

Команды **plot3(...)** являются трехмерными аналогами функции **plot(...)**.

Команда **plot3(x, y, z)**, где x, y, z – одномерные массивы одинакового размера, строит точки с координатами $x(i), y(i), z(i)$ и соединяет их прямыми линиями.

Команда **plot3(X, Y, Z)**, где X, Y, Z – двумерные массивы одинакового размера, строит точки с координатами $x(i, :), y(i, :), z(i, :)$ для каждого столбца и соединяет их прямыми линиями.

Команда **plot3(x, y, z, s)** позволяет выделить график функции $z(x, y)$, указав способ отображения линии, способ отображения точек, цвет линий и точек с помощью строковой переменной s , которая может включать до трех символов из следующей таблицы 2.4.

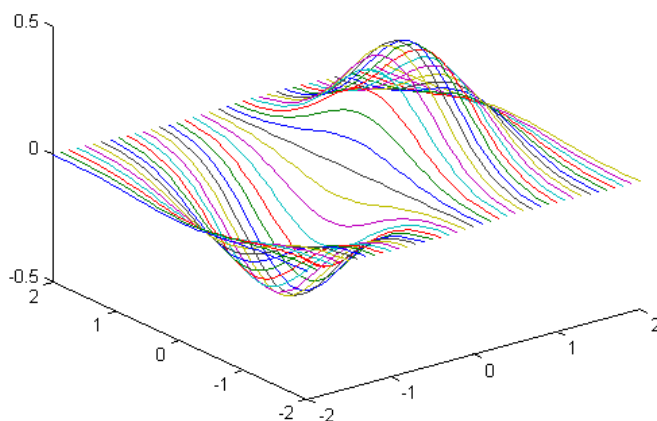
Если цвет линии не указан, он выбирается по умолчанию из шести первых цветов, с желтого до синего, повторяясь циклически.

Команда **plot3(x1, y1, z1, s1, x2, y2, z2, s2, ...)** позволяет объединить на одном графике несколько функций $z_1(x_1, y_1), z_2(x_2, y_2), \dots$, определив для каждой из них свой способ отображения.

Обращение к команде вида **plot3(x, y, z, s1, x, y, z, s2)** позволяет для графика $z(x, y)$ определить дополнительные свойства, для указания которых применения одной строковой переменной s_1 недостаточно, например при задании разных цветов для линии и для точек на ней.

Примеры: построение графика функции $z = x * \exp(-x^2 - y^2)$ в трехмерном пространстве (рисунок Б.4).

```
[ X, Y ] = meshgrid([ -2 : 0.1 : 2 ]);
Z = X .* exp(- X .^ 2 - Y .^ 2);
plot3(X, Y, Z)
```

Рис. Б.4. График `plot3(X, Y, Z)`

<code>mesh(X, Y, Z, C)</code>	<code>meshc(X, Y, Z, C)</code>	<code>meshz(X, Y, Z, C)</code>
<code>mesh(x, y, Z, C)</code>	<code>meshc(x, y, Z, C)</code>	<code>meshz(x, y, Z, C)</code>
<code>mesh(Z, C)</code>	<code>meshc(Z, C)</code>	<code>meshz(Z, C)</code>
<code>mesh(X, Y, Z)</code>	<code>meshc(X, Y, Z)</code>	<code>meshz(X, Y, Z)</code>
<code>mesh(x, y, Z)</code>	<code>meshc(x, y, Z)</code>	<code>meshz(x, y, Z)</code>
<code>mesh(Z)</code>	<code>meshc(Z)</code>	<code>meshz(Z)</code>

Команда `mesh(X, Y, Z, C)` выводит на экран сетчатую поверхность для значений массива **Z**, определенных на множестве значений массивов **X** и **Y**. Цвета узлов поверхности задаются массивом **C**. Цвета ребер определяются свойством `EdgeColor` объекта `surface`. Можно задать одинаковый цвет для всех ребер, определив его в виде вектора `[r g b]` интенсивности трех цветов – красного, зеленого, синего. Если определить спецификацию `none`, то ребра не будут прорисовываться. Если определить спецификацию `flat`, то цвет ребер ячейки определяется цветом того узла, который был первым при обходе этой ячейки. Поскольку одни и те же ребра об-

ходятся несколько раз, то цвета будут замещаться. Если определить спецификацию `interp`, то будет реализована линейная интерполяция цвета между вершинами ребра.

Команда `mesh(x, y, Z, C)` выполняет ту же функцию, но вместо двумерных массивов **X**, **Y** использует их одномерные проекции, так что если `length(x) = n`, а `length(y) = m`, то `[m, n] = size(Z)`. В этом случае узлы сетчатой поверхности определяются тройками $\langle x(j), y(i), Z(i, j) \rangle$, где вектор **x** определяет столбцы массива **Z**, а **y** – строки.

Команда `mesh(Z, C)` использует сетку, которая определяется одномерными массивами $x = 1 : n$ и $y = 1 : m$.

Команды `mesh(X, Y, Z)`, `mesh(x, y, Z)`, `mesh(Z)` используют в качестве массива цвета $C = Z$, то есть цвет в этом случае пропорционален высоте поверхности.

Группа команд `meshc(...)` в дополнение к трехмерным поверхностям строит проекцию линий постоянного уровня.

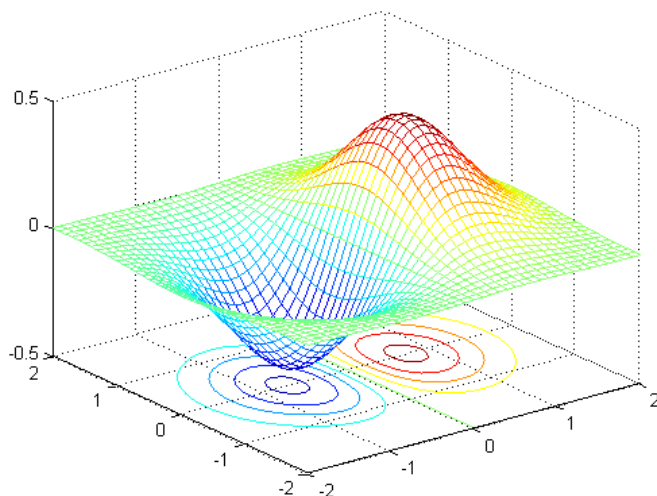
Группа команд `meshz(...)` в дополнение к трехмерным поверхностям строит плоскость отсчета на нулевом уровне, закрывая поверхность, лежащую ниже этого уровня.

Примеры: построение трехмерной поверхности функции $z = x * \exp(-x^2 - y^2)$ с проекциями линий постоянного уровня (рисунки [Б.5](#), [Б.6](#)).

```
[ X, Y ] = meshgrid([ -2 : 0.1 : 2 ]);
Z = X.*exp(-X.^2 - Y.^2);
meshc(X, Y, Z)
```

<code>surf(X, Y, Z, C)</code>	<code>surfc(X, Y, Z, C)</code>
<code>surf(x, y, Z, C)</code>	<code>surfc(x, y, Z, C)</code>
<code>surf(Z, C)</code>	<code>surfc(Z, C)</code>
<code>surf(X, Y, Z)</code>	<code>surfc(X, Y, Z)</code>
<code>surf(x, y, Z)</code>	<code>surfc(x, y, Z)</code>
<code>surf(Z)</code>	<code>surfc(Z)</code>

Команда `surf(X, Y, Z, C)` выводит на экран сетчатую поверхность для значений массива **Z**, определенных на множестве значений массивов **X** и **Y**. Цвет ячейки определяется массивом **C**. Цвет

Рис. Б.5. График `meshc(X, Y, Z)`

ребер – черный, определяется свойством `EdgeColor`, специфицированным как `[0 0 0]`. Можно задать одинаковый цвет для всех ребер, определив его в виде вектора `[r g b]` интенсивности трех цветов – красного, зеленого, синего. Если определить спецификацию `none`, то ребра не будут прорисовываться.

Команда `surf(x, y, Z, C)` выполняет ту же функцию, но вместо двумерных массивов **X**, **Y** использует их одномерные проекции, так что если `length(x) = n`, а `length(y) = m`, то `[m, n] = size(Z)`. В этом случае узлы сетчатой поверхности определяются тройками `<x(j), y(i), Z(i, j)>`, где вектор **x** определяет столбцы массива **Z**, а **y** – строки.

Команда `surf(Z, C)` использует сетку, которая определяется одномерными массивами `x = 1 : n` и `y = 1 : m`.

Команды `surf(X, Y, Z)`, `surf(x, y, Z)`, `surf(Z)` используют в качестве массива цвета `C = Z`, то есть цвет в этом случае пропорционален высоте поверхности.

Группа команд `surfc(...)` в дополнение к трехмерным затененным поверхностям строит проекцию линий постоянного уровня.

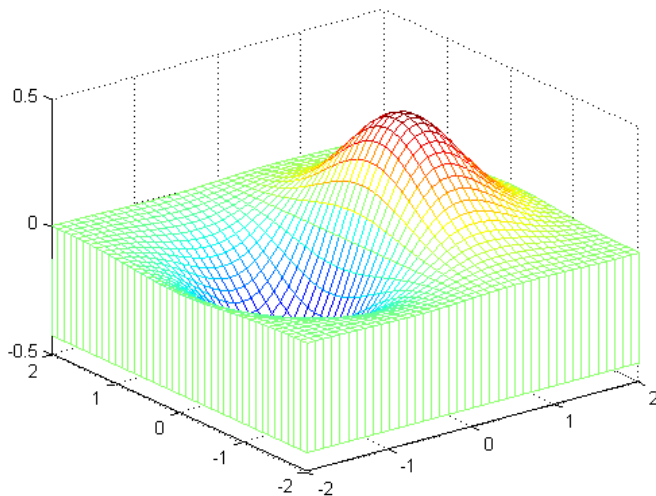


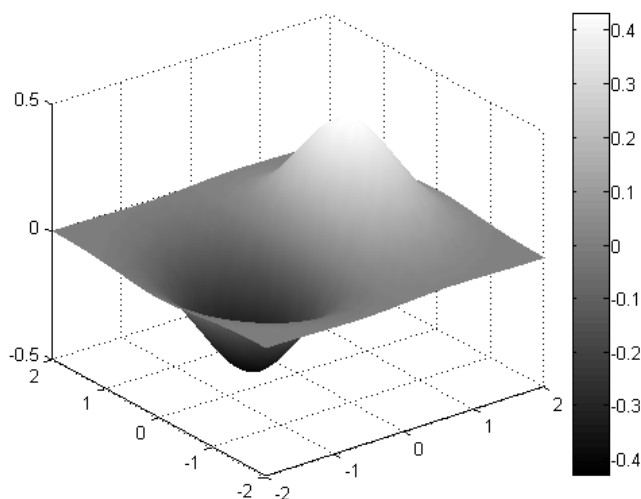
Рис. Б.6. График `meshz(X, Y, Z)`

Примеры: построения трехмерной затененной поверхности функции $z = x * \exp(-x^2 - y^2)$ со шкалой затененности (рисунки Б.7).

```
[ X, Y ] = meshgrid([ -2 : 0.1 : 2 ]);
Z = X.*exp(-X.^2 - Y.^2);
surf(X, Y, Z)
colormap(gray)
shading interp
colorbar
```

`surf1(X, Y, Z, s)`, `surf1(Z, s)`, `surf1(X, Y, Z, s, k)`, `surf1(Z, s, k)`, `surf1(X, Y, Z)`, `surf1(Z)`

Команда `surf1(X, Y, Z, s)` выводит на экран затененную поверхность с подсветкой для значений массива **Z**, определенных на множестве значений массивов **X** и **Y**. Направление на источник света может быть задано с помощью вектора **s** = [Sx, Sy, Sz] в

Рис. Б.7. График `surf(X, Y, Z)`

декартовых координатах или вектора $\mathbf{s} = [\text{az}, \text{elev}]$ в сферических координатах. По умолчанию азимут $\text{az} = -37.5^\circ$, возвышение $\text{elev} = 30^\circ$. Подсветка учитывает модели рассеяния, отражения и зеркального эффекта освещения поверхности.

Команда `surf1(X, Y, Z, s, k)` позволяет управлять параметрами рассеяния, отражения и зеркального эффекта, используя вектор $\mathbf{k} = [\text{ka}, \text{kd}, \text{ks}, \text{spread}]$, который учитывает эффекты отраженного света ka , диффузного отражения kd , зеркального отражения ks и зеркального распространения `spread`. По умолчанию вектор \mathbf{k} имеет значения `[0.55 0.6 0.4 10]`.

Команда `surf1(X, Y, Z)` использует значения параметров по умолчанию.

Команды `surf1(Z, ...)` строят графики, не учитывая истинных значений массивов \mathbf{X} и \mathbf{Y} .

Из-за того что алгоритм `surf1` вычисляет нормали к поверхности, необходимо, чтобы входные матрицы имели размер по крайней мере 3×3 .

Примеры: построение изображения функции `peaks`, используя

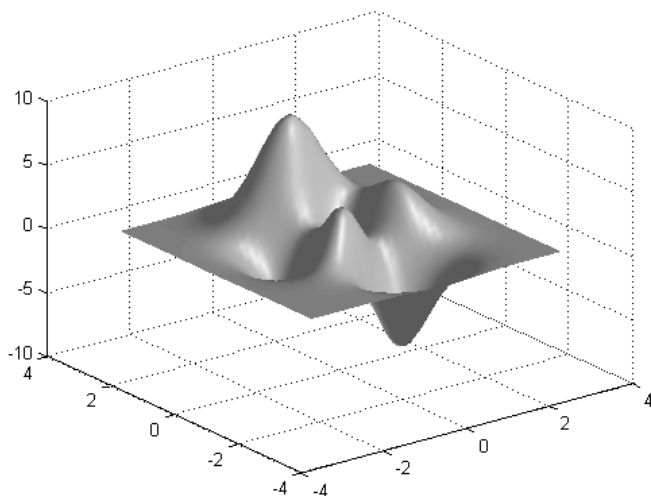


Рис. Б.8. График `surfl(X, Y, Z)`

подсветку (рисунок Б.8).

```
[X, Y] = meshgrid(-3 : 1/8 : 3);  
Z = peaks(X, Y);  
surfl(X, Y, Z)  
shading interp  
colormap(gray)
```

grid on, grid off, grid

Команда `grid on` наносит координатную сетку на текущие оси.

Команда `grid off` удаляет координатную сетку.

Команда `grid` выполняет роль переключателя с одной функции на другую.

hold on, hold off, hold

Команда `hold on` включает режим сохранения текущего графика и свойств объекта `axes`, так что последующие команды приведут к добавлению новых графиков в графическом окне.

Команда `hold off` выключает режим сохранения графика.

Команда `hold` реализует переключение от одного режима к другому.

Команды `hold` воздействуют на значения свойства `NextPlot` объектов `figure` и `axes`:

`hold on` присваивает свойству `NextPlot` для текущих объектов `figure` и `axes` значение `add`;

`hold off` присваивает свойству `NextPlot` для текущих объектов `figure` и `axes` значение `replace`.

`subplot(m, n, p)`, `subplot(h)`, `subplot(mnp)`

Данная команда выполняется перед обращением к функциям построения графиков для одновременной выдачи нескольких графиков в различных частях графического окна.

Команды `subplot(mnp)` или `subplot(m, n, p)`, где `mnp` – 3 цифры, производит разбивку графического окна на несколько подокон, создавая при этом новые объекты `axes`; значение `m` указывает, на сколько частей разбивается окно по горизонтали, `n` – по вертикали, а `p` – номер подокна, куда будет выводиться очередной график. Эти же команды могут использоваться для перехода от одного подокна к другому.

Команда `subplot(h)`, где `h` – дескриптор для объекта `axes` соответствующего подокна, – другой способ выбора подокна для размещения графика.

Команды `clf`, `subplot(111)`, `subplot(1, 1, 1)` выполняют одну и ту же функцию – удаляют все подокна и возвращают графическое окно в штатное состояние.

Пример: в верхней части экрана строится функция $y_1 = \sin(x)$, в нижней – $y_2 = \log(\text{abs}(y))$ (рисунок Б.9).

```
x = -1:.1:1;
y1 = sin(x);
subplot(2, 1, 1), plot(x, y1)
y2 = log(abs(y1));
```

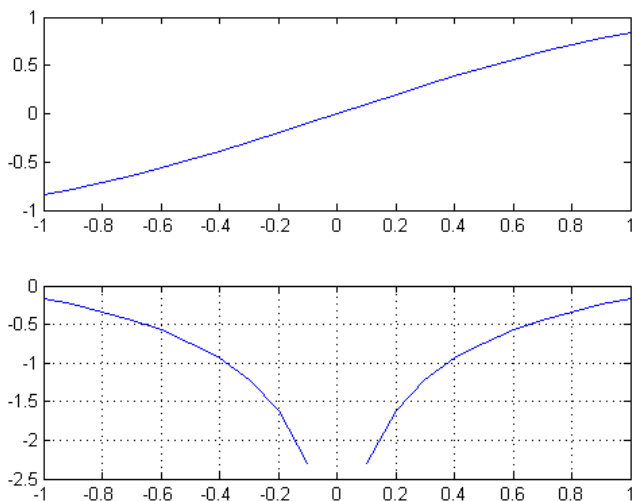


Рис. Б.9. Построение графика с использованием subplot и grid

```
subplot(2, 1, 2), plot(x, y2)
grid on
```

contour(Z), contour(x, y, Z), contour(Z, n), contour(x, y, Z, n), contour(Z, v), contour(x, y, Z, v), contour(...,'тип_линии'), C = contour(...), [C, h] = contour(...)

Команда **contour(Z)** рисует двумерные линии уровня для массива данных **Z**, определяющих поверхность в трехмерном пространстве без учета диапазона изменения координат **x** и **y**.

Команда **contour(x, y, Z)**, где **x** и **y** – векторы, рисует линии уровня для массива данных **Z** с учетом диапазона изменения координат **x** и **y**.

Команды **contour(Z, n), contour(x, y, Z, n)** рисует *n* линий уровня для массива данных **Z**; по умолчанию, *n* равно 10.

Команды **contour(Z, v), contour(x, y, Z, v)** рисуют линии уровня для заданных значений, которые указаны в векторе **v**.

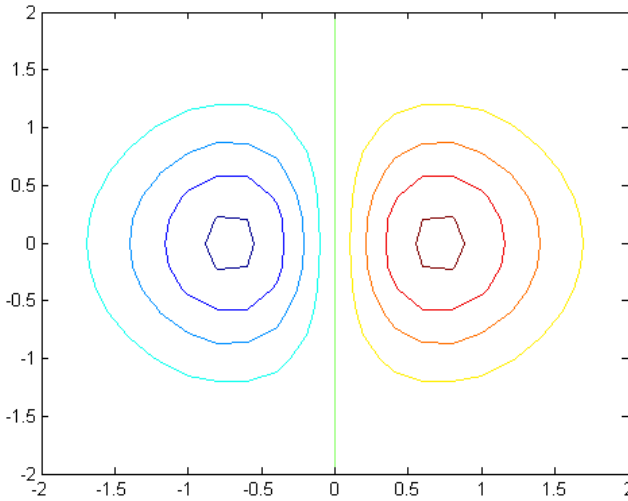


Рис. Б.10. Построение графика с использованием `contour`

Команда `contour(..., '<тип_линии>')` рисует линии уровня, тип и цвет которых определяются параметром `<тип_линии>` команды `plot`.

Функция `C = contour(...)` возвращает массив `C` описания линий уровней по аналогии с функцией `contourc` для последующего использования командой `clabel`.

Функция `[C, h] = contour(...)` возвращает массив `C` и вектор-столбец дескрипторов `h` графических объектов `line` для каждой линии уровня.

Пример: построение линий уровня функции $z(x, y) = x \exp(-x^2 - y^2)$ в области $-2 \leq x \leq 2$, $-2 \leq y \leq 2$ (рисунок Б.10).

```
x = -2 : .2 : 2;  
y = x;  
[X, Y] = meshgrid(x);  
Z = X.* exp(- X.^2 - Y.^2);  
contour(X, Y, Z)
```

contour3(Z), contour3(X, Y, Z), contour3(Z, n), contour3(X, Y, Z, n), C = contour3(...), [C, h] = contour3(...)

Команда **contour3(Z)** рисует трехмерные линии уровня для массива данных **Z**, определяющих поверхность в трехмерном пространстве без учета диапазона изменения координат **x** и **y**.

Команда **contour3(X, Y, Z)**, где **X** и **Y** – двумерные массивы, вычисленные с помощью функции **meshgrid**, рисует линии уровня для массива данных **Z** с учетом диапазона изменения координат **x** и **y**.

Команды **contour3(Z, n)**, **contour(X, Y, Z, n)** рисуют n линий уровня для массива данных **Z**; по умолчанию n равно 10.

Функция **C = contour3(...)** возвращает массив **C** описания линий уровней по аналогии с функцией **contourc** для последующего использования командой **clabel**.

Функция **[C, h] = contour3(...)** возвращает массив **C** и вектор-столбец дескрипторов **h** графических объектов **line** для каждой линии уровня.

Пример: построение линий уровня функции $z(x, y) = x \cdot \exp(-x^2 - y^2)$ в области $-2 \leq x \leq 2$, $-2 \leq y \leq 3$ (рисунок Б.11).

```
x = -2 : .2 : 2;
y = -2 : .2 : 2;
[X, Y] = meshgrid(x, y);
Z = X.* exp(- X.^2 - Y.^2);
contour3(X, Y, Z)
```

Б.4.3. Надписи и пояснения к графикам

TITLE – заголовки для двух- и трехмерных графиков

XLABEL, YLABEL, ZLABEL – обозначение осей

CLABEL – маркировка линий уровня

TEXT – добавление к текущему графику текста

GTEXT – размещает заданный текст на графике с использованием мыши

LEGEND – пояснение к графику

COLORBAR – шкала палитры

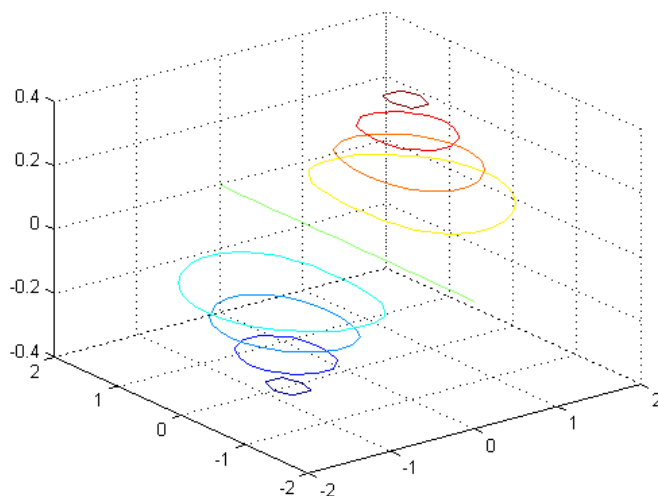


Рис. Б.11. Построение графика с использованием `contour3`

Команда `title('<текст>')` размещает текст над графиком.

Команда `xlabel('<текст>')` размещает текст для двумерного графика вдоль оси x , для трехмерного графика – вдоль оси x либо под графиком.

Команда `ylabel('<текст>')` размещает текст для двумерного графика вдоль оси y , для трехмерного графика – вдоль оси y либо под графиком.

Команда `zlabel('<текст>')` размещает текст вдоль оси трехмерного графика.

Повторное использование команды приводит к замене старого текста новым.

`clabel(C)`, `clabel(C, v)`, `clabel(C, 'manual')`

Команда `clabel(C)` добавляет метки к линиям уровня в случайно выбранных позициях.

Команда `clabel(C, v)` маркирует линии уровня контура, которые заданы в векторе v .

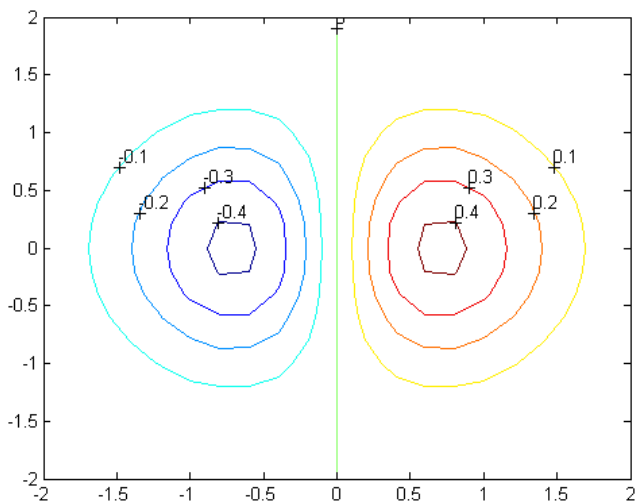


Рис. Б.12. Построение графика с использованием `contour3`

Команда `clabel(C, 'manual')` маркирует линии уровней в позициях, указываемых с помощью мыши. Нажатие клавиши Return или правой кнопки мыши завершает маркировку.

Пример: маркировка линий уровня для функции $z(x, y) = x * \exp(-x^2 - y^2)$ (рисунок Б.12).

```
x = -2 : .2 : 2;
y = x;
[X, Y] = meshgrid(x);
Z = X.* exp(- X.^2 - Y.^2);
C = contour(X, Y, Z);
clabel(C)
```

```
text(x,y,'<текст>'), text(x,y,z,'<текст>'), gtext('<текст>')
```

Команда `text(x, y, '<текст>')` помещает в заданной точке (x, y) двумерного графика начало текста, указанного в качестве тре-

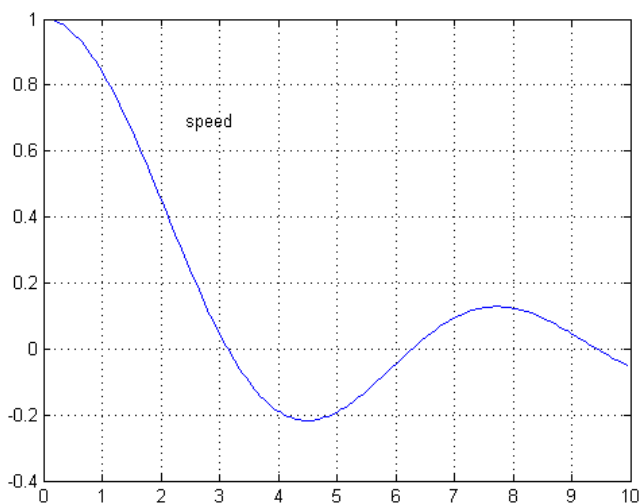


Рис. Б.13. Построение графика с использованием `gtext`

того аргумента. Если x и y одномерные массивы, заданный текст помещается во все позиции, определяемые координатами $[x(i) \ y(i)]$.

Команда `text(x, y, z, '<текст>')` выводит текст на трехмерный график.

Команда `gtext('<текст>')` высвечивает в активном графическом окне перекрестие, перемещение которого позволяет указать место ввода заданного текста; по завершении позиционирования нажатие кнопки мыши или любой клавиши вводит заданный текст.

Примеры: Размещение на графике функции $y = \sin(x)/x$ маркер «speed» (рисунок Б.13).

```
x = 1e-6 : 0.05 : 10;  
plot(x, sin(x)./x)  
grid  
gtext('speed')
```

```
legend('<текст1>', '<текст2>', '<текст3>', ...),
```

```
legend('<тип линии1>','<текст1>','<тип линии2>',
'<текст2>', ...),
```

```
legend(h,...), legend(M), legend(h, M), legend off ,
legend(..., n)
```

Команда `legend('<текст1>', '<текст2>', '<текст3>', ...)` добавляет к текущему графику пояснение в виде указанных текстовых строк.

Команда `legend('<тип линии1>', '<текст1>', '<тип линии2>', '<текст2>', ...)` позволяет специфицировать тип линии, которая выносится в пояснение, так, как это делается в команде `plot`.

Команда `legend(h, ...)` добавляет пояснение к графику с дескриптором `h`.

Команды `legend(M)` и `legend(h, M)`, где **M** – массив строк, также допустимы для формирования пояснения. Следует помнить, что строки массива **M** должны иметь одинаковую длину.

Команда `legend off` удаляет пояснение с текущего графика.

Команда `legend(..., n)` устанавливает предельное количество позиций для размещения пояснения. Если оказывается, что в области графика места недостаточно, график перестраивается и пояснение размещается вне пределов графика. Если $n = -1$, то пояснение размещается вне области графика. Если $n = 0$, то пояснение размещается в области графика, если места для этого достаточно.

Для перемещения пояснения следует нажать левую кнопку мыши, находясь в этой области, а затем переместить пояснение в нужную позицию.

Пример: Построение на одном графике функции Бесселя 1, 3 и 5-го порядка и размещение соответствующих пояснений (рисунки Б.14).

```
x = 0 : .2 : 12;
hp = plot(x, bessell(1, x), '-', x, bessell(3, x), '--', ...
x, bessell(5, x), '-.');
set(gca, 'FontName', 'TimesET')
set(gca, 'FontSize', 10)
legend('Bessel 1', 'Bessel 3', 'Bessel 5', -1)
grid
```

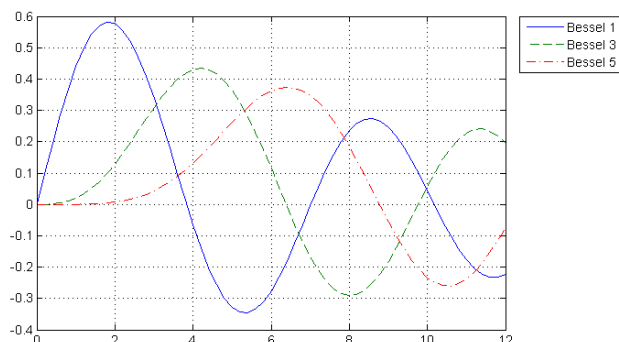


Рис. Б.14. Построение графика с использованием `gtext`

`colorbar('vert')`, `colorbar('horiz')`, `colorbar(h)`, `colorbar`

Команда `colorbar('vert')` добавляет к текущему графику вертикальную шкалу палитры.

Команда `colorbar('horiz')` добавляет к текущему графику горизонтальную шкалу палитры.

Команда `colorbar(h)` добавляет к графику с дескриптором `h` шкалу палитры. Размещение шкалы реализуется автоматически в зависимости от соотношения ширины и высоты графика.

Команда `colorbar` без аргументов размещает на текущем графике новую вертикальную шкалу палитры или обновляет прежнюю.

Б.4.4. Специальная графика

QUIVER – поле градиентов функции

SLICE – сечения функции от трех переменных

`quiver(X, Y, DX, DY)`, `quiver(x, y, DX, DY)`,
`quiver(DX, DY)`, `quiver(... '<тип_линии>')`,
`quiver(x, y, dx, dy, s)`, `quiver(dx, dy, s)`

Команда `quiver(X, Y, DX, DY)` формирует и выводит на экран поле градиентов функции в виде стрелок для каждой пары элемен-

тов массивов **X** и **Y**, а пары элементов **DX** и **DY** используются для указания направления и размера стрелки.

Команда `quiver(x, y, DX, DY)`, где **x** и **y** – одномерные массивы размеров $length(x) = n$ и $length(y) = m$, где $[m, n] = size(DX) = size(DY)$, формирует и выводит на экран поле градиентов для каждой точки; стрелки задаются четверками $\langle x(j), y(i), DX(i, j), DY(i, j) \rangle$. Обратите внимание, что **x** соответствует столбцам **DX** и **DY**, а **y** – строкам.

Команда `quiver(DX, DY)` использует массивы $x = 1 : n$ и $y = 1 : m$.

Команды `quiver(x, y, dx, dy, s)`, `quiver(dx, dy, s)` используют скаляр s как коэффициент масштаба стрелки, например $s = 2$ вдвое увеличивает, а $s = 0.5$ вдвое уменьшает размер стрелки.

Команда в форме `quiver(... 'тип_линии')` позволяет задать тип и цвет линии по аналогии с функцией `PLOT`.

Пример: построение поля направлений для функции $z(x, y) = x * \exp(-x^2 - y^2)$ в области $-2 \leq x \leq 2$, $-2 \leq y \leq 2$ (рисунок Б.15).

```
[x, y] = meshgrid(-2 : .2 : 2);
z = x.*exp(-x.^2 - y.^2);
[dx, dy] = gradient(z, .2, .2);
contour(x, y, z), hold on
quiver(x, y, dx, dy, 2)
```

`slice(x, y, z, V, xi, yi, zi, n), slice(X, Y, Z, V, xi, yi, zi, n), slice(V, xi, yi, zi, n), h = slice(...)`

Команда `slice(x, y, z, V, xi, yi, zi, n)` строит плоские сечения функции от трех переменных $v(x, y, z)$ вдоль осей x, y, z ; позиции сечений определяются векторами xi, yi, zi . Размер трехмерного массива **V** равен $m \times n \times p$, где $m = length(y)$, $n = length(x)$, $p = length(z)$.

Команда `slice(X, Y, Z, V, xi, yi, zi, n)` вместо одномерных массивов использует двумерные массивы **X, Y, Z**, которые вычисляются с помощью функции `meshgrid`.

Команда `slice(V, xi, yi, zi, n)` использует для задания области построения массивы $x = 1 : n$, $y = 1 : m$, $z = 1 : p$.

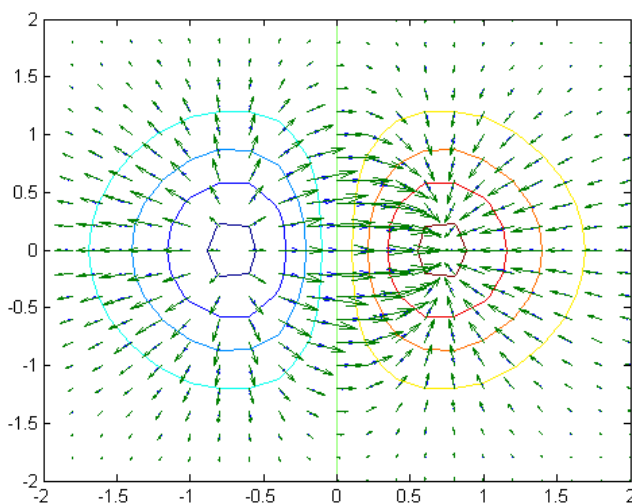


Рис. Б.15. Построение графика с использованием quiver

Функция `h = slice(...)` возвращает вектор-столбец дескрипторов для графических объектов `surface`, которыми являются сечения трехмерной функции.

Пример: построение сечения функции $V(x, y, z) = x*y*z \exp(-x^2 - y^2 - z^2)$ и $V1(x, y, z) = x \exp(-x^2 - y^2 - z^2)$ в трехмерной области $-2 \leq x \leq 2$, $-2 \leq y \leq 2$, $-2 \leq z \leq 2$ (рисунки Б.16, Б.17).

```
x = -2 : .2 : 2; y = -2 : .25 : 2;
z = -2 : .16 : 2;
[X, Y, Z] = meshgrid(x, y, z);
V = X.*Y.*Z.*exp(-X.^2 - Y.^2 - Z.^2);
V1 = X.*exp(-X.^2 - Y.^2 - Z.^2);
figure
slice(x, y, z, V, [2], [2], [-0.75 0.5], length(x))
grid on
figure
slice(x, y, z, V1, [1], [0], [0], length(x))
grid on
```

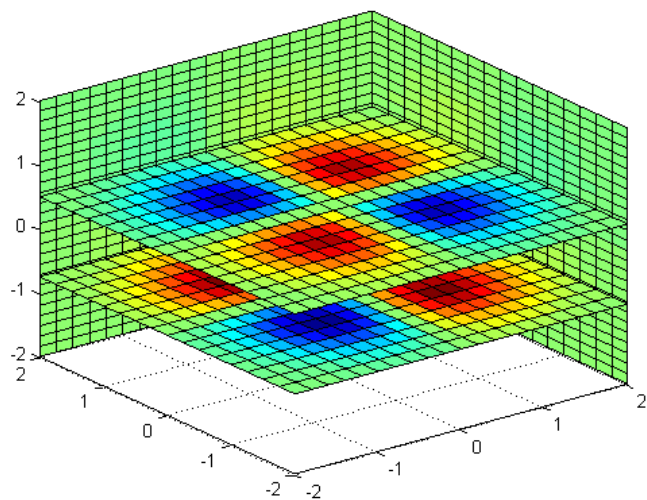


Рис. Б.16. Построение графика с использованием slice

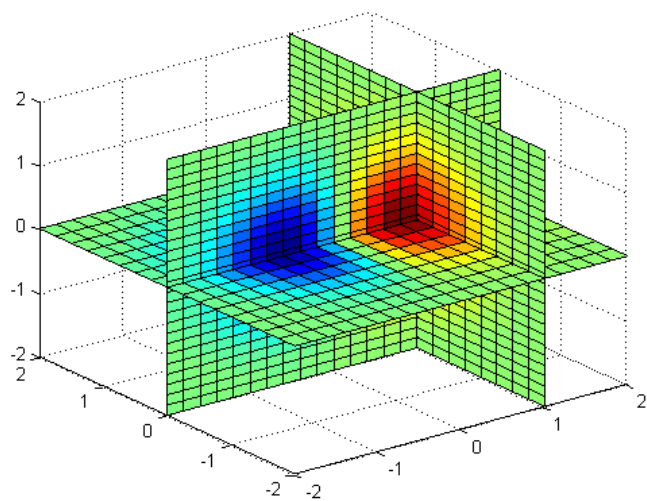


Рис. Б.17. Построение графика с использованием slice

Приложение В

Векторный анализ

Векторный анализ используется при исследовании векторных функций методами дифференциального и интегрального исчисления [9].

В.1. Умножение векторов

Скалярным произведением векторов \mathbf{a} и \mathbf{b} (обозначение $\mathbf{a}\mathbf{b}$, $\langle \mathbf{a}, \mathbf{b} \rangle$, (\mathbf{a}, \mathbf{b}) , $\mathbf{a} \cdot \mathbf{b}$ или $(\mathbf{a} \cdot \mathbf{b})$) называется число $\mathbf{a}\mathbf{b} = |\mathbf{a}| \cdot |\mathbf{b}| \cos \phi$, где ϕ – угол между векторами \mathbf{a} и \mathbf{b} [9].

Скалярное произведение векторов, заданных своими координатами, равно сумме произведений соответствующих координат

$$\mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z, \quad (\text{В.1})$$

где (a_x, a_y, a_z) и (b_x, b_y, b_z) – координаты векторов \mathbf{a} и \mathbf{b} .

Векторным произведением векторов \mathbf{a} и \mathbf{b} (обозначение $\mathbf{a} \times \mathbf{b}$, $[\mathbf{a} \cdot \mathbf{b}]$, $[\mathbf{a}, \mathbf{b}]$, $[\mathbf{a}\mathbf{b}]$ или $[\mathbf{a} \times \mathbf{b}]$) называется вектор, длиной $|| = |\mathbf{a} \times \mathbf{b}| = |\mathbf{a}||\mathbf{b}|\sin \phi$ (площадь параллелограмма, построенного на \mathbf{a} и \mathbf{b} как на сторонах) и направленный перпендикулярно к \mathbf{a} и \mathbf{b} , причем так, что векторы \mathbf{a} , \mathbf{b} , $= \mathbf{a} \times \mathbf{b}$ образуют правую тройку векторов [9]. В декартовой системе координат векторное произведение может

имеет вид:

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix} = (a_y b_z - a_z b_y)\mathbf{i} + (a_z b_x - a_x b_z)\mathbf{j} + (a_x b_y - a_y b_x)\mathbf{k}. \quad (\text{B.2})$$

В.2. Градиент скалярного поля

Градиентом поля $U(\mathbf{r})$ называется вектор (обозначение $\text{grad}U$, ∇U), определяемый в каждой точке поля соотношением для декартовой системы координат [9]

$$\text{grad}U = \frac{\partial U}{\partial x}\mathbf{i} + \frac{\partial U}{\partial y}\mathbf{j} + \frac{\partial U}{\partial z}\mathbf{k}; \quad (\text{B.3})$$

тогда

$$\frac{\partial U(\mathbf{r})}{\partial n} = \mathbf{n} \text{grad}U. \quad (\text{B.4})$$

В.3. Дивергенция векторного поля

Дивергенцией векторного поля $\mathbf{V}(M)$ (обозначение $\text{div}\mathbf{V}$, $\nabla\mathbf{V}$) называют следующую производную по объему поля в точке M , которая для декартовой системы координат будет иметь вид [9]:

$$\text{div}\mathbf{V} = \frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z}. \quad (\text{B.5})$$

В.4. Оператор Лапласа

Пусть $U(M)$ – скалярное поле; тогда оператором Лапласа ΔU (обозначение $\nabla^2 U$, ΔU) для декартовой системы координат определяется следующим образом:

$$\Delta U(M) = \text{div grad } U(M) = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2}. \quad (\text{B.6})$$

Приложение Г

Краткая инструкция по работе в GMSH

Введение

Gmsh – это свободно распространяемая программа под лицензией GNU General Public License (GPL), которая позволяет создавать трёхмерные конечно-элементные сетки для расчёта МКЭ. Gmsh снабжён САПР подобным движком для создания геометрии расчётной области. Благодаря библиотеке OpenCascade позволяет импортировать геометрию в формате iges, step и включает довольно мощный постпроцессор.

Включает сборки для всех основных платформ Windows, Linux, Mac OS.

Gmsh распространяется по лицензии GNU General Public License v2 или более поздней версии. Является свободным программным обеспечением с одним ограничением. При написании статей и других публикаций вы обязаны написать ссылку на этот ресурс, которая будет следующей:

C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering, Volume 79, Issue 11, pages 1309-1331, 2009.

Разработчик	Christophe Geuzaine, Jean-François Remacle
Написана на	C++
ОС	Windows, Linux, Mac OS X
Последняя версия	2010
Лицензия	GNU GPL (with linking exception)
Сайт	http://www.geuz.org/gmsh/

Г.1. Начало работы

На рис.Г.1 представлены скриншоты с Gmsh после запуска.

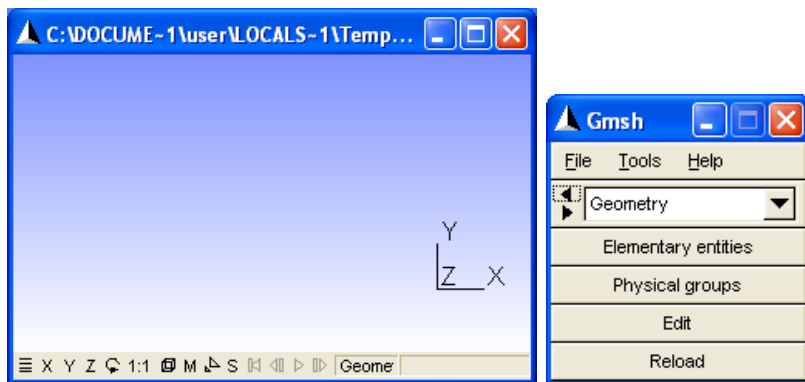


Рис. Г.1. Скриншоты с Gmsh

Можно пользоваться:

- интерфейсом для создания геометрии;
- загрузить файл в формате iges, step;
- создать файл в командном режиме в формате geo.

Г.2. Создания файла в командном режиме

Для того чтобы отредактировать файл, необходимо нажать кнопку Edit (рис.Г.1). Будет предложено создать файл, а файл будет открыт в блокноте. Ниже приведен пример кода:

```
lc = 0.1;  
Point(1) = {0.0, 0.0, 0.0, lc};  
Point(2) = {1, 0.0, 0.0, lc};  
Point(3) = {0.0, 1, 0.0, lc};
```

```
Line(1) = {1,2};  
Line(2) = {2,3};  
Line(3) = {3,1};
```

```
Line Loop(4) = {1,2,3};  
Plane Surface(5) = {4};
```

Разберем текст: первая строка присваивает переменной lc значение 0.1, которая является параметром для размера элемента (чем меньше lc , тем гуще будет построена сетка). На рис.Г.2 приведен результат геометрического построения.

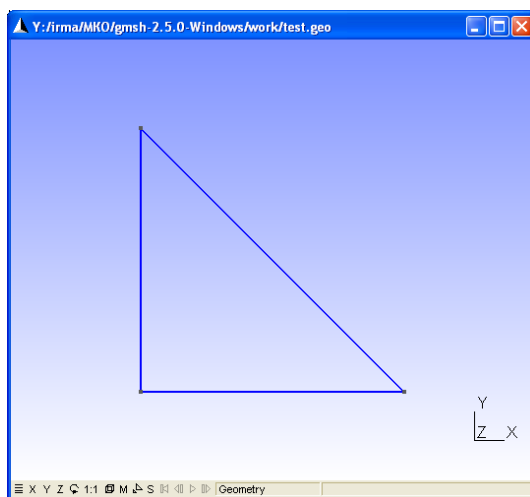


Рис. Г.2. Результат геометрического построения

Г.3. Наложение сетки

Для того чтобы наложить сетку, в верхней панели выбираем Mesh (Рис.Г.3). Появляется меню наложения сетки. Поскольку наша фигура двумерная, выбираем 2D, результат наложения сетки показан на рис.Г.3 .

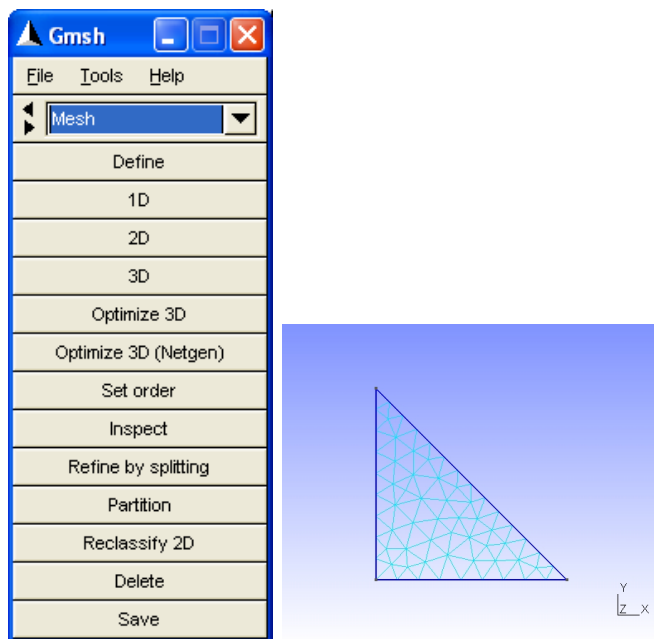


Рис. Г.3. Меню Mesh и наложенная сетка

Г.4. Формат файлов .msh ASCII

Файл, содержащий данные сетки, состоит из следующих обязательных разделов: **\$MeshFormat**, затем следуют разделы, в которых описаны узлы (**\$Nodes**), элементы (**\$Elements**), наименования областей (**\$PhysicalName**) и данные построения (**\$NodeData**, **\$ElementData**, **\$ElementNodeData**).

Файл имеет следующий формат (часть областей в формате опущено, полное описание приведено в руководстве пользователя):

```
$MeshFormat
version-number file-type data-size
$EndMeshFormat
$Nodes
number-of-nodes
node-number x-coord y-coord z-coord
...
$EndNodes
$Elements
number-of-elements
elm-number elm-type number-of-tags < tag > ... node-number-list
...
$EndElements
```

где *version-number* – действительное число, равное 2.2; *file-type* – целое число, равное 0, для файлов формата ASCII; *data-size* – целое число, показывающее точность данных в файле (поддерживает `sizeof(double)`).

number-of-nodes – число узлов в сетке.

node-number – номер узла, является положительным целым не нулевым числом, далее идут *x-coord y-coord z-coord* – *x*, *y* и *z* координаты, являющиеся числами с плавающей запятой.

number-of-elements – количество элементов в сетке.

elm-number – номер элемента в сетке, является положительным ненулевым целым числом.

elm-type – задает тип элемента (рис.Г.4):

- 1 – 2-узловая линия;
- 2 – плоский треугольник, построенный на 3-х узлах;
- 3 – плоский прямоугольник, построенный на 4-х узлах;
- 4 – плоский четырехугольник, построенный на 4-х узлах;
- 5 – куб, построенный на 8-ми узлах;
- 6 – призма, построенная на 6-ти узлах;
- 7 – пирамида, построенная на 5-ти узлах;

15 – вершина, 1 узел.

Gmsh поддерживает 31 тип элементов, в том числе и второго порядка. Все типы элементов описаны в руководстве пользователя, который можно скачать на официальном сайте.

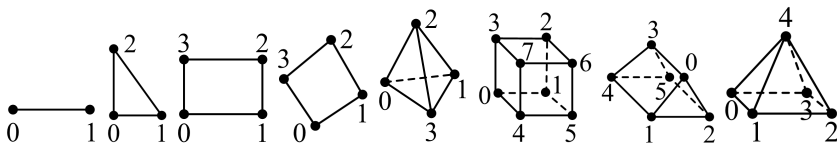


Рис. Г.4. Элементы

number-of-tags – задает количество тэгов для текущего элемента и является целым числом. Далее идут тэги, первый описывает физический объект (или характеристику), которому он принадлежит; второй тэг является номером элементарной геометрической фигуры, которой принадлежит данный элемент.

node-number-list – список номеров узлов сетки, на которых построен элемент.

Описание остальных областей (*\$PhysicalName*, *\$NodeData*, *\$ElementData*, *\$ElementNodeData*) приведены в руководстве пользователя.

Ниже приведен пример файла, содержащего сеточные данные, полученные при помощи Gmsh для плоской фигуры, приведенной на рис.Г.2 (комментарии добавлены вручную и не являются частью полученного файла):

```
$MeshFormat
2.2 0 8
$EndMeshFormat
$Nodes
11 \\ Количество узлов
1 0 0 0 \\ № узла и координаты x, y, z
2 1 0 0 \\ поскольку фигура плоская для всех точек z=0
3 0 1 0
4 0.33333333333326071 0 0
5 0.66666666666658988 0 0
```

```
6 0.75000000000004735 0.2499999999995265 0
7 0.5000000000011831 0.499999999998817 0
8 0.25000000000008518 0.749999999991482 0
9 0 0.66666666666673606 0
10 0 0.33333333333344899 0
11 0.30555555555558525 0.3055555555553303 0
$EndNodes
$Elements
23 \\ количество элементов
1 15 2 0 1 1 \\ № узла, тип элемента (15) - вершина
2 15 2 0 2 2 \\ первый тэг = 0 - не накладывали физ. хар-ки
3 15 2 0 3 3 \\ второй тэг - № точки (Point)
    \\ последняя цифра - № узла,
    \\ на котором построен элемент

4 1 2 0 1 1 4 \\ тип элемента (1) - линия
5 1 2 0 1 4 5 \\ второй тэг - № линии (Line)
6 1 2 0 1 5 2 \\ последние две цифры - № узлов элемента
7 1 2 0 2 2 6 \\ их в данном случае 2,
8 1 2 0 2 6 7 \\ т.к. элементом является линия
9 1 2 0 2 7 8
10 1 2 0 2 8 3
11 1 2 0 3 3 9
12 1 2 0 3 9 10
13 1 2 0 3 10 1

14 2 2 0 5 2 6 5 \\ тип элемента (2) - треугольник,
15 2 2 0 5 9 8 3 \\ второй тэг - № плоскости 5 (Plane Surface),
16 2 2 0 5 11 1 4 \\ которой принадлежит элемент
17 2 2 0 5 10 1 11 \\ последние три цифры - № узлов, на которых
18 2 2 0 5 8 11 7 \\ построен элемент (3 узла)
19 2 2 0 5 7 11 6
20 2 2 0 5 11 8 9
21 2 2 0 5 10 11 9
22 2 2 0 5 11 4 5
23 2 2 0 5 6 11 5
$EndElements
```

На рис. Г.5 представлена сетка с нумерацией узлов и элементов, описанных в приведенном выше файле.

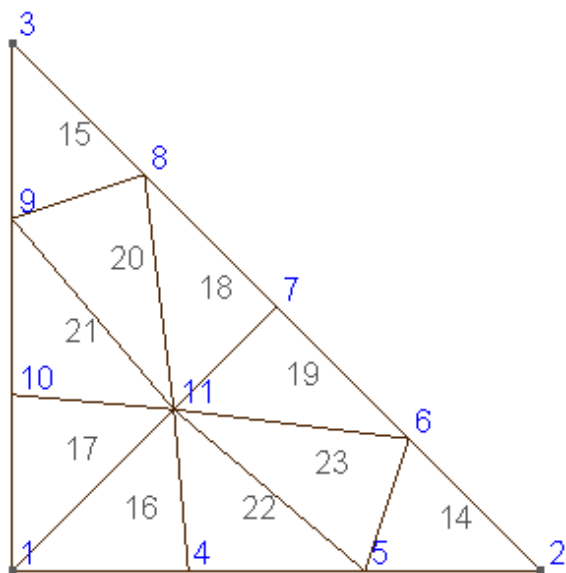


Рис. Г.5. Наложенная сетка с нумерацией узлов и элементов

Приложение Д

Реализация метода Якоби на C++

Программа реализует итерационный алгоритм решения СЛАУ.

Поскольку нулевые недиагональные элементы матрицы коэффициентов не участвуют в расчетах, то не имеет смысла их значение хранить. Это позволит значительно сэкономить память, что особенно важно при решении СЛАУ большой размерности. Поэтому в программу была внесена проверка на «0» недиагональных элементов.

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <cstdlib>
#include <math.h>
#include <el.h> //подключение класса
using namespace std;
int main(int argc, char **argv)
{
    int N; // Размерность СЛАУ
    // Вектор диагональных элементов матрицы коэффициентов
    vector <float> Aii;
    // Матрица недиагональных элементов
```

```
vector < vector<el> > Aij;
vector <float> B; // Вектор свободных членов
float Err; // точность расчета
// Считывание данных из файла
ifstream inFile("a1a1.txt");//открытие файла a1a1.txt
if(!inFile) {
    cerr<<"Error! Can't open file\n";
    return -1;
}
string buffer; // объявляем переменную типа строка
while(!inFile.eof()) { // считывать до конца файла
    inFile>>buffer;
    if (buffer=="$Err")
        inFile>>Err;
    if (buffer=="$N")
        inFile>>N;
    if (buffer=="$B")
        for (int i=0; i<N; ++i) {
            float temp;
            inFile>>temp;
            B.push_back(temp);
        }
    if (buffer=="$A")
        for (int i=0; i<N; ++i) {
            vector <el> temp_vect;
            for (int j=0; j<N; ++j) {
                float temp;
                inFile>>temp;
                if (i==j)
                    Aii.push_back(temp);
                else if (temp!=0) {\\для экономии памяти нулевые
                    \\ недиагональные элементы не используем
                    el temp_el;
                    temp_el.aij=temp;
                    temp_el.j=j;
                    temp_vect.push_back(temp_el);
                }
            }
        }
    }
```

```
    }
    Aij.push_back(temp_vect);
}

// Вывод на экран считанных данных
cout << "Err=" << Err<<"\n";
cout << "N=" << N << "\n";
cout << "Aii=( ";
for ( int i=0;i<N;++i)
cout << Aii[i]<<" ";
cout << ")\n";
cout << "Aij=\n";
for ( int i=0;i<N;++i){
for (int j=0;j<Aij[i].size();++j)
cout << "("<<i<<","<<Aij[i][j].j<<")-("<< Aij[i][j].aij<<") ";
cout << "\n";
}
// Решение СЛАУ
vector <float> X0; // Начальное приближение
vector <float> X_new(N);
float err_new; // невязка
float sum;
err_new=5*Err;
for (int i=0; i<N; ++i)
    X0.push_back(0);
int counter=1; //счетчик итераций
while (err_new>Err) {
    for (int i=0; i<N; ++i) {
        sum=0;
        for (int j=0; j<Aij[i].size(); ++j)
            sum=sum + Aij[i][j].aij*X0[Aij[i][j].j];
        X_new[i]=(B[i]-sum)/Aii[i];
    }
    float sum_err=0;
    for (int i=0; i<N; ++i)
        sum_err=sum_err+fabs(X0[i]-X_new[i]);
    err_new=sum_err;
```

```
    cout <<"err_new="<< err_new <<"\n";//вывод текущей невязки
    for (int i=0; i<N; ++i)
        X0[i]=X_new[i];
    ++counter;
}
cout << "точность решения ="<<Err;
cout << " , получена за "<<counter<<" итераций!\n";
for (int i=0; i<N; ++i)
    cout << X_new[i] << "    ";
return 0;
}
```

В программе используется класс «el», описание которого вынесено в отдельный файл (el.h), текст которого приведен ниже:

```
class el {
public:
    int j;
    float aij;
    el(){ //constructor
        j=0;
        aij=0;
    }
};
```

Для создания файла в CodeLite необходимо правой кнопкой мыши щелкнуть на папке «src» вашего проекта во вкладке «Workspace», выбрать «Add a New File». В качестве типа файла выбрать «Header File(.h)». В поле «name» имя файла, которое должно совпадать с названием создаваемого класса, в данном примере это «el».

В качестве входного файла данная программа использует файл с именем «alal.txt», расположенный в той же папке, что и проект. Например, при использовании файла с приведенным ниже текстом:

```
$Err
1e-3
$N
4
$A
```

```
10 -1  0  0
  0 10 -1  1
  0  1 10  0
  0  0  0  1
$B
8 21 32  4
```

Решение, выведенное на экран будет следующим:

```
Err=0.001
N=4
Aii=( 10 10 10 1 )
Aij=
(0, 1) - (-1)
(1, 2) - (-1) (1, 3) - (1)
(2, 1) - (1)

err_new=10.1
err_new=0.5
err_new=0.0369998
err_new=0.000369906
точность решения =0,001 , получена за 6 итераций!
0.99998  2.00001  3.00002  4
```

Изменяя точность решения, матрицу коэффициентов и вектор свободных членов в данном файле, можно решить СЛАУ любой размерности с диагональным преобладанием итерационным методом.

Приложение Е

Реализация методов Якоби, Гаусса-Зейделя и сопряженных градиентов на C++

Программа реализует итерационные алгоритмы решения СЛАУ: два стационарных метода – Якоби и Гаусса-Зейделя и один нестационарный – метод сопряженных градиентов. Результаты решения и невязка на каждом шаге выводятся в файл для каждого метода.

Для экономии памяти хранятся только ненулевые значения коэффициентов матрицы **A**. Данные для расчета: матрица **A**, вектор свободных членов **b**, точность решения и размерность СЛАУ считываются из файла с именем «SLAU.txt». Например, при использовании файла с приведенным ниже текстом:

```
$Err
1e-5
$N
4
$A
2 -1 0 -1
0 2 -1 -1
```

```
-1 -1 2 0
0 0 0 1
$B
-4 -3 3 4
```

Решение, записанное в файл «rezalt.txt», будет следующим (текст выходного файла приведен не полностью.):

The Conjugate Gradient method

```
k= 1
err= 1.53846
X={ -1.53846 -1.15385 1.15385 1.53846 }
...
k= 50
err= 7.15256e-007
X={ 0.999996 2 3 4 }
```

The Jacobi method

```
k= 1
err= 4
X={ -2 -1.5 1.5 4 }
...
k= 31
err= 7.62939e-006
X={ 0.999991 1.99999 2.99999 4 }
```

The Gauss-Seidel method

```
k= 1
err= 4
X={ -2 -1.5 -0.25 4 }
...
k= 20
err= 6.4373e-006
X={ 0.999994 1.99999 2.99999 4 }
```

Изменяя точность решения, матрицу коэффициентов и вектор свободных членов в данном файле, можно получить решение СЛАУ любой размерности методами Якоби, Гаусса-Зейделя и сопряженных градиентов.

Ниже приведен текст программы:

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <math.h>
#include "elSLAU.h"

using namespace std;

int main(int argc, char **argv)
{
    float Err; // Погрешность
    float err; // Невязка
    float N; // Размерность СЛАУ
    vector <float> B; //Вектор свободных членов
    vector <float> Aii; // Диагональные элементы
    vector < vector <elSLAU> > Aij; // Матрица
        // недиагональных ненулевых элементов

    // Считывание данных из файла=====
    ifstream inFile("SLAU.txt");
    if (!inFile){
        cerr<<"Error! Can't open file\n";
        return -1;
    }
    string buffer;
    while(!inFile.eof()){
        inFile>>buffer;
        if (buffer=="$Err")
            inFile>>Err;
        if (buffer=="$N")
            inFile>>N;
        if (buffer=="$B")
            for (int i=0; i<N;++i){
                float temp;
```



```

        inFile>>temp;
        B.push_back(temp);
    }
    if (buffer=="$A")
        for (int i=0; i<N;++i){
            vector <elSLAU> temp1;
            for (int j=0; j<N;++j){
                float temp;
                inFile>>temp;
                if (i==j)
                    Aii.push_back(temp);
                else if (temp!=0) {
                    elSLAU temp0;
                    temp0.aij=temp;
                    temp0.j=j;
                    temp1.push_back(temp0);
                }
            }
            Aij.push_back(temp1);
        }
}

ofstream outfile("rezalt.txt");
if (!outfile){
    cerr<< "Error of the output file opening.\n";
    return -2;
}

// Решение=====
// Метод сопряженных градиентов=====
outfile<<"\n The Conjugate Gradient method \n";
vector <float> X_k_1;
vector <float> X_k;
vector <float> r_k;
vector <float> r_k_1;
vector <float> p_k;
vector <float> p_k_1;
vector <float> Ap_k;

```

```
float alfa_k, betta_k;
for (int i=0; i<N; ++i){
    X_k_1.push_back(0);
    X_k.push_back(0);
    r_k.push_back(0);
    r_k_1.push_back(0);
    p_k.push_back(0);
    p_k_1.push_back(0);
    Ap_k.push_back(0);
}

for (int i=0; i<N; ++i){
    r_k[i]=B[i]-Aii[i]*X_k[i];
    for (int j=0; j<Aij[i].size(); ++j)
        r_k[i]=r_k[i]-Aij[i][j].aij*X_k[Aij[i][j].j];
    p_k[i]=r_k[i];
}

err=5*Err;
int counter=0;
while (err>Err){
    counter=counter+1;
    // Расчет A*p_k
    for (int i=0; i<N; ++i){
        Ap_k[i]=Aii[i]*p_k[i];
        for (int j=0; j<Aij[i].size(); ++j)
            Ap_k[i]=Ap_k[i]+Aij[i][j].aij*p_k[Aij[i][j].j];
    }
    float alfa_1=0;
    float alfa_2=0;
    for (int i=0; i<N; ++i){
        alfa_1=alfa_1+r_k[i]*p_k[i];
        alfa_2=alfa_2+Ap_k[i]*p_k[i];
    }
    alfa_k=alfa_1/alfa_2;

    for (int i=0; i<N; ++i){
```

```
X_k_1[i]=X_k[i]+alfa_k*p_k[i];
r_k_1[i]=r_k[i]-alfa_k*Ap_k[i];
}

float betta_1=0;
float betta_2=0;
for (int i=0; i<N; ++i){
    betta_1=alfa_1+ r_k_1[i]*Ap_k[i];
    betta_2=alfa_2+ p_k[i]*Ap_k[i];
}
betta_k=-betta_1/betta_2;

for (int i=0; i<N; ++i)
    p_k_1[i]=r_k_1[i]+betta_k*p_k[i];

float err_min;
float err_max=0;
// Расчет невязки
for (int i=0; i<N; ++i){
    err_min=fabs(X_k[i]-X_k_1[i]);
    if (err_min>err_max)
        err_max=err_min;
}
err=err_max;

for (int i=0; i<N; ++i){
    p_k[i]=p_k_1[i];
    r_k[i]=r_k_1[i];
    X_k[i]=X_k_1[i];
}
// Запись данных в файл
outfile<<"k= "<< counter<<" \n";
outfile<<"err= "<< err<<" \n";
outfile<<"X={ ";
for (int i=0; i<N; ++i)
    outfile<< X_k[i]<<" ";
outfile<<"}\n";
```

```
}

// Метод Якоби =====
outfile<<"\n The Jacobi method \n";
vector <float> X1_k_1;
vector <float> X1_k;
float sum;
for (int i=0; i<N; ++i){
    X1_k_1.push_back(0);
    X1_k.push_back(0);
}

err=5*Err;
int counter1=0;
while (err>Err){
    counter1=counter1+1;
    for (int i=0; i<N; ++i){
        sum=0;
        for (int j=0; j<Aij[i].size(); ++j)
            sum=sum+Aij[i][j].aij*X1_k[Aij[i][j].j];
        X1_k_1[i]=(B[i]-sum)/Aii[i];
    }
    // Рачет невязки
    float err_min;
    float err_max=0;
    for (int i=0; i<N; ++i){
        err_min=fabs(X1_k[i]-X1_k_1[i]);
        if (err_min>err_max)
            err_max=err_min;
    }
    err=err_max;

    for (int i=0; i<N; ++i)
        X1_k[i]=X1_k_1[i];

    outfile<<"k= "<< counter1<<" \n";
    outfile<<"err= "<< err<<" \n";
```

```
    outfile<<"X={ ";
    for (int i=0; i<N; ++i)
        outfile<< X1_k[i]<<" ";
    outfile<<"}\n";
}

// Метод Гаусса - Зейделя =====
outfile<<"\n The Gauss-Seidel method\n";
vector <float> X2_k_1;
vector <float> X2_k;
float sum2;

for (int i=0; i<N; ++i){
    X2_k_1.push_back(0);
    X2_k.push_back(0);
}

err=5*Err;
int counter2=0;
while (err>Err){
    counter2=counter2+1;
    for (int i=0; i<N; ++i){
        sum2=0;
        for (int j=0; j<Aij[i].size(); ++j)
            sum2=sum2+Aij[i][j].aij*X2_k_1[Aij[i][j].j];
        X2_k_1[i]=(B[i]-sum2)/Aii[i];
    }
    // Рачет невязки
    float err_min;
    float err_max=0;
    for (int i=0; i<N; ++i){
        err_min=fabs(X2_k[i]-X2_k_1[i]);
        if (err_min>err_max)
            err_max=err_min;
    }
    err=err_max;
```

```
        for (int i=0; i<N; ++i)
            X2_k[i]=X2_k_1[i];

        outfile<<"k= "<< counter2<<" \n";
        outfile<<"err= "<< err<<" \n";
        outfile<<"X={ ";
        for (int i=0; i<N; ++i)
            outfile<< X2_k[i]<<" ";
        outfile<<"}\n";
    }
    return 0;
}
```

В программе используется класс «elSLAU», описание вынесено в отдельный файл (elSLAU.h), текст которого приведен ниже:

```
class elSLAU{
public:
    float aij;
    int j;
    elSLAU(){ //constructor
        j=0;
        aij=0;
    }
};
```

Предметный указатель

- ∇ , 3, 190
 ∇^2 , 190
 \triangle , 190
 Форматы чисел в Matlab, 154
 Итерация неподвижной точки, 74
 Критерий сходимости итерации
 неподвижной точки, 76
 Критерий сходимости метода Ньютона – Рафсона, 84
 Метод Ньютона-Рафсона, 75
 Оператор двоеточие :, 156
 Оператор присваивания, 153
 Перенос строки в Matlab, 153
 СЛАУ, 49
 Вектора и матрицы в Matlab, 154
 Ввод и вывод данных в Matlab, 153
 антиградиент, 72
 диагональное доминирование, 67
 дивергенция, 190
 эллиптические уравнения, 2, 3
 гиперболические уравнения, 2, 12
 градиент, 72, 190
 граничные условия, 14, 15
 граничные условия I рода, 16
 граничные условия II рода, 16
 граничные условия III рода, 16
 итерации Якоби, 58
 итерационные методы решения –
 СЛАУ, 49
 итерация Гаусса – Зейделя, 60
 краевая задача, 15
 критерий сходимости, 67
 матричный метод решения СЛАУ, 50
 метод CG, 71, 204
 метод LU-разложения, 53
 метод Гаусса-Зейделя, 56, 60, 204
 метод Ньютона – Рафсона, 81
 метод Якоби, 56, 58, 199, 204
 метод исключения Гаусса, 52
 метод сопряженных градиентов, 71, 204
 метод верхней релаксации, 57, 70
 методы одновременного решения, 74
 методы последовательного решения, 74
 методы прямой и обратной подстановки, 50
 начальные условия, 14, 18
 нестационарные итерационные методы, 71
 нижняя треугольная матрица, 51
 нормировка, 19
 оператор Наббля, 3, 190
 параболические уравнения, 2, 10
 погрешность, 87

- погрешность абсолютная, 88
 погрешность метода, 87
 погрешность неустраняемая, 87
 погрешность относительная, 88
 погрешность вычислительная, 87
 прямые методы решения СЛАУ, 49
 прямоугольные координатные сетки, 24
 разбиение Дирихле, 27
 системы нелинейных алгебраических уравнений, 74
 скалярное произведение, 189
 смешанная задача, 15
 стационарные итерационные методы, 57
 стационарные методы решения – СЛАУ, 56
 триангуляция Делоне, 27
 триангулярные координатные сетки, 24, 26
 уравнение Лапласа, 8
 уравнение Пуассона, 3, 90
 уравнение теплопроводности, 10, 117
 уравнение волновое, 133
 векторное произведение, 189
 верхняя треугольная матрица, 50
 волновое уравнение, 12
 ячейка Дирихле, 26
 задача Дирихле, 16
 задача Коши, 15, 18
 задача Неймана, 16
 закон Стефана – Больцмана, 17
 ans, 154
 clabel, 181
 clf, 177
 colorbar, 185
 contour, 178
 contour3, 180
 div, 5, 190
 else, 158
 elseif, 158
 for, 159
 grad, 5, 190
 gradient, 164
 grid, 176
 gtext, 182
 hold, 176
 if, 158
 Inf, 154
 legend, 183
 linspace, 157
 logspace, 158
 max, 162
 mesh, 171
 meshc, 171
 meshgrid, 158
 meshz, 171
 min, 163
 NaN, 154
 ones, 157
 Pi, 154
 plot, 166
 plot3, 170

quiver, 185

size, 157

slice, 186

subplot, 177

surf, 172

surfc, 172

surfl, 174

text, 182

title, 181

while, 160

xlabel, 181

ylabel, 181

zeros, 157

zlabel, 181

Библиографический список

1. Рындин Е.А. Методы решения задач математической физики. Таганрог: Изд-во ТРТУ, 2003. — 120 с.
2. Самарский А.А., Андреев В.Б. Разностные методы для эллиптических уравнений. — М.: Наука, 1976. — 352 с.
3. Мэтьюз Д.Г., Финк К.Д. Численные методы. Использование MATLAB. — 3-е издание / Пер. с англ. — М.: Изд. дом «Вильямс», 2001. 720 с.
4. Абрамов И.И., Харитонов В.В. Численное моделирование элементов интегральных схем / Под ред. А.Г. Шашкова. — Минск.: Выш. шк., 1990. — 224 с.
5. Бубенников А.Н., Садовников А.Д. Физико-технологическое проектирование биполярных элементов кремниевых БИС. — М.: Радио и связь, 1991. — 288 с.
6. Данилина Н.И., Дубровская Н.С., Кваша О.П., Смирнов Г.Л., Феклистов Г.И. Численные методы. — М.: Высш. школа, 1976. — 368 с.
7. Моделирование полупроводниковых приборов и технологических процессов. Последние достижения: Пер. с англ. / Под ред. Д. Миллера. — М.: Радио и связь, 1989. — 280 с.
8. Патанкар С. Численные методы решения задач теплообмена и динамики жидкости / Пер. с англ. — М.: Энергоатомиздат, 1984. — 152 с.

9. Бронштейн И.Н., Семендяев К.А. Справочник по математике для инженеров и учащихся втузов. — 13-е изд., исправленное. — М.: Наука, Гл. ред. физ.-мат. лит., 1986. — 544 с.
10. Зализняк В.Е. Основы вычислительной физики. Часть 1. Введение в конечно-разностные методы. — М.: Техносфера, 2008. — 224 с.
11. Яншин А.А. Теоретические основы конструирования, технологии и надежности ЭВА: учеб. пособие для вузов. — М.: Радио и связь, 1983. — 312 с.
12. Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June M. Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, Henk Van der Vorst. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. (<http://www.siam.org/books>).
13. А.Г.Трифонов. Постановка задачи оптимизации и численные методы ее решения. (<http://matlab.exponenta.ru/optimiz/index.php>)
14. C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering, Volume 79, Issue 11, pages 1309-1331, 2009.
15. Бахвалов Н.С. Численные методы / Н.С. Бахвалов, Н.П. Жидков, Г.М. Кобельков. — 7-е изд. — М.: БИНОМ. Лаборатория знаний, 2013. — 636 с.
16. Скворцов А.В. Обзор алгоритмов построения триангуляции Делоне // Вычислительные методы и программирование. — 2002. — Т.3. — С. 14 – 39.
17. Потемкин В.Г. Система инженерных и научных расчетов MATLAB 5.X. В 2-х томах. Т. 1. — М.: ДИАЛОГ-МИФИ, 1999. 366 с.
18. Потемкин В.Г. Система инженерных и научных расчетов MATLAB 5.X. В 2-х томах. Т. 2. — М.: ДИАЛОГ-МИФИ, 1999. — 304 с.