П. Н. Вабищевич

ЧИСЛЕННЫЕ МЕТОДЫ

Вычислительный практикум

URSS
МОСКВА

ББК 22.19

Вабищевич Петр Николаевич

Численные методы: Вычислительный практикум. — М.: Книжный дом «ЛИБРОКОМ», 2010. — 320 с.

Настоящая книга посвящена отработке навыков практического применения численных методов при использовании алгоритмического языка *Python*. Рассматриваются прямые и итерационные методы линейной алгебры, спектральные задачи, системы нелинейных уравнений, задачи минимизации функций, задачи интерполирования функций, численного интегрирования, интегральные уравнения, краевые задачи и задачи с начальными данными для обыкновенных уравнений и уравнений с частными производными. На уровне пользователя применяются специализированные математические пакеты, на уровне разработчика проводится программирование базовых алгоритмов численного анализа. Часть материала посвящена информации о программном обеспечении, кратко описаны основные элементы языка *Python* и используемые пакеты.

Книга рассчитана на студентов вузов, обучающихся по специальности «Прикладная математика», и специалистов по вычислительной математике и математическому моделированию.

Издательство «Книжный дом "ЛИБРОКОМ"».
117312, Москва, пр-т Шестидесятилетия Октября, 9.
Формат 60×90/16. Печ. л. 20. Зак. № 3625.

Отпечатано в ООО «ЛЕНАНД».
117312, Москва, пр-т Шестидесятилетия Октября, 11А, стр. 11.

ISBN 978–5–397–01372–7

© Книжный дом «ЛИБРОКОМ», 2010

НАУЧНАЯ И УЧЕБНАЯ ЛИТЕРАТУРА
E-mail: URSS@URSS.ru
Каталог изданий в Интернете:
http://URSS.ru
Тел./факс: 7 (499) 135–42–16
URSS Тел./факс: 7 (499) 135–42–46

8529 ID 113221

Все права защищены. Никакая часть настоящей книги не может быть воспроизведена или передана в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, а также размещение в Интернете, если на то нет письменного разрешения владельца.

Содержание

Предисловие

Современные научные вычисления проводятся на основе использования численных методов. Вычислительные методы являются интеллектуальным ядром прикладного математического моделирования, которое базируется прежде всего на решении нелинейных нестационарных многомерных задач для уравнений с частными производными. Это обуславливает возрастающее внимание к подготовке специалистов по численным методам как на уровне разработчика, так и на уровне квалифицированного пользователя.

В курсах по численным методам основное внимание уделяется численным методам решения задач алгебры и анализа, рассматриваются вопросы решения краевых задач для обыкновенных дифференциальных уравнений и уравнений с частными производными. В вычислительной математике изучаются важнейшие вопросы построения и теоретического обоснования вычислительных алгоритмов. Не менее важной является проблема практического использования численных методов при решении прикладных задач.

Поддержка курса по численным методам проводится как в теоретическом, так и в практическом плане. Аудиторные и самостоятельные занятия направлены, с одной стороны, на закрепление базового материала по теории. Здесь отрабатываются навыки построения вычислительных алгоритмов для решения базовых задач численного анализа, теоретического исследования свойств алгоритма (точность, устойчивость, вычислительная работа при реализации и т.д.). С другой стороны, навыки грамотного практического использования численных методов закладываются в вычислительном практикуме.

Высокая техническая оснащенность, рост возможностей вычислительной техники позволяют существенно обогатить содержание вычислительного практикума по численным методам. На уровне разработчика вычислительные алгоритмы отрабатываются на основе разработки программного обеспечения для приближенного решения типовых задач. Этим обеспечивается достижение первой цели вычислительного практикума. Вторая цель, которая связана с грамотным использованием современного программного обеспечения по численному анализу, решается на уровне пользователя.

Предлагаемое учебное пособие ориентировано на практическое закрепление слушателями теоретического материала по курсу численных методов. Для

основных задач численного анализа рассматриваются вопросы построения и практической реализации вычислительных алгоритмов, использования библиотек численного анализа. Рассмотрены прямые и итерационные методы линейной алгебры, задачи интерполирования и приближения функций, численного интегрирования, спектральные задачи линейной алгебры, системы нелинейных уравнений, задачи минимизации функций, интегральные уравнения, краевые задачи и задачи с начальными данными для обыкновенных уравнений, стационарные и нестационарные задачи математической физики.

Программная реализация решения задач вычислительной математики базируется на относительно новом алгоритмическом активно развиваемом языке Python. Этот высокоуровневый язык программирования общего назначения максимально ориентирован на производительность разработчика и читаемость кода, поддерживается большинством используемых платформ и распространяется свободно.

Предлагаемая книга построена по следующему плану. Первая часть посвящена программному обеспечению, используемым средам разработки и основам языка Python. Отдельно рассмотрены базовые численные и графические пакеты. Вторая часть книги посвящена численному решению основных задач численного анализа. Каждая глава содержит справочный материал по алгоритмам, приведена программная реализация, проводится решение типовых задач.

Автор с благодарностью примет любые конструктивные замечания по книге.

П.Н.Вабищевич

Москва, январь 2010 г.

Программное обеспечение

Вычислительный практикум подразумевает организацию рабочего места. Будем считать, что вычисления выполняются на персональном компьютере с операционной системой Windows. Используемое нами программное обеспечение (интерпретатор языка и среда разработки) относится к классу кроссплатформенного, и поэтому можно работать и в операционной системе Linux.

Кратко обсуждаются вопросы установки Python на компьютере и его пакетов. Процесс разработки программ иллюстрируется использованием простейшей интегрированной среды разработки (IDE, Integrated development environment) IDLE, которая поставляется вместе с Python. Более комфортная работа обеспечивается использованием NetBeans.

1.1 Установка Python

Основные реализации, получение дистрибутива и установка, переменные окружения Python, установка пакетов.

Основные реализации

Существуют три различные реализации интерпретатора языка программирования Python: CPython, Jython и IronPython[1].

CPython[2] – это стандартная реализация, которая написана на переносимом языке ANSI C. Именно с этой эталонной реализацией интерпретатора языка мы и будем работать.

Jython[3] обеспечивает интеграцию с языком программирования Java. Реализация Jython состоит из Java-классов, которые выполняют компиляцию программного кода на языке Python в байт-код Java, а в качестве среды исполнения используется виртуальная машина Java (JVM, Java Virtual Machine).

[1] Python Implementations — www.python.org/dev/implementations/
[2] CPython — www.python.org/
[3] Jython — www.jython.org/

IronPython[4] предназначена для обеспечения интеграции программ Python с приложениями, созданными для работы в среде Microsoft.NET Framework (для операционной системы Windows и в Mono для Linux). Компилирует Python программы в промежуточный язык MSIL (Microsoft Intermediate Language).

Получение дистрибутива и установка

Возможно, что на вашем компьютере уже установлен Python: ищите пункт Python x.x меню кнопки Пуск (Start) на рабочем столе вашего компьютера. Обновите вашу версию, если это необходимо.

В настоящее время поддерживаются две серии: Python 2.x и Python 3.x. Python 3.x не совместим с предыдущей серией Python 2.x. С учетом того, что основные пакеты написаны для Python 2.x большого смысла переходить на Python 3.x при его использовании в научных вычислениях пока нет.

Для загрузки Python зайдите на официальный сайт www.python.org. Со страницы Download выберите версию (загрузить) на этой странице и выберите версию Python для своей операционной системы. Для Windows дистрибутив Python распространяется в виде стандартного инсталляционного файла с расширением .msi. Стандартный каталог установки C:\Python2x (для более новых версий — C:\Python3x).

После установки в подменю Пуск (Start) | Все программы (All Programs) появляется группа Python2x (Python3x), которая обеспечивает: запуск IDLE (IDLE Python GUI), доступ к документации (Module Docs), запуск интерактивной командной строки (Python (command line)), доступ к руководствам (Python manuals) и удаление версии Python. С интерпретатором Python связываются файлы с расширением .py.

Переменные окружения Python

Для запуска интерпретатора языка Python из любой директории необходимо, чтобы путь к соответствующему исполняемому файлу python.exe был прописан в переменной окружения path или PATH. Эти путь прописываются автоматически при первичной установке интерпретатора языка Python в каталог по умолчанию. Его необходимо, например, корректировать при установке параллельно нескольких версий Python.

Доступ к переменным окружения осуществляется вызовом диалога Свойства системы (System Properties): Панель управления (Control Panel) | Система (System). На вкладке Дополнительно (Advanced) выберите Переменные среды (Environment Variables).

[4] IronPython — http://ironpython.codeplex.com/

Вторая переменная окружения PYTHONPATH задает список каталогов для поиска интерпретатором Python включаемых модулей. По умолчанию интерпретатор ищет модули, которые расположены в том же каталоге, что и выполняемый файл.

Установка пакетов

Для Python имеется большое количество разработанного программного обеспечения, которое удобно использовать при решении различных задач (список пакетов и их описание[5]). Программное обеспечение оформляется в виде модулей, которые в свою очередь могут быть собраны в пакеты. Модуль оформляется в виде отдельного файла, а пакет — в виде отдельного каталога.

Есть различные варианты установки пакетов: стандартный инсталлятор для Windows, из исходных текстов с использованием команды

```
python setup.py install
```

и Python eggs, который позволяют использовать механизм зависимостей для пакетов Python. В первых двух вариантах приходиться внимательно следить, что нужно установить дополнительно, чтобы пакет заработал.

Простейший подход связан с использованием инсталляционного файла, который скомпилирован под необходимую вам версию Python. Инсталляционные файлы берутся на сайте пакетов Python (если они есть), или с домашней страницы пакета в его описании на этом интернет-ресурсе.

1.2 Работа в IDLE

Работа в командной строке, стандартная интегрированная среда разработки, отладка в IDLE.

Работа в командной строке

Для работы с интерпретатором языка Python в интерактивном режиме можно использовать командную строку. Командная строка (Command Prompt) расположена в разделе Стандартные (Accessories), меню Все программы (All Programs). После набора команды

```
python
```

открывается окно, в котором выведена информация о версии интерпретатора и приглашение к вводу >>>. При работе в интерактивном режиме каждая введенная команда выполняться сразу же после ввода. Введем, для примера

[5] Package Index — http://pypi.python.org/pypi/

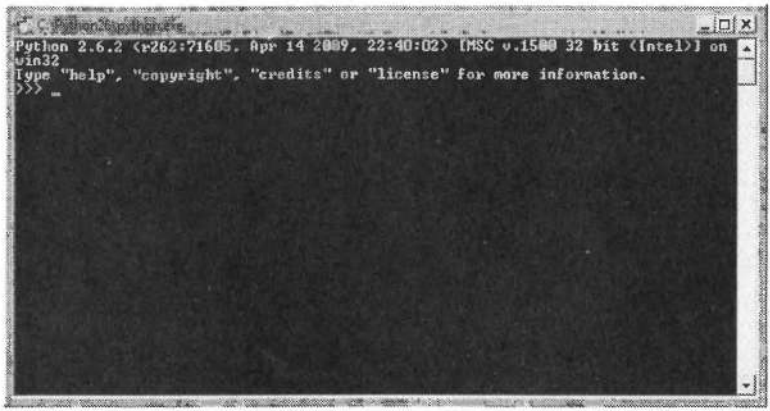

Рис. 1.1 Python в командной строке

инструкцию присваивания `a=2**10` ($a = 2^{10}$) а затем — напечатаем величину a командой `print a`. Результат будет выглядеть следующим образом:

```
>>> a=2**10
>>> print a
1024
```

При более содержательном программировании команды для интерпретатора записываются в файлы, которые называются модулями. Например, приведенные выше инструкции интерпретатора с помощью текстового редактора запишем в файл `test.py`:

```
a=2**10
print a
```

Запуск этого модуля (программы) производиться командой

```
python test.py
```

при переходе в каталог, в котором расположен файл `test.py`. В противном случае указывается полный путь к файлу.

В операционной системе Windows стандартной является процедура открытия файла щелчком мыши на его ярлыке. В этом случае после появления консольного окна и выполнения программы окно закрывается. Если вам необходимо увидеть результаты добавьте в конце модуля вызов встроенной функции `raw_input()` командой

```
raw_input()
```

в конце модуля. Выполнение программы приостанавливается пока не будет нажата клавиша `Enter`.

Стандартная интегрированная среда разработки

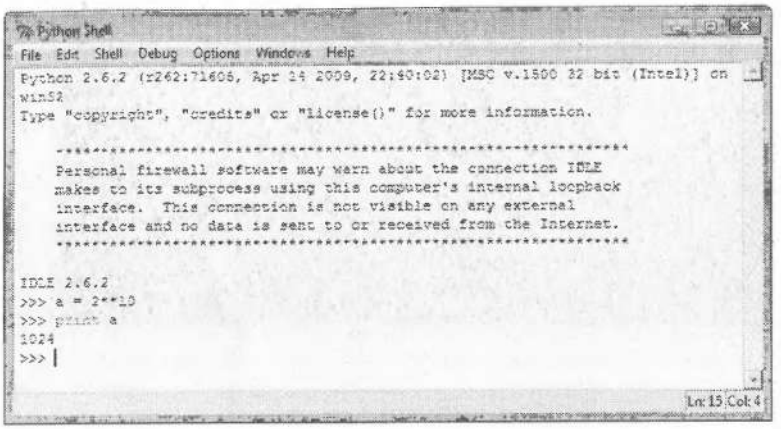

Рис. 1.2 Главное окно IDLE

Выше продемонстрирована возможность подготовки модулей в текстовом редакторе при запуске интерпретатора Python. Для более комфортной работы используются интегрированные среды разработки (IDE). Они позволяют решать разнообразные задачи (подготовка модулей, отладка и расчеты) в единой оболочке с графическим интерфейсом пользователя (GUI, Graphical user interface).

Простейшей интегрированной средой для Python является IDLE, которая является стандартной и свободно распространяемой частью дистрибутива Python. Запуск IDLE выполняется по ярлыку IDLE (Python GUI) в разделе Python x.x меню Пуск (Start) | Все программы (All programs). Быстрый старт обеспечивается контекстным меню (правая кнопка мыши) на ярлыке модуля Python (файла с расширением .py).

Главное окно (рис. 1.2) обеспечивает работу с Python в интерактивном режиме. В IDLE имеются стандартные функции текстового редактора. В частности, обеспечивается подсветка синтаксиса кода, которая позволяет визуально выделять ключевые элементы.

Поддерживается работа с модулями (текстовыми файлами с расширением .py). Для открытия и редактирования файла с программным кодом в IDLE откройте окно текстового редактора: в меню File главного окна Python Shell выберите Open. Результат для нашего тестового модуля показан на рис. 1.3. Для запуска модуля из редактора IDLE, выберите пункт Run Module в меню Run. Вывод результатов работы модуля, в частности, сообщения об ошибках, производится в основное окно Python Shell. Для создания нового модуля используйте команду File | New Window или в редакторе, или же в Python Shell.

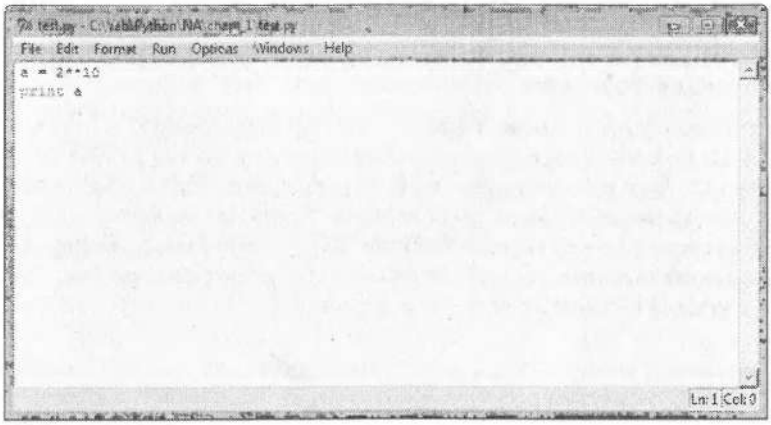

Рис. 1.3 Редактор IDLE

Отладка в IDLE

Среди основных свойств IDLE необходимо отметить возможность отладки. В отлаживаемом модуле в окне редактирования на выбранным строках кода устанавливаются точки останова по контекстному меню (правая кнопка мыши). Встроенный отладчик вызывается командой меню Debug | Debugger главного окна. После появления окна Debug Control запускается отлаживаемый модуль (команда Run | Run Module в редакторе), при этом появляется возможность пошагового выполнения программы с просмотром текущих значений переменных.

1.3 NetBeans IDE для Python

Не только IDLE, установка NetBeans, создание проекта, редактирование и отладка в NetBeans, плагины в NetBeans.

Не только IDLE

Помимо IDLE имеется много программных продуктов (редакторов, интегрированных сред разработки), которые решают задачу облегчения работы с Python. Помимо обычных возможностей, таких как подсветка синтаксиса (выделение ключевых слов, строк, комментариев), редактирование текста, отладка, они позволяют работать с файлами проектов, обеспечивают интеграцию с системами контроля версий исходных текстов и т.д.

Различные редакторы и IDE для Python для основных операционных систем, описание их базовых возможностей, сравнение можно найти на странице

PythonEditors[6]. Среди бесплатных продуктов можно выделить eric4 IDE[7]. Для программирования на языке Python в операционной системе Windows можно рекомендовать PyScripter[8].

Необходимо отметить также Eclipse[9] — кроссплатформенную интегрированную среду разработки программного обеспечения на многих языках программирования. При использовании модуля расширения PyDev обеспечивается возможность разработки программ на языке Python. К подобному классу программных продуктов относится NetBeans IDE[10] — свободная интегрированная среда разработки приложений на языках программирования Java, JavaFX, Ruby, Python, PHP, JavaScript, C++ и других.

Установка NetBeans

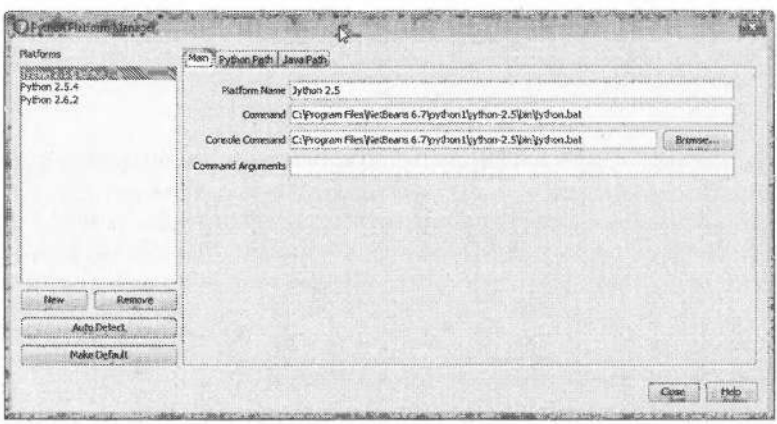

Рис. 1.4 Окно диспетчера платформ Python

Хотя проект NetBeans IDE поддерживается и спонсируется фирмой Sun Microsystems, разработка NetBeans ведется независимо сообществом разработчиков-энтузиастов (NetBeans Community) и компанией NetBeans Org[11]. NetBeans IDE доступна в виде готовых дистрибутивов (прекомпилированных бинарных файлов) для основных платформ (Microsoft Windows, GNU/Linux, FreeBSD, Mac OS X, Solaris). Загрузите NetBeans для Python (Python (Early Access 2)[12]).

[6] PythonEditors — http://wiki.python.org/moin/PythonEditors
[7] eric4 IDE — http://eric-ide.python-projects.org/
[8] PyScripter — http://code.google.com/p/pyscripter/
[9] Eclipse — www.eclipse.org/
[10] NetBeans IDE — www.netbeans.org/
[11] NetBeans Org — www.netbeans.org/index.html
[12] Python (Early Access 2) — www.netbeans.org/features/python/index.html

Для разработки программ в среде NetBeans и для успешной инсталляции и работы самой среды NetBeans должен быть предварительно установлен Sun JDK (Java Development Kit) — бесплатно распространяемый фирмой Sun комплект разработчика приложений на языке Java. Вы всегда можете загрузить последнюю версию JDK[13].

Инсталляция и запуск программы NetBeans в Windows является стандартной и осуществляется из группы NetBeans в меню Пуск (Start) | Все программы (All programs). После установки NetBeans IDE распознает установленные на компьютере версии Python. На рис. 1.4) представлено окно диспетчера платформ Python (Python Platform Menager), которое открывается по команде Tools | Python Platforms. Реализация Jython интерпретатора языка Python включена в NetBeans. В этом окне можно добавить или удалить версии Python, доступные для среды NetBeans IDE. С помощью диспетчера платформ Python можно также настроить пути к Python. В процессе настройки можно разместить модули Python в пути таким образом, чтобы они были доступны при каждом запуске NetBeans для каждой отдельной версии Python.

Создание проекта

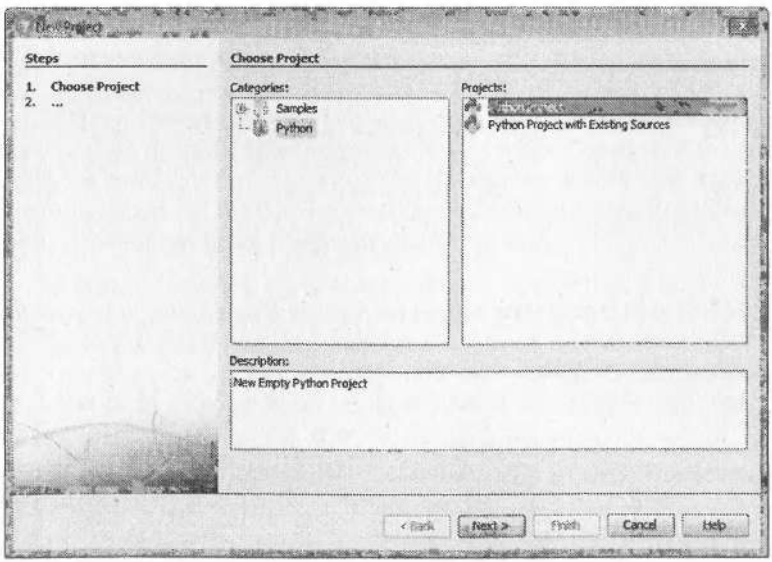

Рис. 1.5 Мастер New Project

Программа на языке Python состоит из набора модулей, которые обычно связаны с файлами. Модули позволяют объединять разрозненные файлы в круп-

[13] Java SE Downloads — http://java.sun.com/javase/downloads

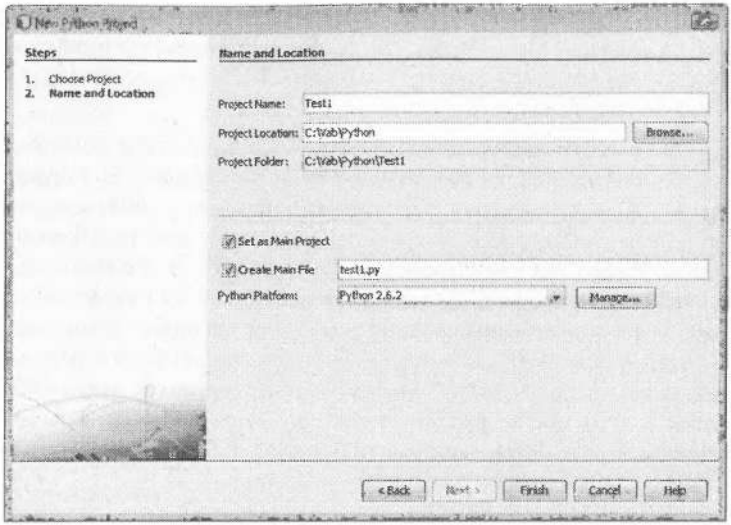

Рис. 1.6 Свойства нового проекта

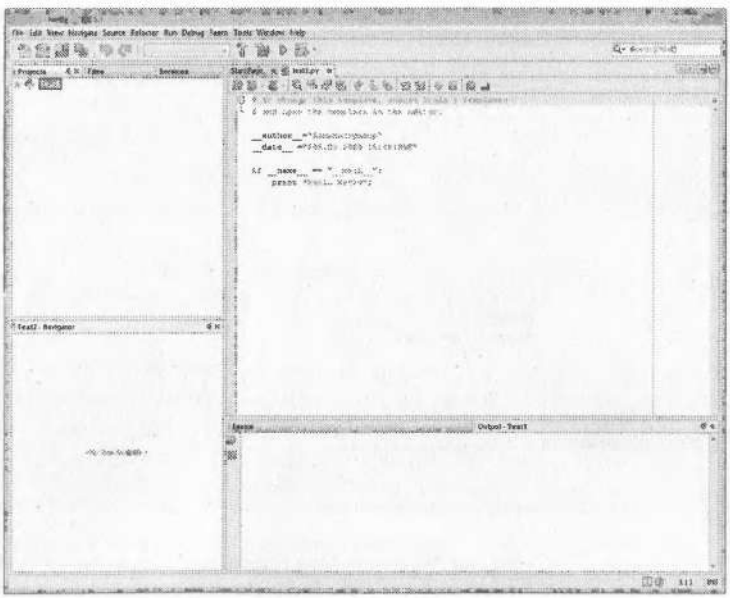

Рис. 1.7 Основное окно NetBeans

ные программные системы. Программа организована как один главный файл, к которому могут подключаться дополнительные файлы (модули).

В среде разработки NetBeans программа на языке Python связывается с проектом, который состоит из каталога с файлами. Проект можно или создавать с нуля, или из уже существующих исходников.

Сначала необходимо создать новый проект Python. Для этого используется мастер New Project, который содержит встроенные шаблоны различных типов проекта. Для открытия мастера New Project выберите команду File | New Project на главной панели инструментов или по контекстному меню на окне Projects (рис. 1.5).

Выберите значение Python Project в качестве типа проекта, если новый проект не использует ранее подготовленные модули, и нажмите кнопку Next. В поле имени проекта введем Test1 и выберем каталог, в котором этот проект будет располагаться. Выберите нужную нам версию Python из раскрывающегося списка Python Platform. Флажок Set as Main Project делает проект активным в списке проектов, а флажок Create Main File определяет файл проекта test1.py как главный (рис. 1.6). Результат работы мастера New Project показан рис. 1.7.

Редактирование и отладка в NetBeans

Редактор NetBeans для файлов Python поддерживает базовые функции завершение кода, подсветка синтаксиса и ошибок компиляции. Когда вы набираете код, вы можете открыть окно завершения кода (Code Completion) (нажатием комбинации клавиш Ctrl + Space), которое содержит список для завершения текущего выражения. Если далее продолжать набирать код, то список будет изменяться в соответствии с введенными символами.

Для некоторых наиболее часто используемых фрагментов кода можно использовать сокращения (аббревиатуры) взамен набора полного фрагмента — шаблоны кода (Code Templates). NetBeans преобразует аббревиатуры в полные фрагменты кода после нажатия на клавишу табуляции. Список шаблонов кода доступен по команде Tools | Options | Editor на вкладке Code Templates. Здесь вы можете создать свой собственный шаблон кода.

Еще одна возможность связана с подсказками (Editor Hints) Когда NetBeans обнаруживает ошибку, такую как недостающий код, то в месте ее предполагаемого возникновения появляется на левой границе значок лампочки. Вы можете кликнуть на этот значок или нажать комбинацию клавиш Alt + Enter для отображения возможных исправлений. Если одно из исправлений подходит вам, то вы можете выделить его и нажать Enter для исправления ошибки в вашем коде.

Упомянем также о возможностях рефакторинга - последовательность мелких изменений в коде, результат которых не изменяет поведение программы.

В результате код становиться проще для понимания и изменения. В NetBeans имеется возможность переименовывать классы, методы и переменные и некоторые другие возможности. Для переименования выделенной переменной воспользуйтесь командой Refactor | Rename. Перед применением рефакторинга показывается сравнение файлов до него и после.

Среда NetBeans содержит мощный отладчик Python, который позволяет отлаживать приложение. В зависимости от размера, сложности и области действия приложения отладчик Python предоставляет различные способы отладки проекта. Для простых приложений многие из этих функций не требуются. Параметры отладки настраиваются в окне Options, после команды Tools | Options | Python на главной панели инструментов и выбора вкладки Debugger. Запускается отладчик Python командой Debug | Debug Main Project. В простейших случаях будет достаточно сообщений об ошибках интерпретатора языка Python (в окне Output).

Плагины в NetBeans

Для работы в NetBeans IDE используется большое количество плагинов. Информация о доступных и используемых плагинах имеется на окне Plugins, которое открывается командой Tools | Plugins. В частности, по умолчанию устанавливаются установлены плагины для трех систем контроля версий: SVN, CVS и Mercurial. Поэтому на вкладке Installed можно что-то из этого отключить (Deactivate) или даже удалить (Uninstall).

Обратите внимание на список всех плагинов для NetBeans[14], характеристику их основных возможностей. Поддерживается система поиска плагинов по ключевым словам.

NetBeans IDE помимо редактора кода для языка Python дает возможность просматривать графические файлы, что полезно использовать при организации вычислений. Эти файлы с результатами расчетов удобно поместить в каталог проекта и они становятся доступны для просмотра (рис. 1.8).

Для работы с документами, в частности, с файлами помощи, учебными и исследовательскими материалами полезно иметь возможность просмотра файлов в формате PDF (Portable Document Format). Поиск по порталу плагинов (ключевое слово PDF) даст нам несколько вариантов. Например, для просмотра PDF файлов можно использовать NetBeans-PDFViewer[15]. С сайта проекта загружается файл с расширением .nbm. На вкладке Downloaded окна Pluguns нажимаем кнопку Add Plugins... и выбираем загруженный файл. После этого инсталлируем его (кнопка Install). Пример использования плагина представлен на рис. 1.9.

[14] NetBeans Plugin Portal — http://plugins.netbeans.org/PluginPortal/
[15] NetBeans-PDFViewer — http://code.google.com/p/netbeans-pdfviewer/

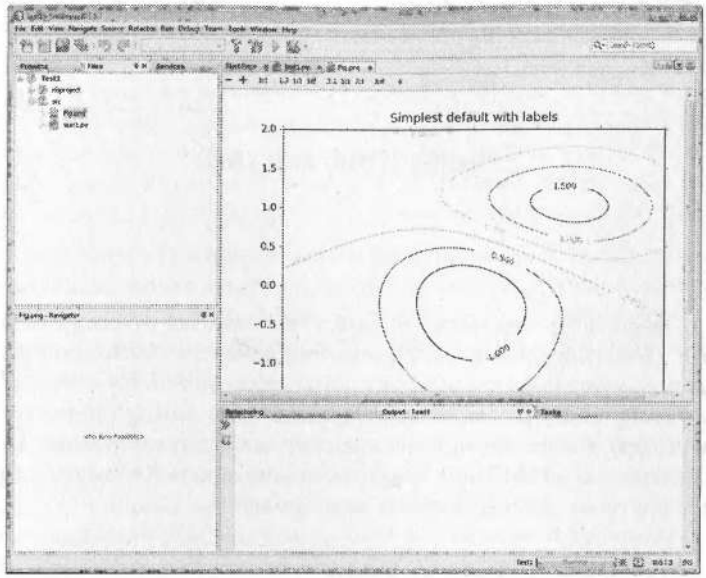

Рис. 1.8 Просмотр графического файла в NetBeans

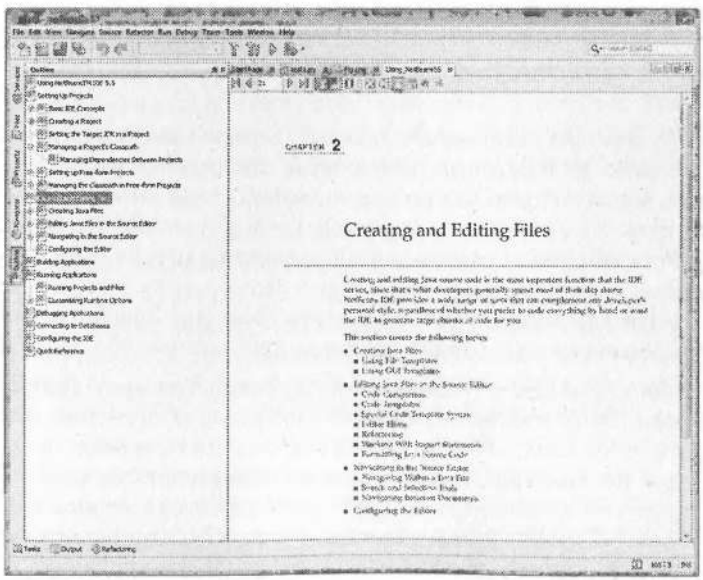

Рис. 1.9 Просмотр PDF файла

2

Элементы языка

Эта глава никак не претендует на полное и всестороннее руководство по языку Python. Основная цель состоит в том, чтобы предоставить минимальную информацию, которая достаточна для того, чтобы начать работать с Python. Особенно это будет нетрудно, если Вы знакомы с другими алгоритмическими языками. После общего обсуждения дается краткая характеристика базовых элементов языка с ориентацией на пользователя, который намерен использовать Python в своих вычислительных экспериментах.

2.1 Общая характеристика языка Python

Язык Python, объекты Python, динамическая типизация.

Язык Python

Python — это активно развиваемый, мощный, переносимый, простой в использовании и свободно распространяемый язык программирования. Он ориентирован на высокую производительность разработчика и максимальную читаемость кода. Не предлагая революционных особенностей и нововведений, язык Python комбинирует основные лучшие качества многих различных языков программирования. Разработка языка Python была начата в конце 1980-х годов голландским математиком Гвидо ван Россумом (Guido van Rossum), работа над языком активно поддерживается сообществом пользователей.

Несомненное преимущество языка Python состоит в простоте, удобочитаемости программ. Это достигается жесткими правилами оформления программного кода (см. Python Style Guide[1]), что облегчает его понимание. Небольшое число ясных базовых концепций (философия программирования на языке Python[2]) делает язык простым в освоении и использовании. Возможность диалогового режима работы интерпретатора Python сокращает время изучения самого языка и облегчает переход к решению поставленных задач.

[1] Python Style Guide — www.python.org/doc/essays/styleguide.html
[2] The Zen of Python — www.python.org/dev/peps/pep-0020/

Python ориентирован на быструю разработку приложений (RAD, Rapid Application Development). Создание программ за меньшее время достигается использованием встроенных высокоуровневых структур данных, динамической типизацией и простым синтаксисом. Программирование на языке Python значительно увеличивает производительность труда разработчика по сравнению с традиционными языками программирования, такими как C, C++ и Java. Меньший объем программного кода на Python дает значительный выигрыш времени на написание, отладку и сопровождение программного продукта.

Python относится у категории открытых программных продуктов. Вы можете получить полные исходные тексты реализации Python и использовать без всяких ограничений (копировать, распространять, встраивать в другие продукты). Поддержка обеспечивается через Интернет квалифицированными экспертами всего мира.

Python доступен для многих платформ, а написанные на нем программы переносимы между платформами без каких-либо изменений. Стандартная реализация языка Python написана на переносимом ANSI C, благодаря чему он компилируется и работает практически на всех основных платформах. Программы на языке Python, которые используют базовые возможности языка и стандартные библиотеки, компилируются в переносимый байт-код и одинаково работают в Windows и в Linux, а также в любых других операционных системах, где установлен Python.

Стандартная библиотека Python предоставляет широкие возможности, которые используются в прикладных программах. Она включает модули для работы с текстом, мультимедийными форматами, архивами, сетевыми протоколами и форматами Интернета, обеспечивается поддержка юнит-тестирования и др. В Python существует множество прикладных библиотек для решения самых разнообразных задач (Web-приложения, базы данных, графические библиотеки и т. д.). Среди них для нас наибольший интерес вызывают библиотеки численных методов, которые по своей функциональности позволяют заменить MATLAB.

Функциональные возможности разрабатываемых программных продуктов расширяются за счет интеграции отдельных компонентов. Можно как встраивать (embedding) интерпретатор Python в программу на другом языке, так и, наоборот, писать модули для Python на других языках (extending). Стандартная библиотека позволяет программам Python напрямую обращаться к динамическим библиотекам (DLL, Dynamic-link library), написанным на C. Существуют модули, позволяющие встраивать код на C/C++ прямо в исходные файлы Python, программный интерфейс для написания собственных модулей на других языках. Такие модули расширения позволяют объединить эффективность кода на C/C++ с удобством и гибкостью интерпретатора Python. На этом пути нивелируется недостатки языка Python, которые связаны, прежде всего, со скоростью выполнения программ, которая не всегда может быть та-

кой же высокой, как у программ, написанных на языках программирования низкого уровня, таких как C или C++.

Объекты Python

Не забираясь в терминологические дебри объектно-ориентированного программирования, можно сказать, что все данные в языке Python являются объектами, над которыми выполняются те или иные действия. С отдельным объектом связывается область памяти со значениями и ассоциированными с ними наборами операций.

Программа на языке Python можно разложить на такие основные составляющие, как модули, инструкции, выражения и объекты. Иерархическое построение основано на том, что:

- программы состоят из модулей;
- модули содержат инструкции;
- инструкции состоят из выражений, которые создают и обрабатывают объекты.

Сами объекты могут быть встроенными, когда они предоставляются языком Python, а также объектами, которые создаются с помощью других инструментов, например, библиотек расширений, написанных на языке C. Объекты могут быть неизменчивыми и изменчивыми. Например, строки в языке Python являются неизменчивыми и в силу этого операции над строками создают новые строки.

Python предоставляет мощную коллекцию объектных типов, встроенных непосредственно в язык, которые решают многие типовые задачи программирования. Такие встроенные объекты, как коллекции (списки) и таблицы поиска (словари) можно использовать непосредственно. Среди встроенных типов языка Python как базовые можно отметить числа (целые, числа с плавающей точкой, комплексные числа), строки, списки, словари, кортежи и файлы.

Динамическая типизация

Язык Python относится к классу языков программирования с динамической типизацией, при которой переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной, как при статической типизации. Типы данных определяется автоматически, и их не требуется явно объявлять в программном коде В различных участках программы одна и та же переменная может принимать значения разных типов.

В языке Python отсутствует конструкция объявления типа, сам синтаксис выполняемых выражений задаст типы создаваемых и используемых объектов. После создания объекта, он будет ассоциирован со своим собственным

набором операций на протяжении всего времени существования — над объектом можно выполнять только те операции, которые применимы к его типу (строгая типизация).

При динамической типизации тип переменной определяется только во время исполнения. Переменные не имеют никакой информации о типе или ограничениях, связанных с ним: тип является свойством объекта, а не его имени. Переменные являются всего лишь ссылками на конкретные объекты и в конкретные моменты времени: одна и та же переменная может связываться в разных частях программы с объектами разного типа. Прежде чем переменную можно будет использовать, ей должно быть присвоено значение (связать с объектом).

Объект в Python помимо типа характеризуется числом ссылок. Если на объект нет ссылок, то происходит автоматическое освобождение памяти, занимаемой объектами — сборка мусора (garbage collection).

2.2 Типы данных

Числа в Python, строки, листы, кортежи и словари, файлы.

Числа в Python

В Python имеются стандартные типы объектов чисел (целые и с плавающей точкой), которые присутствуют в других языках программирования. Python поддерживает работу с комплексными числами. Точность представления определяется точностью компилятора языка C, который использован для сборки интерпретатора Python. В Python имеются также длинные целые числа с неограниченной точностью представления (точность зависит от объема доступной памяти).

Тип int (целые числа) соответствует типу long в компиляторе C для используемой архитектуры. При работе на 32-разрядной системе максимальное целое равно $2^{31} - 1$, а минимальное равно -2^{31}.

Длинное целое число (тип long) идентифицируется символом l или L в конце числа. Интерпретатор языка Python осуществляет преобразования значения, выходящее за рамки допустимого типа int, в long. Ниже приведен пример, поясняющий сказанное.

```
>>> import sys
>>> a = sys.maxint
>>> print a
2147483647
>>> type(a)
<type 'int'>
```

```
>>> b = a + 1
>>> type(b)
<type 'long'>
```

Сначала мы находим (команда maxint) максимальное целое типа int, а затем (команда type) — определяем типы чисел a и b.

Признаком восьмеричного числа является 0 в начале числа, вслед за которым цифры 0-7. Для шестнадцатеричного числа используется 0x или 0X в начале, и затем — шестнадцатеричные цифры 0-9 и A-F. В силу этого

```
>>> print 10, 012, 0xA, 10L
10 10 10 10
```

Вещественные числа (тип float) в языке Python реализованы как тип double в языке C. Числа с плавающей точкой могут содержать символ точки и/или символ e или E для выделения экспоненты:

```
>>> 2**100
1267650600228229401496703205376L
>>> 2. ** 100
1.2676506002282293e+30
```

Комплексные числа на языке Python состоят из двух чисел с плавающей точкой, представляющих вещественную и мнимую части. Для индикации мнимой части используется символ j или J в конце. В примере

```
>>> z = 2 + 3j
>>> z.real
2.0
>>> z.imag
3.0
```

выделяется вещественная и мнимая часть комплексного числа z.

В Python имеется логический тип bool, который доступен в виде двух новых встроенных предопределенных имен True (истина) и False (ложь). True и False ведут себя точно так же, как и целые числа 1 и 0, однако при выводе на экран они выводятся как слова True и False, вместо цифр 1 и 0:

```
>>> b = 2 > 3
>>> a = b + 1
>>> print a, b
1 False
```

Для работы с числовыми объектами используются стандартные модули math (вещественные числа) и cmath (комплексные числа). О подобных расширениях Python, которые особенно важны при научных вычислениях, мы будем говорить отдельно.

Строки

Для работы с текстовой информацией в Python используются строки (тип string), которые представляют собой упорядоченные последовательности символов. Язык Python имеет мощный набор средств для обработки строк. Как мы уже отмечали выше, строки являются неизменчивыми объектами языка.

Строки задаются с использованием апострофов (') или кавычек (").

```
>>> s1 = "Sample"
>>> s2 = 'lines'
>>> print s1, s2
Sample lines
```

Для работы с произвольным числом строк используются утроенные апострофы или утроенные кавычки.

```
>>> s = """Multiline text:
... line 1
... line 2"""
>>> print s
Multiline text:
line 1
line 2
```

Символ \ используются для вставки специальных символов. В примере

```
>>> s = 'a\nb\tc'
>>> print s
a
b       c
```

символ \n обеспечивает переход на новую строку, а \t — горизонтальную табуляцию.

Базовый набор операций над строками включает объединение строк с помощью оператора конкатенации + и дублирование — оператор повтора *:

```
>>> s1 = 'Hello'
>>> s2 = 'world'
>>> s3 = '!'
>>> print s1 + ' ' + s2 + s3*3
Hello world!!!
```

Встроенная функция len возвращает длину строки:

```
>>> s = 'string'
>>> len(s)
6
```

Строка в Python представляет собой последовательность символов с произвольным доступом. Отдельные элементы этой последовательности извлекаются с помощью процедуры индексации по номеру позиции последовательности следующим образом:

```
>>> s = 'string'
>>> s[1]
't'
```

В языке Python индексация начинается с 0: первый элемент имеет индекс 0, второй — 1 и т. д. Предусмотрена также возможность индексации в обратном порядке:

```
>>> s = 'string'
>>> s[-1]
'g'
```

Для получения части символов строки можно использовать более общую форму индексирования, известную как получение среза. В примере

```
>>> s = 'string'
>>> s[1:3]
'tr'
```

выбраны символы с 2 (индекс 1) по 3. Левая граница среза по умолчанию принимается равной нулю, а правая — длине последовательности (строки):

```
>>> s = 'string'
>>> s[1:]
'tring'
```

Списки

Списки (тип list) в языке Python являются аналогом массивов в других языках программирования, но они обладают более широкими возможностями. В отличие от строк списки могут содержать объекты любых типов: числа, строки и даже другие списки. Кроме того списки являются изменяемыми объектами, списки могут увеличиваться и уменьшаться непосредственно (их длина может изменяться). Элементы списка заключены в квадратные скобки и разделены запятой:

```
>>> lst = [123, 1.23, 'adc']
>>> print lst
[123, 1.23, 'adc']
```

В этом примере каждый элемент списка принадлежит к разному типу. Списки в Python поддерживают все операции над последовательностями, которые мы упоминали при обсуждении строк, так что

```
>>> lst = [123, 1.23, 'adc']
>>> len(lst)
```

```
3
>>> lst[1:]
[1.23, 'adc']
```

Метод append увеличивает размер списка и вставляет в конец новый элемент, а pop удаляет элемент из списка:

```
>>> lst = [1, 2, 3, 4]
>>> lst.append('abc')
>>> lst
[1, 2, 3, 4, 'abc']
>>> lst.pop(2)
3
>>> lst
[1, 2, 4, 'abc']
```

Для того чтобы вставить новый элемент в нужное место списка используется метод insert, для удаления элемента, заданного значением — метод remove:

```
>>> lst = [3, 5, 2, 1]
>>> lst.insert(2, 4)
>>> lst
[3, 5, 4, 2, 1]
>>> lst.remove(5)
>>> lst
[3, 4, 2, 1]
```

Для сортировки элементов списка используются методы sort (сортировка по возрастанию) и reverse — по убыванию:

```
>>> lst = [3, 4, 2, 1]
>>> lst.sort()
>>> lst
[1, 2, 3, 4]
>>> lst.reverse()
>>> lst
[4, 3, 2, 1]
```

Работа с матрицами (многомерными массивами) в языке Python может быть обеспечена использованием вложенных списков. Пример извлечения элементов матрицы 3×3:

```
>>> A = [[1,2,3], [4,5,6], [7,8,9]]
>>> A[1]
[4, 5, 6]
>>> A[2][2]
9
```

С помощью первого индекса (`A[1]`) извлекается вторая строка, а с помощью второго (`A[2][2]`) — элемент этой строки.

Кортежи и словари

Кортежи (тип tuple) также являются последовательностями, как и списки, но в отличие от списков кортежи являются неизменяемыми. В этом смысле они похожи на строки. Для задания кортежа используются не квадратные, а круглые скобки:

```
>>> tpl = ('ab', 12, 1.2)
>>> type(tpl)
<type 'tuple'>
>>> tpl[0]
'ab'
```

При передаче коллекции объектов между компонентами программы используются кортежи вместо списков для обеспечения неизменности передаваемых данных.

Словари (тип dict) представляют собой коллекции объектов, доступ к которым осуществляется по ключам: пары ключ-значение, где значение однозначно определяется ключом. В качестве ключа может выступать неизменчивый тип данных (число, строка, кортеж). Порядок пар ключ-значение произволен. При задании словаря используются фигурные скобки, для разделения пар ключ-значение — запятая, символ : отделяет ключ от значения. Доступ к элементам словаря осуществляется по ключу:

```
>>> dct = {'name': 'Nik', 'age': 25}
>>> dct
{'age': 25, 'name': 'Nik'}
>>> dct['hobby'] = 'sport'
>>> dct
{'hobby': 'sport', 'age': 25, 'name': 'Nik'}
```

В данном примере также проиллюстрировано создание новой записи в словаре (пара `'hobby': 'sport'`).

Для словарей имеются свои специальные методы: проверка на вхождение, список ключей, список значений, копирование, получение значения по умолчанию, слияние, удаление и т. д.

Файлы

Объекты типа file обеспечивают интерфейс между программным кодом на языке Python и внешними файлами на компьютере.

Встроенная функция open создает объект файла, а close разрывает связь с внешним файлом. Первый аргумент open — строка, содержащая имя файла, второй аргумент — строка режима использования файла. Выбор 'r' (по умолчанию) соответствует открытию файла для чтения, 'w' — только для записи (файл будет перезаписан).

Пример записи и чтения из файла:

```
>>> f = open('c:\\tmp\\data.txt', 'w')
>>> f.write('line 1\nline 2\n')
>>> f.close()
>>> f = open('c:\\tmp\\data.txt')
>>> s1 = f.readline()
>>> s2 = f.readline()
>>> s1
'line 1\n'
>>> s2
'line 2\n'
```

Метод write используется для записи в файл, а readline — для построчного чтения.

2.3 Инструкции

Имена в Python, логические строки программы, стиль программирования, инструкции присваивания, инструкция print, условные инструкции, циклы while, циклы for, обработка исключений.

Имена в Python

Имена переменных должны начинаться с латинской буквы (любого регистра) или подчеркивания, а дальше допустимо использование букв, цифр, цифр или символов подчеркивания. В языке Python регистр символов имеет значение как в именах, которые вы создаете, так и в зарезервированных словах.

В качестве идентификаторов нельзя применять ключевые (зарезервированных) слова языка и нежелательно переопределять встроенные имена (табл. 2.1).

Таблица 2.1 *Ключевые слова Python*

and	as	assert	break	class	continue	def	del
elif	else	except	exec	finally	for	from	global
if	import	in	is	lambda	not	or	pass
print	raise	return	try	while	with	yield	

Список ключевых слов для установленной версии Python можно вывести следующим образом:

```
>>> import keyword
>>> keyword.kwlist
```

Дополнительные соглашения об именах касаются использования подчеркивания в начале и в конце имени. Например, имена с двумя символами подчеркивания в начале и конце являются системными.

Логические строки программы

Базовым элементом программы на языке Python является логическая строка. Программа на Python, с точки зрения интерпретатора, состоит из логических строк. Работа интерпретатора состоит в последовательной обработке логических строк. Логическая строка составляется из одной или нескольких физических строк, если подразумевается объединение строк. Физическая строка заканчивается символом конца строки, принятым для используемой платформы. Комментарий начинается символом # и заканчивается в конце физической строки.

Обычно на каждой строке располагается одна инструкция, при записи нескольких инструкций в одной строке они разделяются символом ;. Для объединения нескольких физических строк в одну логическую строку используется символ обратной косой черты (\):

```
print a; print b; c = a
sr = a + b + c \
     d + e
```

Выражения в круглых, квадратных и фигурных скобках могут быть разделены на несколько физических строк без использования символа обратной косой черты:

```
lst = ['abc',
       123,
       1.23]
```

Инструкции языка Python не могут быть разделены на несколько логических строк, за исключением составных инструкций. Составные инструкции состоят из основной инструкции и блока вложенных инструкций. Запись составной инструкции проводится по шаблону

```
Основная инструкции:
    Блок вложенных инструкций
```

с использование двоеточия в качестве разделителя.

Вложенный блок оформляется использованием одинаковых отступов от левого края. Именно по величине отступа интерпретатор языка Python определяет, где находится начало блока и где — его конец. В пределах одного блока все инструкции должны иметь один и тот же отступ от левого края.

Отступы являются частью синтаксиса языка Python. Их использование позволяет получить однородный и удобочитаемый программный код, простой в сопровождении. Подобные ограничения на оформление программы важны для любого языка программирования и оказывают огромное влияние на применимость вашего программного кода, на его пригодность к многократному использованию.

Стиль программирования

Основные ограничения на вид программы дает синтаксис языка программирования и при его нарушении интерпретатор выдает синтаксические ошибки. Под стилем программирования будем понимать дополнительные ограничения на оформление кода, которые принимается и используется некоторой группой разработчиков программного обеспечения с целью получения удобных для применения, легко читаемых и эффективных программ.

Стиль программирования касается все аспектов оформления кода: выбора названий и используемый регистр символов для имён переменных, стиль комментариев, оформление логических блоков, модулей, документирования и т.д. Для языка Python разработан официальный стиль (Python Style Guide[3]).

Отметим некоторые положения стиля программирования на языке Python. При написании кода рекомендуется:

- использовать отступы в 4 пробела;
- использовать физические строки не более 79 символов;
- логические строки разбивать неявно (внутри скобок);
- выравнивать отступы строк продолжения по скобкам или по первому операнду в предыдущей строке;
- не ставить пробелы сразу после открывающей скобки или перед закрывающей, перед запятой, точкой с запятой;
- не ставить более одного пробела вокруг знака равенства в присваиваниях (пробелы вокруг знака равенства не ставятся в случае, когда он применяется для указания значения по умолчанию).

Среди рекомендаций по написанию комментариев отметим следующие:

- обновляйте комментарии, когда модифицируете код;
- для короткого комментария лучше не ставить в конце точку, длинные лучше писать по обычным правилам написания текста;
- пишите комментарии на английском;
- комментарии к фрагменту кода (блочный комментарий) идут с тем же отступом, что и комментируемый код, после символа # идет одиночный пробел, абзацы можно отделять строкой с # на том же уровне, блочный комментарий отделяется от окружающего кода пустыми строками;

[3] Python Style Guide — www.python.org/doc/essays/styleguide.html

- встроенные комментарии относятся к конкретной строке и их не следует использовать часто, символ # должен отстоять от комментируемого оператора по крайней мере на два пробела;
- все модули, классы, функции и методы, предназначенные для использования за пределами модуля, должны иметь строки документации, описывающие способ их применения, входные и выходные параметры;
- для строк документации использовать утроенные кавычки;
- для очевидных случаев используйте одинарные строки документации;
- многострочное документирование состоит из подытоживающей строки с последующей пустой строкой и более подробным описанием;
- вставлять пустую строку между последним параграфом в многострочной документации и ее закрывающими кавычками, размещая кавычки в отдельной строке.

Третья группа соглашений касается правил для именования различных объектов, с тем чтобы это было понятно любому программисту, использующему Python:

- имена модулей лучше давать строчными буквами, либо делать первые буквы слов прописными, имена написанных на C модулей расширения обычно начинаются с подчеркивания;
- в именах классов первые буквы слов являются прописными;
- имена констант (они не должны переопределяться) лучше записывать прописными буквами;
- имена исключений содержат в своем составе слово error (или warning).

Инструкции присваивания

Инструкция присваивания всегда создает ссылку на объект и никогда не создают копии объектов. Для того, чтобы связать идентификатор (существующий или новый) с объектом, для создания и изменения атрибутов объектов, изменения элементов изменяемых последовательностей, добавления и изменения записей в отображениях используется символ `=`.

Можно использовать групповое присваивание типа

```
>>> a = b = c = 0
```

Ту же задачу минимизации кода решает компактная форма, когда, например, вместо `x = x + y` мы используем `x += y`.

Инструкция print

С ее помощью производится вывод на экран для каждого выражения в списке выражений — эта инструкция преобразует объекты в текстовое представление и посылает результат на устройство стандартного вывода.

```
>>> a = 2.
>>> b = 3
>>> print b/2, a
1 2.0
```

Инструкция `print` может быть использована для вывода в любой файл. Для этого после ключевого слова `print` необходимо поставить `>>` и соответствующий файловый объект.

```
>>> f = open('c:\\tmp\\data.txt', 'w')
>>> s = 'Text'
>>> print >> f, s
>>> f.close()
```

Этот же результат можно получить использованием обычной команды `print` при перенаправлении вывода в файл вместо стандартного потока вывода.

Условные инструкции

Условная инструкция `if` в языке Python является примером составной инструкции. Наиболее общая конструкция этой инструкции в Python имеет вид:

```
if condition1:
   body1
elif condition2:
   body2
elif condition3:
   body3
. . .
elif condition(n-1):
   body(n-1)
else:
   body(n)
```

Если условие `condition1` истинно, то выполняется `body1`; иначе, если условие `condition2` истинно, выполняется `body2` и так далее, пока не находится истинное условие, или если нет истинных условий, то выполняется `body(n)`. Если при этом нет части `else`, то условный оператор ничего выполнять не будет.

Типичный фрагмент кода с условной инструкцией:

```
if a < 0:
    b = 1
elif a == 0:
    b = 0
else:
    b = -1
```

Циклы while

Инструкция `while` является самой универсальной конструкцией организации итераций в языке Python, в которой блок инструкций выполняется до тех пор, пока условное выражение продолжает возвращать истину. Цикл с `while` выглядит следующим образом:

```
while condition:
    body
else:
    post-code
```

Здесь `condition` это условие, которое оценивается на истинность или ложность. Пока условие справедливо, тело цикла `body` будет постоянно выполняться. Если условие ложно, будет выполняться блок программы `post-code`. Если цикл начат, а условие `condition` ложно, тело цикла `body` не будет выполняться, а один раз выполниться часть цикла `post-code`.

```
s = 0.
i = 1
while i < 100:
    s = s + 1./i**2
    i = i + 1
```

Отметим две простые инструкции `break` и `continue`, которые могут использоваться только внутри циклов. Если выполняется `break`, цикл `while` немедленно прекращается, даже часть `post-code` не выполняется. При выполнении инструкции `continue` прекращается работа тела цикла `body`, условие `condition` оценивается снова и цикл начинает работать сначала. Часть `else` цикла `while` может не использоваться, если нет оператора `break` в `body`.

В тех случаях, когда синтаксис языка требует наличия инструкции, используется инструкция `pass`, которая не выполняет никаких действий.

Циклы for

Цикл `for` выполняет выполнять тело цикла для каждого элемента последовательности. Инструкция `for` работает, например, со строками, списками, кортежами. Общая форма выполнения цикла при работе со списком значений:

```
for variable in list:
    body
else:
    post-code
```

Тело `body` будет выполнено один раз для каждого элемента списка. Конструкция с `else` обычно не используется. Операторы `break` и `continue` используются также, как и в цикле `while`. Вычисление суммы чисел:

```
x = [1, 2, 3, 4, 5]
s = 0
for n in x:
    s = s + n
```

Для работы с индексами используется функция `range()`. Функция `range()` с одним аргументом генерирует список целых чисел в диапазоне от нуля до указанного в аргументе значения, не включая его. В функции `range()` с двумя аргументами, первый будет рассматриваться как нижняя граница диапазона. Необязательный третий аргумент определяет шаг в списке целых чисел, так что

```
>>> range(-3, 10,2)
[-3, -1, 1, 3, 5, 7, 9]
```

Использование функции `range` в цикле:

```
for i in range(1, 10):
    for j in range(1, 10):
        print i, j, i*j
```

В этом примере иллюстрируется работа с вложенными циклами.

Обработка исключений

Исключение возбуждается, когда интерпретатор обнаруживает ошибку во время выполнения программы. В своей программе мы можем перехватывать такие ошибки и обрабатывать их или просто игнорировать. В последнем случае интерпретатор останавливает выполнение программы и выводит сообщение об ошибке.

```
>>> a = 0.
>>> b = 2 / a
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ZeroDivisionError: float division
```

Можно добавить инструкцию `try`, которая позволит перехватить обнаруженные ошибки (деление на ноль) и продолжить выполнение программы. Стандартная форма инструкции `try`:

```
try:
    do something
except error:
    do something else
```

В нашем примере результатом работы программы

```
a = 0.
```

```
try:
    b = 2 / a
except ZeroDivisionError:
    print 'Division by zero!'
```

будет строка

```
Division by zero!
```

Более общие конструкции обработчиков исключений можно найти в документации по языку.

2.4 Функции

Инструкция `def`, переменное число аргументов, `lambda` функции, пространство имен в Python.

Инструкция def

Функции в Python являются основными программными структурами, которые обеспечивают структурирование программы и многократное использование программного кода.

Составные инструкция `def` в языке Python состоит из строки заголовка и следующего за ней блока инструкций:

```
def name(arg1, arg2,..., argN):
    body
```

Инструкции `def` определяются имя функции `name`, с которым будет связан объект функции, и список из нуля или более аргументов (параметров) `arg1`, `arg2`, ..., `argN`. Параметры связаны с объектами, которые передаются в функцию в точке вызова. Тело функции `body` образует программный код, который выполняется интерпретатором всякий раз, когда производится вызов функции.

В любом месте в теле функции может располагаться инструкция `return`. Инструкция `return` является необязательной — при ее отсутствии работа функции завершается, когда поток управления достигает конца тела функции.

Пример создания функции и ее вызова:

```
def add(x, y):
    return x + y

print add(2, 3), add('Poly', 'morphism')
```

Результат работы программы

```
5 Polymorphism
```

Вызов осуществляется по имени функции (add), добавляя круглые скобки после ее имени, внутри которых указаны аргументы, значения которых будут присвоены аргументам, указанным в заголовке функции.

При первом обращении к функции проводится сложение простых чисел 2 и 3, во втором случае мы работаем со строками `'Poly'` и `'morphism'`, результат есть конкатенация этих строк. Этот пример демонстрирует полиморфизм в языке Python, когда одна и та же функция реализована одинаково для разного типа данных.

Переменное число аргументов

В языке Python реализованы различные возможности определения функции с переменным числом аргументов. Наиболее простой и важный способ состоит в установлении значения по умолчанию для одного или нескольких аргументов.

Заданные по умолчанию значения аргумента используются в том случае, если при вызове функции соответствующий аргумент не был определен. Для этого необходимо присвоить соответствующему аргументу некоторое значение в определении функции. Если во время вызова функции не будет передан аргумент, то переменная примет присвоенное ей по умолчанию значение.

Результат работы программы

```
def f(x, a=1):
    return a*x**2

print f(2), f(2, 2)
```

будет

```
4 8
```

В Python имеется возможность определить функцию таким образом, чтобы ее можно было вызвать с произвольным числом аргументов. В этом случае аргументы будут переданы в виде кортежа. Перед переменным числом аргументов может присутствовать произвольное число обычных аргументов.

Используем, например, функция расчета суммы элементов списка

```
def lsum(*args):
    smm = 0
    for arg in args:
        smm += arg
    return smm

print lsum(), lsum(1), lsum(1, 2, 3)
```

Результатом работы будет

0 1 6

Возможна также передача аргументов аргументов по ключам с использованием конструкции `def name(**args)`, когда аргументы будут собраны в новый словарь, который можно обрабатывать обычными инструментами, предназначенными для работы со словарями.

lambda функции

Python разрешает создание анонимных функций (например, функции, которые не связаны с именем) во время выполнения, используя конструкцию `lambda`. Объекты функций в этом случае создаются в форме выражений.

За ключевым словом `lambda` следуют один или более аргументов и далее, вслед за двоеточием, находится выражение:

`lambda arg1, arg2, ..., argN: выражение.`

Разницу между обычным определением функции (`f1`) и `lambda` функции (`f2`) наиболее просто пояснить на примере работы программы

```
def f1 (x):
    return x**2

f2 = lambda x: x**2
print f1(4), f2(4)
```

Такие конструкции удобны для создания малых функций и позволяют встраивать определения функций в программный код, который их использует.

Пространство имен в Python

С каждой точкой программы можно связать три пространства имен: локальное, глобальное и встроенное. Пространство имен определяет отображение имен в объекты.

Пространства имен создаются в различные моменты времени и имеют разную продолжительность жизни. Пространство встроенных имен создается при запуске интерпретатора и существует все время его работы. Глобальное пространство имен модуля создается, когда он считывается, и, обычно, также существует до завершения работы интерпретатора. Локальное пространство имен функции создается при вызове функции и удаляется при выходе из нее.

Имена, определяемые внутри тела функции, видны только программному коду внутри тела функции. К этим именам нельзя обратиться за пределами функции. Свойство глобальности во вмещающем модуле переменной внутри тела функции обеспечивается использованием инструкции `global`.

Поиск имен ведется последовательно в четырех областях видимости: локальной, затем в объемлющей функции (если таковая имеется), затем в глобальной и, наконец, во встроенной.

2.5 Модули

Программа в Python, создание и использование модулей, поиск модуля, стандартные модули, пакеты модулей.

Программа в Python

При модульном подходе к программированию большая задача разбивается на несколько более мелких, каждую из которых (в идеале) решает отдельный модуль. Важно составить такую композицию модулей, которая позволила бы свести к минимуму связи между ними. Набор функций, имеющий множество связей между своими элементами, логично расположить в одном модуле.

Программа на языке Python организована как один главный файл, к которому могут подключаться дополнительные файлы. Каждый такой файл представляет собой отдельный модуль, содержащий инструкции. Именно такая процедура реализуется при работе в среде разработки NetBeans IDE.

Модули для использования в программах на языке Python по своему происхождению делятся на обычные (написанные на Python) и модули расширения, написанные на другом языке программирования (например, на C).

Для включения модуля в программу на языке Python он должен быть импортирован главным файлом, или другими файлами программы.

Создание и использование модулей

Для создания модуля достаточно ввести программный код на языке Python в текстовый файл и сохранить его с расширением .py. Такой файл по умолчанию считается модулем Python. Все имена, которым выполнено присваивание на верхнем уровне модуля, станут его атрибутами (именами, ассоциированными с объектом модуля) и будут доступны для использования клиентами.

При работе в NetBeans IDE для добавление модуля в проект выбирается команда File | New File и тип файла Python. Далее в режиме диалога задается имя файла и его расположение.

Создадим, например, модуль

```
%  module1 (file module1.py)
def times(x, y):
    return x * y
```

```
def add(x, y):
    return x + y
```

который содержит две функции: `times` и `add`.

Будем использовать этот модуль в главном файле `main.py`. Для этого применяется инструкция `import`:

```
%  main (file main.py)
import module1
print module1.add(2, 3), module1.times('well ', 4)
```

После запуска программы выводиться строка

```
5 well well well well
```

В результате выполнения инструкции `import` в главном модуле становиться доступным имя (`module1`), которое ссылается на полный объект модуля. При обращении к его атрибутам (функциям `times` и `add`) необходимо использовать имя модуля с точкой (`module1.add()` и `module1.times()`).

Для изменения имени ссылки на импортируемый модуль используется конструкция `import ... as`. Результатом работы программы

```
%  main (file main.py)
import module1 as m1
s = m1.add('Import ', 'as ')
print m1.add(s, 'm1')
```

будет

```
Import as m1
```

Иногда бывает более удобно импортировать модуль не целиком, а отдельные имена из его области видимости. В этом случае используется инструкция `from ... import`. В нашем примере

```
%  main (file main.py)
from module1 import add
print add('Import ', 'add')
```

Импортируемое имя (`add`) копируется в область видимости, в которой находится сама инструкция `from ... import`, и его можно использовать без указания имени вмещающего модуля.

Имеется специальная форма инструкции `from ... import` с использованием символа `*` вместо списка импортируемых имен. В этом случае импортируются все имена:

```
%  main (file main.py)
from module1 import *
print add('Import ', 'all'), times('!', 3)
```

```
Import all !!!
```

Для того чтобы выяснить, какие имена определены в модуле, можно использовать встроенную функцию `dir()`. Она возвращает отсортированный список строк:

```
% main (file main.py)
import module1
print dir(module1)
```

```
['__builtins__', '__doc__', '__file__', '__name__',
'__package__', 'add', 'times']
```

Импортирование является относительно дорогостоящей операцией и она выполняется интерпретатором всего один раз первой инструкцией `import` или `from ... import`. Последующие операции импорта просто получают объект уже загруженного модуля. Возможность повторной загрузки модулей реализуется встроенной функцией `reload`.

Поиск модуля

При импорте модуля, интерпретатор Python ищет модуль в следующей последовательности:

- в текущем каталоге (где расположен главный файл);
- если модуль не найден, Python просматривает каждый каталог в переменной окружения PYTHONPATH;
- если все остальное не даст результатов Python проверяет каталоги, в которые были установлены модули стандартной библиотеки Python.

В силу такой организации поиска желательно располагать все модули программы в одном каталоге. Вы можете узнать как выглядит путь поиска при запуске вашей программы, просмотрев содержимое встроенного списка `sys.path` (после импорта модуля `sys`, который входит в состав стандартной библиотеки Python):

```
% main (file main.py)
import sys
import module1
print sys.path
```

```
['C:\\Vab\\Python\\Test1\\src', 'C:\\Program Files\\
NetBeans 6.7\\python1', 'C:\\Windows\\system32\\
python26.zip', 'C:\\Python26\\DLLs', 'C:\\Python26\\lib',
'C:\\Python26\\lib\\plat-win', 'C:\\Python26\\lib\\lib-tk',
'C:\\Python26', 'C:\\Python26\\lib\\site-packages',
'C:\\Vab\\Python\\Test1\\src']
```

При работе в среде разработки NetBeans IDE пути поиска можно посмотреть и отредактировать на вкладке Python Path диалога Python Platform Manager, который активизируется командой Tools | Python Platforms. Дополнительные пути для основного проекта устанавливаются в диалоге Project Properties (команда File | Project Properties) при выборе Categories: Python.

Стандартные модули

Привлекательной стороной языка программирования Python является стандартная библиотека, которая является обширной коллекцией дополнительных модулей. Эта коллекция насчитывает многие десятки крупных модулей и обеспечивает поддержку наиболее распространенных задач.

Набор модулей для работы с операционной системой позволяет писать кроссплатформенные приложения. Имеются также средства для работы со многими сетевыми протоколами и форматами интернета, для разбора и создания почтовых сообщений, для работы с XML. Отметим также модули для работы с регулярными выражениями, текстовыми кодировками, мультимедийными форматами, криптографическими протоколами, архивами, поддержки сериализации данных. Некоторые возможности стандартной библиотеки Python, которые интересны при научных вычислениях, мы рассмотрим позже.

Более полную информацию можно найти в справочном руководстве по стандартной библиотеке языка Python. Она доступна для установленной версии Python по команде Пуск (Start) | Python x.x | Python Manual в разделе The Python Standard Library.

Пакеты модулей

В больших программных продуктах для организации иерархической файловой структуры удобно использовать каталог. Типичной является ситуация, когда в отдельных подкаталогах находятся файлы с одинаковыми именами. Python даст возможность импортировать не только имена модулей но и имена каталогов. Каталог в этом контексте является пакетом, а такая операция импортирования в Python называется импортированием пакетов.

Основная особенность импортирования пакетов состоит в указании пути к соответствующему модулю в инструкциях `import` и `from ... import`. Пусть пакет идентифицируется как каталог `dir1`. В нем есть подкаталог `dir2`, который содержит нужный нам модуль `module1` (файл `module1.py`). Тогда импорт этого модуля будет осуществляться инструкцией

```
% main (file main.py)
import dir1.dir2.module1
```

При этом каталог `dir1` должен находится в пути поиска модулей.

3

Математический Python

Встроенные функции и стандартная библиотека Python предоставляют широкие возможности для решения самых различных задач программирования. Кроме того, Python допускает расширение за счет библиотек, созданных сторонними разработчиками. В данной главе рассматриваются инструментальные средства для решения задач численного анализа. Дается краткое описание математических модулей стандартной библиотеки Python и расширений NumPy и SciPy, которые могут рассматриваться как свободный и более мощный эквивалент системы программирования математических вычислений MATLAB. Визуализация расчетов обеспечивается пакетом научной графики Matplotlib.

3.1 Встроенные функции и стандартная библиотека

Встроенные функции, модули стандартной библиотеки, математические модули.

Встроенные функции

Python имеет ряд встроенных функций, некоторые из них мы уже упоминали. Общий список встроенных функций с описанием содержится в документации (команда Пуск (Start) | Python x.x | Python Manual). Здесь мы отметим лишь некоторые из них, которые интересны при проведении вычислений.

При арифметических операциях над числами смешанного типа проводится автоматическое преобразование к общему более старшему типу. Иерархия чисел: целые числа, длинные целые числа, числа с плавающей точкой, комплексные числа. В частности, интерпретатор Python автоматически преобразует целые числа в длинные целые, если их значения оказываются слишком большими, чтобы уместиться в формат простого целого числа. Для принудительного преобразования используются встроенные функции `int()`, `long()`, `float()`, `complex()`. Примеры их использования:

```
>>> a = 1
>>> b = - 2.3
```

```
>>> c = '4.5'
>>> print a + b
-1.3
>>> print int(b)
-2
>>> print float(c)
4.5
>>> print complex(a, b)
(1-2.3j)
```

Core Python поддерживает только несколько математических функций. Например, `abs(x)` возвращает абсолютное значение числа, т.е модуль данного числа $|x|$:

```
>>> print abs(-1.2), abs(3 + 4j)
1.2 5.0
```

Встроенные функции `max(x)` и `min(x)` используются для поиска максимального и минимального значений:

```
>>> a = [1, 2, 3, 4]
>>> b = [5, 3, 1]
>>> print max(a), min(b)
4 1
>>> print max(3, 1, 7, 4)
7
```

Функция `cmp(x, y)` сравнивает x и y и возвращает 1, когда `x > y`, -1, когда `x < y` и ноль при `x = y`:

```
>>> a = 0.5
>>> b = 1
>>> print cmp(a, b), cmp(a+1, b)
-1 1
>>> print cmp(2*a, b)
0
```

Упомянем также встроенную функцию `pow(x, y)`, которая возвращает x возведённый в степень y. Ранее мы упоминали некоторые другие встроенные функции, такие как `range`, `open`.

Модули стандартной библиотеки

Из большой массы модулей стандартной библиотеки отметим лишь некоторые. Модуль `sys` предоставляет доступ к переменным, которые содержит информацию о среде выполнения программы, об интерпретаторе Python. Выше

мы уже упоминали объект из этого модуля `path` — список путей поиска модулей. Функция `exit(arg)` с необязательным аргументом прерывает выполнение программы. Можно воспользоваться `exit('error')` для того чтобы прервать выполнение программы и вывести сообщение об ошибке. Инструкция `platform` выводит информацию об используемой платформе. Файловые объекты, соответствующие стандартным потокам ввода, вывода и ошибок интерпретатора связаны с объектами `stdin`, `stdout`, `stderr` модуля `sys`.

Взаимодействие с операционной системой поддерживается модулем `os`. В частности, функция `getcwd()` возвращает строку, представляющую текущий рабочий каталог. Для изменения текущий рабочий каталог (на `path`) используется `chdir(path)`. Для удаления файла (каталога) с именем `path` можно использовать функцию `remove(path)` (`removedirs(path)`). Переименовывать файл (каталог) `src` в `dst` поможет функция `rename(src, dst)`.

Для выявления узких мест в программе необходимо замерить временные интервалы на выполнения отдельных частей программы. С этой целью можно использовать модуль `time`, который предоставляет различные функции для работы со временем. Точка отсчета (начало эпохи — 0 часов 1 января 1970г.) для вашей платформы выводиться функцией `gmtime(0)`:

```
import time
print time.gmtime(0)
```

```
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1,
tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3, tm_yday=1,
tm_isdst=0)
```

Для замера производительности может быть использована функция `clock()`, которая возвращает текущее процессорное время в секундах в виде вещественного числа. Функция `sleep(secs)` приостанавливает выполнение на `secs` (вещественное число) секунд. Более полный анализ производительности программ на языке Python осуществляется с использованием специализированных модулей `profile` и `pstats`, которые включены в стандартную библиотеку.

Весьма полезные функции сохранения и копирования объектов языка Python предоставляет модуль `pickle`, который преобразует объекты языка Python в последовательность байтов и обратно (сериализация). Полученную таким образом последовательность байтов можно сохранить во внешней памяти или передать его по каналам связи. Модуль `pickle` реализован на языке Python и работает достаточно медленно. Для более быстрой работы можно использовать реализованный на C модуль `cPickle`.

Для сохранения данных используем следующую программу для записи данных в файл `mydata.dat`:

```
import pickle
data = ("abc", 1.23, [1, 2, 3])
```

```
file = open("mydata.dat", "w")
p = pickle.Pickler(file)
p.dump(data)
file.close()
```

Здесь `Pickler(file)` возвращает объект, реализующий сериализацию. Аргумент `file` должен быть файловым объектом, имеющим метод `write()`. Для сериализация данных `data` используется `dump(data)`.

Обратная процедура извлечения данных реализуется следующей программой:

```
import pickle
file = open("mydata.dat", "r")
mydata = pickle.load(file)
file.close()
print mydata
```

```
('abc', 123, [1, 2, 3])
```

Для восстановления и возвращения ранее сериализованных объектов используется `load(file)`.

В ранних версиях Python для работы со строками использовался модуль `string`. В настоящее время функции этого модуля реализованы как методы для строк.

Пример использования для разделения и объединения строк:

```
import string
s = "one, two, three"
print string.split(s, ", ")
r = s.split(",")
print r
print "".join(r)
```

```
['one', 'two', 'three']
['one', ' two', ' three']
one two three
```

Из других полезных методов работы со строками отметим `find()` и `replace()` для поиска и замены.

Математические модули

Несколько более подробно обсудим модули стандартной библиотеки Python для выполнения основных математических операций. Математические функции для работы с вещественными числами представлены в модуле `math`. Работа с комплексными числами обеспечивается модулем `cmath`. Многие функции

в этих модулях имеют одинаковые имена, но результаты могут быть совершенно разные. В частности, при вычислении квадратного корня из отрицательного числа (функции `math.sqrt(-1)` и `cmath.sqrt(-1)`).

Функции модуля `math` приведены в табл. 3.1. Если аргумент обозначен буквой `z`, то функция определена как в `math`, так и в `cmath`.

Таблица 3.1 *Функции в модулях* `math` *и* `cmath`

Функция	**Описание**
acos(z)	арккосинус z
acosh(z)	гиперболический арккосинус z
asin(z)	арксинус z
asinh(z)	гиперболический арксинус z
atan(z)	арктангенс z
atan2(y, x)	atan(y/x)
atanh(z)	гиперболический арктангенс z
ceil(x)	наименьшее целое, большее или равное x
copysign(x, y)	x со знаком y
cos(z)	косинус z
cosh(x)	гиперболический косинус x
degrees(x)	перевод величины угла x из радиан в градусы
exp(z)	экспонента (e^z)
fabs(x)	абсолютное значение x
factorial(x)	факториал x!
floor(x)	наибольшее целое, меньшее или равное x
fmod(x, y)	остаток от деления x на y
frexp(x)	возвращает мантиссу и порядок x как пару (m, i), где m — число с плавающей точкой, а i — целое, такое, что $x = m2^i$.
hypot(x, y)	$\sqrt{x^2 + y^2}$
ldexp(m, i)	Функция, обратная frexp(x) ($m2^i$)
log(z)	натуральный логарифм z
log10(z)	десятичный логарифм z
modf(x)	возвращает пару (p, q) – целую и дробную часть x. Обе части имеют знак исходного числа
phase(z)	полярный угол комплексной величины z
polar(z)	комплексное число z в полярных координатах (r, φ)
pow(x, y)	x^y
radians(x)	перевод величины угла x из градусов в радианы
rect(r, φ)	переход от полярных координат к декартовым
sin(z)	синус z
sinh(z)	гиперболический синус z
sqrt(z)	корень квадратный от z
tan(z)	тангенс z
tanh(z)	гиперболический тангенс z

В модулях `math`, так и в `cmath` также определены две вещественные константы: `pi` — число π и `e` — число e.

Следует отметить также модуль `random`, который позволяет генерировать псевдослучайные числа для различных распределений. Для генерации слу-

чайного вещественное число r такое, что $(0.0 \leq r < 1.0)$ используется функция `random()`. Функция `uniform(a, b)` возвращает случайное вещественное число r такое, что $a \leq r < b$. Для получения случайного целого числа из диапазона `range(start, stop, step)` используется функция `randrange(start, stop, step)`.

Генерация числа из последовательности нормально распределенных псевдослучайных чисел (распределение Гаусса) производиться функцией `gauss(mu, sigma)`, где `mu` — математическое ожидание а `sigma` — стандартное отклонение. Другая возможность связана с использованием функции `normalvariate` с теми же параметрами. Имеется возможность генерации чисел из других последовательностей псевдослучайных чисел (бета-распределение, экспоненциальное распределение, гамма-распределение, логарифмически нормальное распределение и др.).

3.2 Пакет NumPy

Вычислительное ядро Python, массивы в NumPy, создание массивов, работа с массивами, математические функции, матричные объекты, линейная алгебра, работа с полиномами, другие возможности NumPy.

Вычислительное ядро Python

Математические алгоритмы, реализованные на интерпретируемых языках, работают, вообще говоря, гораздо медленнее чем при использовании компилируемых языков, таких как C. В Python для большого количества вычислительных алгоритмов проблема низкого быстродействия решается за счет использования специальных библиотек (вычислительный Python), которые специально ориентированы на поддержку многомерных массивов и множество функций и операторов для работы с ними. Типичной является ситуация, когда вычислительный алгоритм базируется на последовательности операций над массивами и матрицами, и в этих условиях программа на языке Python работает также быстро как и программа написанная на C.

Наиболее интересным примером вычислительного пакета является NumPy[1]. NumPy включает модули для вычислений с многомерными массивами, необходимых для многих численных приложений. Благодаря этому NumPy обеспечивает возможности, которые характерны для таких систем как MATLAB[2] и GNU Octave[3] (аналог MATLAB). Язык программирования MATLAB также является интерпретируемым и позволяет писать быстрые программы пока большинство операций производятся над массивами или матрицами, а не над скалярами.

[1] NumPy — http://numpy.scipy.org/
[2] MATLAB — www.mathworks.com/products/matlab/
[3] GNU Octave — www.gnu.org/software/octave/

NumPy прост и удобен для пользования и включает помимо средств для работы с многомерными массивами модули для:

- решения задач линейной алгебры;
- быстрого преобразования Фурье;
- генерации массивов случайных чисел.

На основе пакета NumPy строятся другие инструментальные средства для решения задач вычислительной математики, некоторые их них мы отметим ниже. Это обстоятельство позволяет отнести пакет NumPy к базовым математическим пакетам, хотя он и не входит в стандартную библиотеку Python.

Массивы в NumPy

Пакет NumPy определяет новый тип данных — N-мерный массив (ndarray). Массив — это упорядоченный набор данных для хранения данных одного типа, идентифицируемых с помощью одного или нескольких индексов. В NumPy массив имеет, как правило, постоянную длину и хранит единицы данных одного и того же типа dtype (Data type objects). Количество используемых индексов массива может быть различным. Массивы с одним индексом называют одномерными (вектора), с двумя — двумерными (матрицы).

Количество размерностей и длина массива по каждой оси называются формой массива (the shape of the array). В NumPy форма массива описывается как кортеж из N натуральных чисел, длина которого (N) есть размерность массива, а элементами кортежа является длина массива по соответствующей оси.

Пример двумерного массива 2×3 целых чисел:

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
print type(a)
print a.shape
print a.dtype
```

```
<type 'numpy.ndarray'>
(2, 3)
int32
```

В качестве элементов массива можно используются все числовые типы (см. табл. 3.2).

Таблица 3.2 *Типы элементов массива*

Тип данных	Описание
bool	логическое (True или False)
int	целое (обычно int32 или int64)
int8	байт (от -128 до 127)
int16	целое (от -32768 до 32767)
int32	целое (от -2147483648 до 2147483647)
int64	целое (от -9223372036854775808 до 9223372036854775807)
uint8	беззнаковое целое (от 0 до 255)
uint16	беззнаковое целое (от 0 до 65535)
uint32	беззнаковое целое (от 0 до 4294967295)
uint64	беззнаковое целое (от 0 до 18446744073709551615)
float	краткая форма для float64
float32	вещественное с одинарной точностью
float64	вещественное с двойной точностью
complex64	комплексное (float32 для вещественной и мнимой части)
complex128	комплексное (float64 для вещественной и мнимой части)

Создание массивов

Наиболее простой способ создания массива связан с использованием функцию `array()`, в которой элементы массива задаются в виде вложенных списков, с явным или неявным заданием типа.

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]], 'float')
print a
print a.shape
print a.dtype
```

```
[[ 1.  2.  3.]
 [ 4.  5.  6.]]
(2, 3)
float64
```

Создание массива из списка и обратная процедура (с помощью метода `tolist()`) иллюстрируется следующим примером:

```
import numpy as np
lst = [[1, 2, 3], [4, 5, 6]]
a = np.array(lst)
print a
print a.tolist()
```

```
[[1 2 3]
 [4 5 6]]
[[1, 2, 3], [4, 5, 6]]
```

При вычислениях часто используются специальные матрицы. В единичных матрицах элементы главной диагонали равны 1, а все остальные — 0, нулевые матрицы, у которых все элементы равны 0. Отметим также матрицы, все элементы которых равны 1. Для инициализации таких массивов в NumPy имеются соответствующие инструменты.

Функция `zeros(sh, dt)` позволяет сформировать массив с нулевыми элементами по форме `sh` и необязательному аргументу `dt` — тип элемента. Аналогично `ones(sh, dt)` формирует массив с единичными элементами. Для получения матрицы с единицами на главной диагонали и нулями для других элементов используется функция `eye(n, m)`, для квадратной матрицы — `eye(n)` или `identity(n)`. Примеры генерации массивов:

```
import numpy as np
a0 = np.zeros((2, 3), 'float')
print 'a0:\n', a0
a1 = np.ones((3, 2), 'int')
print 'a1:\n', a1
a2 = np.eye(3, 2, dtype=int)
print 'a2:\n', a2
a3 = np.identity(4, 'float')
print 'a3:\n', a3
```

```
a0
[[ 0.  0.  0.]
 [ 0.  0.  0.]]
a1
[[1 1]
 [1 1]
 [1 1]]
a2
[[1 0]
 [0 1]
 [0 0]]
a3
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

Имеются специальные процедуры для формирования одномерных массивов (векторов). Функция `arange(start, stop, step)` формирует массив величин, начиная со `start` до (не включая) `stop` с шагом `step`. Такая конструкция подобна `array(range(start, stop, step))`, но работает не только с целыми числами. Функция `linspace(start, stop, n)` создает одномерный массив с числом элементов `n`, значения которого равномерно распределены от `start` до (включая) `stop`.

```
import numpy as np
a0 = np.array(range(1,10,3))
print 'a0:\n', a0
a1 = np.arange(0.,10.,2.)
print 'a1:\n', a1
a2 = np.linspace(0.,1.,11)
print 'a1:\n', a2
```

```
a0
[1 4 7]
a1
[ 0  2  4  6  8]
a1
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1. ]
```

Имеются и некоторые дополнительные функции создания массивов (например, `fromfile()`, `empty()`), о которых вы можете узнать в документации по пакету NumPy.

Работа с массивами

Для копирования массивов можно использовать метод `copy()`, а изменить форму массива — `shape()`. Аналогичный результат достигается при вызове `reshape()`. Функция `resize()` аналогична `reshape()`, но работает и в случае, когда число элементов исходного и получаемого массивом не совпадает.

```
import numpy as np
a0 = np.linspace(1.,10.,10)
print 'a0:\n', a0
a2 = np.copy(a0)
a1 = np.reshape(a0, (2, 5))
print 'a1:\n', a1
a2.shape = (-1,5)
print 'a2:\n', a2
a3 = np.resize(a2, (3, 4))
print 'a3:\n', a3
```

```
a0
[  1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
a1:
[[  1.   2.   3.   4.   5.]
 [  6.   7.   8.   9.  10.]]
a2:
[[  1.   2.   3.   4.   5.]
 [  6.   7.   8.   9.  10.]]
a3:
```

```
[[  1   2   3   4 ]
 [  5   6   7   8 ]
 [  9  10   1   2 ]]
```

Здесь используется -1 при выборе формы массива (в нашем примере (2,-1)) для неявного задания числа элементов по одной оси массива — вычисляется по общему числу элементов.

Для изменения элемента массива используется доступ по индексу, а для части массива — получение среза. Некоторые основные возможности иллюстрируются следующим примером.

```
import numpy as np
a = np.reshape(np.linspace(1,12,12),(3,4))
print 'a:\n', a
print 'element (2,3): ', a[2,3]
print 'row 1: ', a[1]
print 'last row: ', a[-1,:]
print 'column 1: ', a[:,1]
print 'window 2x2: \n', a[1:3,0:2]
```

```
a
[[  1   2   3   4.]
 [  5   6   7   8.]
 [  9  10  11  12.]]
element (2,3)  12.0
row 1  [ 5.  6.  7.  8.]
last row.  [  9.  10.  11.  12.]
column 1.  [  2.   6.  10.]
window 2x2
[[  5.   6.]
 [  9.  10.]]
```

Запись массива в файл (бинарный, NumPy формат, расширение .npy) производиться функцией save(), а чтение из файла — load().

```
import numpy as np
a = np.reshape(np.linspace(1.,8.,8),(2,4))
print 'a:\n', a
np.save('c:/tmp/data.npy', a)
a1 = np.load('c:/tmp/data.npy')
print 'a1:\n', a1
```

```
a
[[ 1.  2.  3.  4.]
 [ 5.  6.  7.  8.]]
a1
[[ 1.  2.  3.  4.]
```

```
[ 5  6  7  8 ]]
```

При чтении и записи массивов в текстовый файл используются функции `tofile()` и `fromfile()`.

Математические функции

В NumPy реализованы стандартные арифметические операции над элементами массивов (табл. 3.3).

Таблица 3.3 *Арифметические операции над массивами*

Функция	Описание
add(x, y)	поэлементное сложение
subtract(x, y)	поэлементное вычитание
multiply(x, y)	поэлементное умножение
divide(x, y)	поэлементное деление
power(x, y)	поэлементное возведение в степень
negative(x)	изменение знака элементов массива

Примеры использования арифметических поэлементных операций над массивами:

```
import numpy as np
a0 = np.linspace(1., 10., 10).reshape(2, 5)
print 'a0:\n', a0
a1 = np.ones(10).reshape(2, 5)
a1 = np.add(a1, 1.)
print 'a1:\n', a1
a2 = np.add(a0, a1)
print 'a2:\n', a2
a3 = np.multiply(a0, a1)
print 'a3:\n', a3
a4 = np.power(a0, a1)
print 'a4:\n', a4
```

```
a0
[[  1.   2.   3.   4.   5.]
 [  6.   7.   8.   9.  10.]]
a1
[[ 2.  2.  2.  2.  2.]
 [ 2.  2.  2.  2.  2.]]
a2
[[  3.   4.   5.   6.   7.]
 [  8.   9.  10.  11.  12.]]
a3
```

```
[[  2   4   6   8.  10 ]
 [ 12  14.  16  18  20 ]]
a4
[[   1    4    9   16   25 ]
 [  36   49   64   81  100 ]]
```

Среди других функций для поэлементных операций над массивами отметим:

Тригонометрические: $\sin(x)$, $\cos(x)$, $\tan(x)$, $\arcsin(x)$, $\arccos(x)$, $\arctan(x)$, $\mathrm{hypot}(x, y)$, $\mathrm{arctan2}(x, y)$, $\mathrm{degrees}(x)$, $\mathrm{radians}(x)$;

Гиперболические: $\sinh(x)$, $\cosh(x)$, $\tanh(x)$, $\mathrm{arcsinh}(x)$, $\mathrm{arccosh}(x)$, $\mathrm{arctanh}(x)$;

Экспоненциальные и логарифмические: $\exp(x)$, $\mathrm{expm1}(x)$ $(= \exp(x) - 1)$, $\log(x)$, $\mathrm{log10}(x)$, $\mathrm{log2}(x)$, $\mathrm{log1p}(x) (= \log(1 + x))$.

Другие функции соответствуют обозначениям математических функций в модуле `math` стандартной библиотеки Python (см. табл. 3.1).

```
import numpy as np
a0 = np.linspace(0., np.pi, 9)
print 'a0:\n', a0
a1 = np.sin(a0)
print 'a1:\n', a1
a2 = np.cos(a0)
print 'a2:\n', a2
a3 = np.multiply(a1, a1) + np.multiply(a2, a2) - 1.
print 'a3:\n', a3
```

```
a0
[ 0          0.39269908  0.78539816
  1.17809725  1.57079633  1.96349541
  2.35619449  2.74889357  3.14159265]
a1
[  0.00000000e+00   3.82683432e-01   7.07106781e-01
   9.23879533e-01   1.00000000e+00   9.23879533e-01
   7.07106781e-01   3.82683432e-01   1.22464680e-16]
a2
[  1.00000000e+00   9.23879533e-01   7.07106781e-01
   3.82683432e-01   6.12323400e-17  -3.82683432e-01
  -7.07106781e-01  -9.23879533e-01  -1.00000000e+00]
a3:
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

Для работы с массивами с комплексными элементами предназначены функции: $\mathrm{angle}(x)$ (аналог функции $\mathrm{phase}(x)$), $\mathrm{real}(x)$, $\mathrm{imag}(x)$ и $\mathrm{conj}(x)$, которая возвращает массив с комплексно-сопряженными элементами.

```
import numpy as np
```

```
a0 = np.linspace(1., 10., 10).reshape(2, 5)
print 'a0:\n', a0
a1 = np.add(a0, 1.j)
print 'a1:\n', a1
a2 = np.conj(a1)
print 'a2:\n', a2
a3 = np.real(a1)
print 'a3:\n', a3
a4 = np.imag(a1)
print 'a4:\n', a4
```

```
a0
[[  1.   2.   3.   4.   5.]
 [  6.   7.   8.   9.  10.]]
a1
[[ 1.+1.j  2.+1.j  3.+1.j  4.+1.j  5.+1.j]
 [ 6.+1.j  7.+1.j  8.+1.j  9.+1.j 10.+1.j]]
a2
[[ 1.-1.j  2.-1.j  3.-1.j  4.-1.j  5.-1.j]
 [ 6.-1.j  7.-1.j  8.-1.j  9.-1.j 10.-1.j]]
a3:
[[  1.   2.   3.   4.   5.]
 [  6.   7.   8.   9.  10.]]
a4
[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]
```

Матричные объекты

В вычислительной практике вместо N-мерных массивов часто удобнее оперировать с векторами и матрицами (одномерными и двумерными массивами). В частности, при программировании вместо поэлементных операций с массивами можно использовать матричные операции. Работа с матричными типами является вычислительной основой системы научных вычислений MATLAB.

В пакете NumPy помимо общего типа N-мерного массива (ndarray) имеются подклассы векторных, матричных и тензорных объектов. Нас интересуют матричные объекты NumPy (matrix) в плане их использования при решении задач линейной алгебры. Здесь мы отметим особенности использования matrix в сравнении с ndarray. Можно заметить, что, с одной стороны, работа с массивами как с более общим типом данных дает больше возможностей. С другой стороны, матричные задачи часто самодостаточны для приложений и нет необходимости в расширенных возможностях работы с общими массивами.

Для матричных объектов возможна инициализация по строке, элементы матрицы задаются в соответствии с синтаксисом MATLAB с пробелом в качестве разделителя для строки и разделителем ; для столбцов.

```
import numpy as np
a = np.mat('1 2 3; 4 5 3')
b = np.mat([[1, 2, 3], [4, 5, 6]])
print type(a), 'a:\n', a
print type(b), 'b:\n', b
```

```
<class 'numpy.core.defmatrix.matrix'> a:
[[1 2 3]
 [4 5 3]]
<class 'numpy.core.defmatrix.matrix'> b:
[[1 2 3]
 [4 5 6]]
```

Основной операцией линейной алгебры является умножением матриц (как частный случай, матрицы на вектор и вектор на вектор). При использовании объектов ndarray и matrix в NumPy синтаксис умножение матриц разный. В случае ndarray символ * соответствует поэлементному умножению, для умножения матриц используется функция dot(). В случае matrix символ * применяется для умножения матриц и функция multiply() для поэлементного умножения.

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
print 'ndarray - a*b:\n', a*b
print 'ndarray - dot(a,b):\n', np.dot(a, b)
c = np.mat(a)
d = np.mat(b)
print 'matrix - c*d:\n', c*d
print 'matrix - multiply(a,b):\n', np.multiply(c, d)
```

```
ndarray - a*b
[[ 5 12]
 [21 32]]
ndarray - dot(a,b)
[[19 22]
 [43 50]]
matrix - c*d
[[19 22]
 [43 50]]
matrix - multiply(a,b)
[[ 5 12]
 [21 32]]
```

В этом примере используется функция `mat()` для преобразования объектов ndarray в matrix. Ее можно использовать при инициализации матриц с помощью функций типа `zeros()`, `ones()`, `eye()`, `linspace()` для задания массивов.

При работе с массивами удобно использовать атрибут `.T` для получения транспонированной матрицы (замена строк на столбцы исходной матрицы). Для матричных объектов используется также следующие атрибуты:

- `.H` — транспонирование с комплексным сопряжением;
- `.I` — обратная матрица;
- `.A` — преобразование матрицы в двумерный массив.

```
import numpy as np
a = np.mat(np.linspace(1, 4, 4).reshape(2,2))
print 'linspace - reshape -> a:\n', a
print 'a: ', type(a)
b = a.I
print 'a.T:\n', a.T
print 'a.I:\n', b
print 'a*a.I:\n', a*b
print 'a.A: ', type(a.A)
```

```
linspace - reshape -> a:
[[ 1.  2.]
 [ 3.  4.]]
a:  <class 'numpy.core.defmatrix.matrix'>
a.T
[[ 1.  3.]
 [ 2.  4.]]
a.I
[[-2.   1. ]
 [ 1.5 -0.5]]
a*a.I
[[  1.00000000e+00   0.00000000e+00]
 [  8.88178420e-16   1.00000000e+00]]
a.A:  <type 'numpy.ndarray'>
```

За описаниями других функций для работы над матрицами и массивами (над ndarray и matrix) мы отсылаем к документации по пакету NumPy.

Линейная алгебра

В пакете NumPy реализованы базовые операции линейной алгебры (модуль linalg). Упомянем вначале функции для вычисления характеристик матрицы. Для вычисления нормы вектора (например, евклидовой) и нормы квадратной

матрицы (например, Фробениуса) привлекается функция `norm()`. Для вычисления числа обусловленности квадратной матрицы используется `cond()`, определителя — `det()`, а для следа матрицы — `trace()` из пакета NumPy.

```
import numpy as np
a = np.mat([[1, 2], [3, 4]])
print 'a:\n', a
print 'norm a: ', np.linalg.norm(a)
print 'cond a: ', np.linalg.cond(a)
print 'det a: ', np.linalg.det(a)
print 'trace a: ', np.trace(a)
```

```
a:
[[1 2]
 [3 4]]
norm a  5.47722557505
cond a  14.9330343737
det a  -2.0
trace a  5
```

В модуле линейной алгебры имеются функции разложения матрицы на множители. Для симметричной положительно определенной матрицы A имеет место разложение Холецкого, когда $A = LL^*$, где L — нижнетреугольная матрица (функция `cholesky()`). Для общих матриц используется QR-разложение, когда $A = QR$, где Q — ортогональная матрица, а R— верхнетреугольная матрица (функция `qr()`). Сингулярное разложение матрицы проводится функцией `svd()`.

```
import numpy as np
a = np.mat([[2, 1], [3, 4]])
b = (a + a.T) / 2
print 'a:\n', a
print 'a:\n', b
print 'lower-triangular (Cholesky) matrix:\n', np.linalg.cholesky(b)
q, r = np.linalg.qr(a)
print 'orthonormal matrix q: \n', q
print 'upper-triangular matrix r: \n', r
print 'q*r: \n', q*r
```

```
a:
[[2 1]
 [3 4]]
a:
[[2 2]
 [2 4]]
lower-triangular (Cholesky) matrix
[[ 1.41421356  0.        ]
```

```
 [ 1.41421356  1.41421356]]
orthonormal matrix q:
[[-0.5547002  -0.83205029]
 [-0.83205029  0.5547002 ]]
upper triangular matrix r:
[[-3.60555128 -3.88290137]
 [ 0.          1.38675049]]
q*r
[[ 2.  1.]
 [ 3.  4.]]
```

Особого внимания в линейной алгебре уделяется решению систем линейных алгебраических уравнений. Для вычисления обратной матрицы используется функция `inv()`. Вычисление псевдообратных матриц проводится с помощью `pinv()`. Функция `solve()` предназначена для решения системы уравнений.

```
import numpy as np
a = np.mat([[4, 1], [2, 3]])
print 'a:\n', a
b = np.linalg.inv(a)
print 'b = a^(-1):\n', b
print 'a*b:\n', a*b
c = np.mat([1, 1]).T
print 'c:\n', c
x = np.linalg.solve(a, c)
print 'solution x:\n', x
print 'a*x - c:\n', a*x - c
```

```
a:
[[4 1]
 [2 3]]
b = a^(-1)
[[ 0.3 -0.1]
 [-0.2  0.4]]
a*b
[[  1.00000000e+00   0.00000000e+00]
 [ -1.11022302e-16   1.00000000e+00]]
c:
[[1]
 [1]]
solution x
[[ 0.2]
 [ 0.2]]
a*x - c
[[ 0.]
 [ 0.]]
```

В методе наименьших квадратов для матрицы A размера $n \times m$ $(n \geq m)$ и правой части b иметься вектор x (псевдорешение), который минимизирует норму невязки $r = b - Ax$. В NumPy для реализации метода наименьших квадратов используется функция `lstsq()`.

```
import numpy as np
a = np.mat([[1, 2], [4, 3], [5, 6]])
print 'a:\n', a
b = np.mat([7, 8, 9]).T
print 'b:\n', b
x, resids, rank, s = np.linalg.lstsq(a, b)
print 'solution x:\n', x
print 'resids r:\n', resids
y = np.linalg.pinv(a)*b
print 'function pinv -> y:\n', y
```

```
a:
[[1  2]
 [4  3]
 [5  6]]
b
[[7]
 [8]
 [9]]
solution x
[[ 0.55737705]
 [ 1.37704918]]
resids r
[[ 20.49180328]]
function pinv -> y
[[ 0.55737705]
 [ 1.37704918]]
```

Как видно из приведенного примера, тот же результат мы можем получить с использованием функции `pinv()`.

Отметим теперь возможности пакета NumPy для решения спектральных задач линейной алгебры. Для нахождения собственных значений используется функция `eigvals()` для общих матриц и функция `eigvalsh()` – для эрмитовых или вещественных симметричных матриц. Собственные значения и собственные вектора вычисляются в `eig()` и `eigh()` соответственно.

```
import numpy as np
a = np.mat([[1.5, 2.], [3., 4.]])
print 'a:\n', a
b = (a + a.T) / 2
c = (a - a.T) / 2
```

```
print 'b:\n', b
print 'c:\n', c
lb = np.linalg.eigvalsh(b)
print 'eigenvalues of b:\n', lb
lc = np.linalg.eigvals(c)
print 'eigenvalues of c:\n', lc
la, v = np.linalg.eig(a)
print 'eigenvalues of a:\n', la
print 'eigenvectors of a:\n', v
```

```
a
[[ 1.5   2. ]
 [ 3     4  ]]
b
[[ 1 5   2 5]
 [ 2 5   4  ]]
c
[[ 0    -0.5]
 [ 0 5   0  ]]
eigenvalues of b
[-0 04508497   5 54508497]
eigenvalues of c
[ 0.+0 5j   0.-0 5j]
eigenvalues of a
[ 0     5 5]
eigenvectors of a
[[-0 8          -0 4472136 ]
 [ 0 6          -0 89442719]]
```

Собственные значения симметричной вещественной матрицы (в нашем случае матрицы b) являются чисто вещественными, а кососимметричной матрицы (матрица c) — чисто мнимыми.

Работа с полиномами

Пакет NumPy поддерживает основные операции над полиномами. Полиномы задаются в виде списка коэффициентов, так что, например, $[1, 2, 3]$ соответствует полином $x^2 + 2x + 3$. Для задания полинома используется функция `poly1d()`, значение полинома `p` в точке `x` вычисляется как `p(x)` или с помощью функции `polyval(p,x)`.

```
import numpy as np
p = np.poly1d([1, 2, 3])
print np.poly1d(p)
print type(p)
```

```
print p(0.5), np.polyval(p, 0.5)
```

```
   2
1 x + 2 x + 3
<class 'numpy lib polynomial poly1d'>
4 25 4 25
```

Арифметические операции над полиномами поддерживаются следующими функциями: `polyadd()` — сложение, `polymul()` — умножение и `polydiv()` — деление.

```
import numpy as np
p = np.poly1d([1, 2, 3, 4, 5])
q = np.poly1d([6, 7, 8])
print 'p:\n', p
print 'q:\n', q
print 'p+q:\n', np.polyadd(p, q)
print 'p*q:\n', np.polymul(p, q)
d, r = np.polydiv(p, q)
print 'quotient:\n', d
print 'remainder:', r
```

```
p
   4     3     2
1 x + 2 x + 3 x + 4 x + 5
q
   2
6 x + 7 x + 8
p+q
   4     3     2
1 x + 2 x + 9 x + 11 x + 13
p*q
   6      5      4      3      2
6 x + 19 x + 40 x + 61 x + 82 x + 67 x + 40
quotient
         2
0 1667 x + 0 1389 x + 0 1157
remainder
2.079 x + 4 074
```

Для вычисления корней полинома p используется функция `roots(p)`, более краткая форма — `p.r`. Поддерживаются также операции дифференцирования (`polyder()`) и интегрирования полиномов (`polyint()`).

```
import numpy as np
p = np.poly1d([1, 2, 3, 4])
print 'p:\n', p
r = np.roots(p)
```

```
print 'roots:\n', np.roots(p)
print 'polynom:\n', np.poly(r)
print 'second derivate:', np.polyder(p, 2)
```

```
p
   3      2
1 x + 2 x + 3 x + 4
roots
[-1.65062919+0.j          -0.17468540+1.54686889j
-0.17468540-1.54686889j]
polynom
[ 1.  2.  3.  4.]
second derivate
6 x + 4
```

Здесь функция poly() вычисляет полином по заданным корням.

Другие возможности NumPy

Пакет NumPy включает другие мощные средства, ориентированные, прежде всего на работу с массивами различной природы. В частности, можно отметить методы сортировки и поиска. С учетом нашей ориентации на научные вычисления отдельного упоминания заслуживают методы быстрого преобразования Фурье. Более подробно мы будем говорить о них ниже при рассмотрении пакета SciPy.

Основной модуль генерации случайных чисел в Python random является частью стандартной библиотеки Python. Для подготовки больших случайных выборок используется модуль random в пакете NumPy. В этом модуле имеются функции для генерации массивов случайных чисел различных распределений и свойств.

Для создания массива случайных чисел, равномерно распределенных на $(0,1)$, используется функция `rand()`. Функция `randint()` дает массив заданной формы равномерно распределенных чисел из заданного интервала.

```
import numpy as np
a = np.random.rand(2, 3)
print 'random uniform distribution values  a:\n', a
b = np.random.randint(0, 100, (2, 5))
print 'random integers b:\n', b
```

```
random uniform distribution values  a
[[ 0.45907951  0.96193664  0.7795528 ]
 [ 0.85306773  0.69342406  0.79931246]]
random integers b
[[64 68 34 23  8]
```

```
 [38 19 90 43  7]]
```

Имеются функции для получения массивов случайных чисел, которые подчиняются тем или иным законам. Например, для получения нормально распределенных чисел (распределение Гаусса) используется функция `normal()`, первый аргумент которой есть заданное среднее (математическое ожидание), второй — стандартное отклонение.

```
import numpy as np
a = np.random.normal(0, 1, (2, 4))
print 'random normal distribution values  a:\n', a
```

```
random normal distribution values  a
[[ 0.01195366  0.671863   -0.62463833 -1.29743642]
 [ 0.73578021 -1.31941736  0.5853565  -1.54388744]]
```

Отметим некоторые возможности пакета NumPy по статистической обработке данных. Имеются функции для вычисления минимальных и максимальных значений всего массива или его отдельной оси (`amax()`, `amin()`), средних значений (`mean()`, `median()`) и стандартное отклонение (`std()`).

```
import numpy as np
a = np.array([[8, 1, 7], [3, 9, 2]])
print 'array a:\n', a
print 'max of a:', np.amax(a)
print 'average of a:', np.mean(a)
print 'standard deviation:', np.std(a)
print 'average along axis 1:', np.mean(a, 1)
```

```
array a
[[8 1 7]
 [3 9 2]]
max of a 9
average of a 5.0
standard deviation 3.10912635103
average along axis 1 [ 5.33333333  4.66666667]
```

Есть также возможность рассчитать корреляции и гистограммы для заданных выборок случайных величин.

3.3 Пакет Matplotlib

Научная графика, базовые возможности Matplotlib, рисование графиков, элементы оформления, 1D графики, 2D графики, трехмерная визуализация.

Научная графика

При анализе результатов расчетов повышенное внимание уделяется визуализации результатов вычислений, которое основывается на построение различных графиков. Многие расширения Python предоставляют пользователям мощные и гибкие средства построения графиков различной природы. Обычные для сегодняшнего дня программные средства постпроцессинга обеспечивают пользователю возможность работы с файлами данных, просмотра результатов в графическом виде, сохранения рисунка в выбранном стандартном графическом формате и печати графика.

Наиболее важными являются задачи визуализации одномерных или двумерных данных. С одномерными (1D) данными (графиками) мы связываем функцию одной переменной, с двумерными (2D) данными — функцию двух переменных. Отметим некоторые расширения Python, которые ориентированы прежде всего на создание качественных одномерных и многомерных графиков.

Основным программным продуктом для визуализации расчетных данных в Python является пакет Matplotlib[4]. Он закрывает основные потребности вычислителей, хорошо проработан и документирован, активно развивается и поддерживается. С учетом этих соображений будем рассматривать пакет Matplotlib как графическое ядро математического Python. Основные возможности пакета подробнее обсуждаются ниже.

Из традиционных средств подготовки графиков в научных вычислениях необходимо отметить Gnuplot[5]. Gnuplot имеет собственную систему команд, возможна интерактивная работа в режиме командной строки, и режиме интерпретации скриптов из файла (командный режим). Gnuplot выводит графики как непосредственно на экран, так и в файлы основных графических форматов, таких как png, eps, svg, jpeg. Поддерживается экспорт для издательской системы LaTeX. Работа с Gnuplot в Python поддерживается пакетом Gnuplot.py[6]. Он позволяет использовать Gnuplot в программах на языке Python для построения графиков для массивов данных из памяти, при чтении данных из файлов, а также графики математических функций.

Библиотека научной графики DISLIN[7] является бесплатной для некоммерческого использования. Она предназначена для отображения расчетных данных в виде кривых, гистограмм, диаграмм, контуров и поверхностей. Поддерживаются графические форматы PostScript, pdf, svg, png, bmp, gif, tiff и другие. DISLIN доступна для языков программирования Фортран 77, Фортран 90/95, C, Perl, Python и Java и работает на большинстве операционных систем.

[4] Matplotlib — http://matplotlib.sourceforge.net/
[5] Gnuplot — www.gnuplot.info/
[6] Gnuplot.py — http://gnuplot-py.sourceforge.net/
[7] DISLIN — www.dislin.de/

Кроссплатформенная библиотека научной графики MathGL[8] обеспечивает быструю обработку и отображение больших массивов данных. Реализованы основные типы графиков для одно-, двух- и трехмерных массивов с поддержкой экспорта в растровые (png, jpeg, tiff или bmp) и векторные (eps и svg) форматы. Библиотека MathGL может быть использована в программах, написанных на C++, C, Fortran, Python и других языках.

Для научных вычислений на языке Python будут интересны продукты компании Enthought, Inc[9]. Для научной визуализации предлагается два пакета: Chaco и Mayavi. Для 2D интерактивной визуализации применяется Chaco с широкой поддержкой различных графических форматов. Пакет Mayavi2 обеспечивает легкую интерактивную 3D визуализацию данных. Это достигается использованием функций визуализации, аналогичных тем, что есть в MATLAB и matplotlib, для отображения скалярных, векторных и тензорных двух- и трехмерных величин.

Базовые возможности Matplotlib

Приведем пример использования пакета Matplotlib при визуализации функции $sin(2\pi x)$, когда $0 \leq x \leq 1$. С учетом того, что мы уже знакомы с работой с массивами в рамках пакета NumPy, сформируем два одномерных массива по 100 элементов x и y. Для рисования графика используем модуль pyplot в пакете Matplotlib.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0., 1., 100)
y = np.sin(2*np.pi*x)
plt.plot(x, y)
plt.show()
```

Результатом работы этой программы будет окно с графиком (рис. 3.1).

Отметим прежде всего некоторые интерактивные возможности работы с графиком в этом диалоговом окне, которое вызывается командой `show()`. Оно содержит навигационную панель инструментов, функциональные возможности которой легко анализируются экспериментально. Вы можете перемещать график, изменять его размеры, выделять часть графика с помощью мыши или сочетания клавиш (клавиши быстрого вызова). Имеется также кнопка для сохранения графика в выбранном графическом формате. Вы можете получить файлы со следующими расширениями: `png`, `emf`, `ps`, `eps`, `svg`, `svgz`, `raw`, `rgba` и `pdf`.

Возможность интерактивной работы с графикой важна при отладке ваших программ, для выбора графического отчетного материала. Для окончатель-

[8] MathGL - http://mathgl.sourceforge.net/
[9] Enthought, Inc — www.enthought.com/

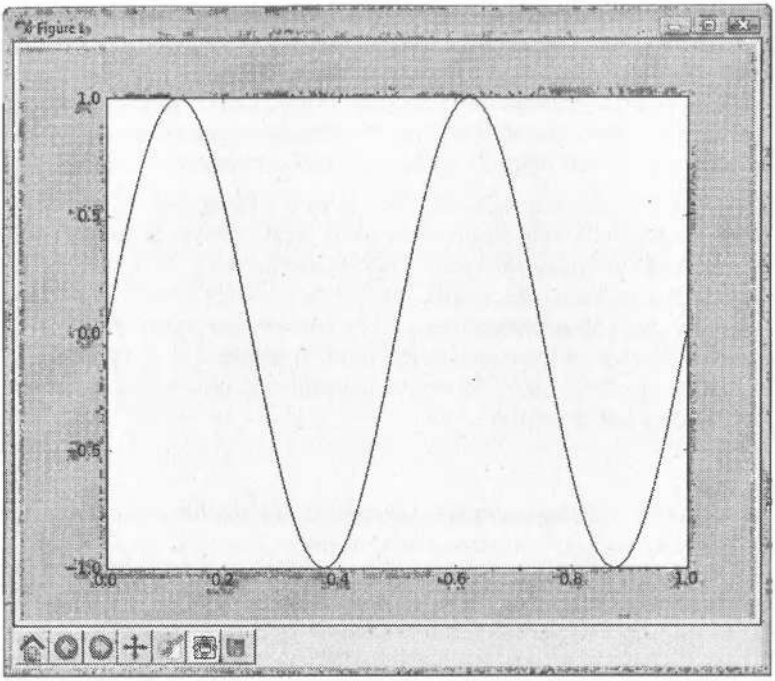

Рис. 3.1 Окно графика

ной версии программы большее значение придается получению результатов в виде файлов с графиками. В пакете Matplotlib такая возможность реализуется командой `savefig()`.

Первый обязательный аргумент `savefig()` есть имя файла. Файл записывается в одном из следующих растровых и векторных форматов (табл. 3.4). Другие необязательные параметры позволяют управлять выводом. В частности,

Таблица 3.4 *Поддерживаемые графические форматы*

Расширение	**Название**
emf	Windows Metafile
eps	Encapsulated PostScript
pdf	Portable Document Format
png	Portable Network Graphics
ps	PostScript
raw, rgba	Raw RGBA bitmap
svg, svgz	Scalable Vector Graphics

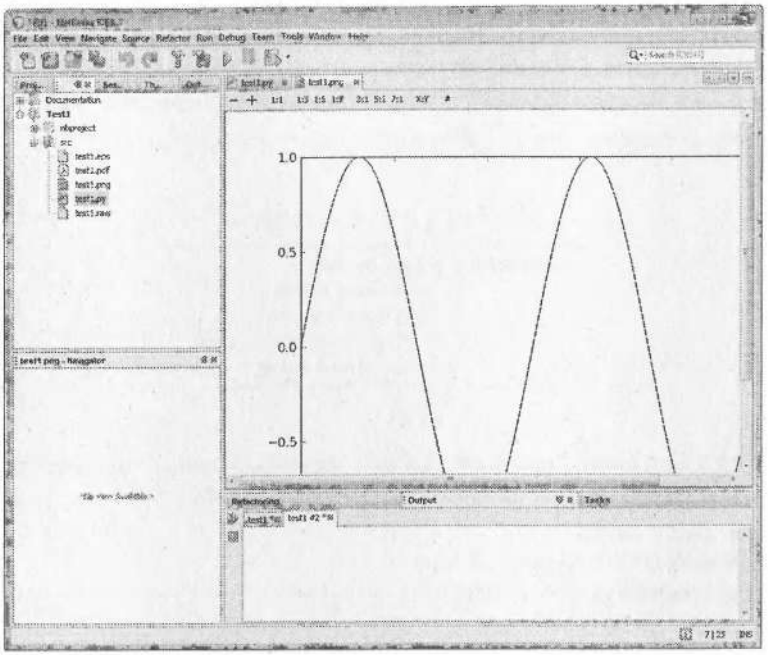

Рис. 3.2 Просмотр графика в NetBeans

можно изменять фон рисунка (параметр `facecolor`, по умолчанию равен 'w' — белый), выбирать графический формат (параметр `format`, по умолчанию — `png`) или разрешение (параметр `dpi`). Для работы с Windows метафайлами необходимо инсталлировать модуль pyemf[10], но проблемы с экспортом в `emf` остались. Можно рекомендовать использовать `eps` как векторный формат и `png` как растровый для подготовки рисунков.

При работе в NetBeans IDE удобно записывать графические данные в формате `png` в рабочий каталог проекта. Тогда результаты расчетов становятся доступны для просмотра (рис. 3.2). Тоже самое можно сказать и о графических `pdf` файлах при установленном плагине NetBeans-PDFViewer для просмотра `pdf` файлов.

Рисование графиков

Отметим некоторые возможности по заданию параметров графиков, которые рисуются с помощью основной функции `plot()`. С помощью задания аргу-

[10] pyemf — http://pyemf.sourceforge.net/

ментов этой функции мы можем управлять типом и толщиной линий, их цветом, использовать маркеры. Все возможности использования этой функции можно узнать по общей команде помощи (в нашем примере, `help(plt.plot)`).

Некоторые возможности по выбору типа линий отражены в табл. 3.5.

Таблица 3.5 *Линии графика*

Символ(ы)	Тип линий
'-'	сплошная линия
'--'	штриховая линия
'-.'	штрих-пунктирная линия
':'	пунктирная линия

На рис. 3.3 приведен результат работы нижеприведенной программы. По умолчанию (см. рис. 3.1) используется сплошная линия.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0., 1., 100)
y1 = np.sin(4*np.pi*x)
y2 = np.exp(x) - 1.
y3 = np.cos(4*np.pi*x)
plt.plot(x, y1, '-', x, y2, '--', x, y3, ':')
plt.show()
```

Для того чтобы показать отдельные точки на графике используются маркеры, некоторые из них приведены в табл. 3.6.

Таблица 3.6 *Маркеры графика*

Символ	Маркер	Символ	Маркер
'.'	жирная точка	's'	квадрат
'o'	круг	'p'	пятиконечная звезда
'v'	треугольник вершиной вниз	'*'	*
'^'	треугольник вершиной вверх	'+'	+
'<'	треугольник вершиной влево	'x'	×
'>'	треугольник вершиной вправо	'd'	ромб

Замена строки

```
plt.plot(x, y1, '-', x, y2, '--', x, y3, ':')
```

в программе рисования графика на рис. 3.3 на строку

```
plt.plot(x, y1, 'p', x, y2, 's', x, y3, 'd')
```

даст рис. 3.4.

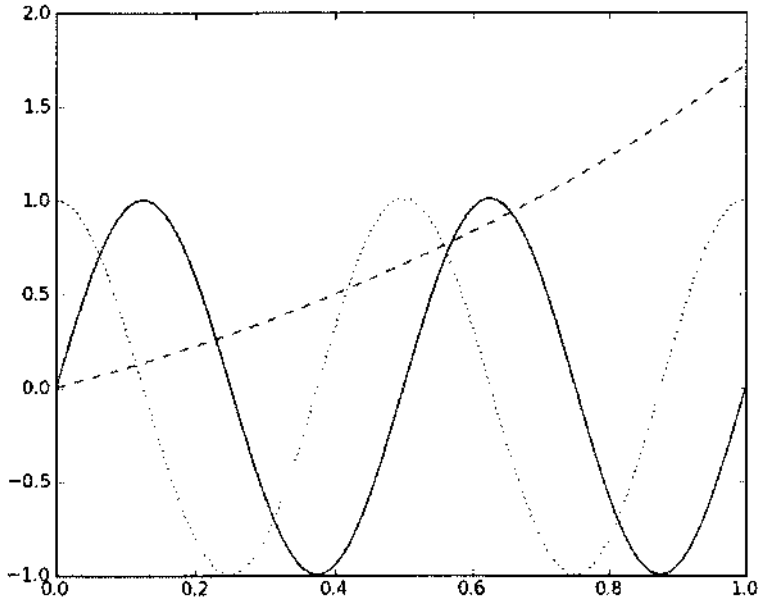

Рис. 3.3 Типы линий

При выводе графика можно использовать различные цвета линий (табл. 3.7). Комбинируя типы линий, маркеров с цветом мы можем получить необходимое представление для графика. Например, при использовании функции `plot(x, y, 'ro:')` график будет нарисован в виде пунктирной красной линии с круговыми маркерами.

Таблица 3.7 *Цвет графика*

Символ	Цвет	Символ	Цвет
`'b'`	синий (blue)	`'m'`	малиновый (magenta)
`'c'`	циановый (cyan)	`'r'`	красный (red)
`'g'`	зеленый (green)	`'w'`	белый (white)
`'k'`	черный (black)	`'y'`	желтый (yellow)

Изменить график можно используя различную толщину линий и размер маркера. Для этого задаются аргументы `linewidth` (`lw`) и `markersize` (`ms`) функции `plot()`, например, `plot(x, y, lw=2, ms=4)`, по умолчанию оба параметра равны 1.

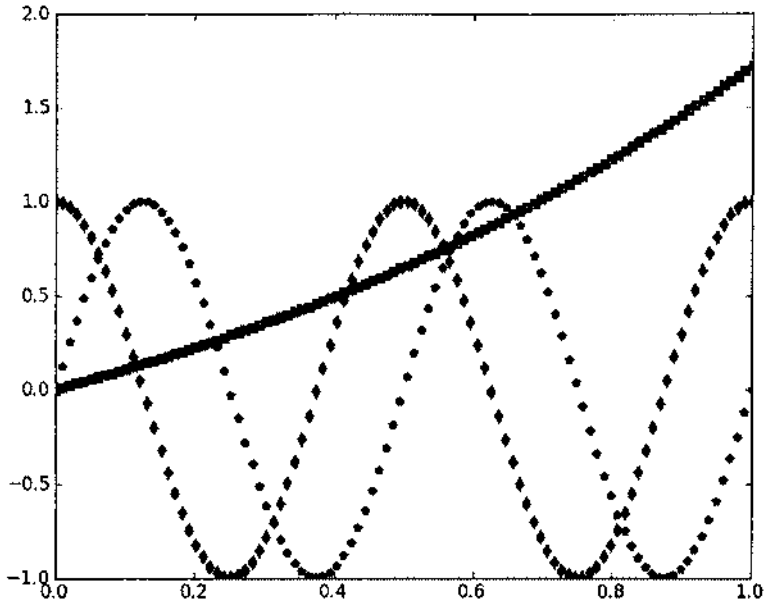

Рис. 3.4 Графики с маркерами

Элементы оформления

Дополнительную смысловую нагрузку графику придают его заголовок, название осей, координатная сетка и легенда. В пакете Matplotlib эти возможности поддерживаются функциями `title()`, `xlabel()`, `ylabel()`, `grid()`, `legend()`.

Функция `title()` помещает заголовок (строка с названием обязательный аргумент) на график. Можно использовать команды издательской системы LaTeXдля математического текста. Для поддержки русского языка в Matplotlib нужны специальные усилия, поэтому на первых порах лучше использовать английский язык. Для контроля размера шрифта используется параметр `fontsize` со значениями: `'large'`, `'medium'`, `'small'` либо явный размер. Положение заголовка по вертикали определяется параметром `verticalalignment` (va), который выбирается из: `'top'`, `'baseline'`, `'bottom'`, `'center'`. Положение по горизонтали определяется `horizontalalignment` (ha) со значениями `'center'`, `'left'`, `'right'`.

Для надписей на осях используются функции `xlabel()`, `ylabel()`. Функция `axis()` управляет диапазонами изменения по осям x и y. Новые диапазоны задаются командой `axis(xmin, xmax, ymin, ymax)`. Кроме этого функция

имеется возможность дополнительных манипуляций с осями. Например, команда `axis('off')` удаляет оси, `axis('equal')` — выравнивает диапазоны изменения по осям и т.д.

При рисовании нескольких графиков полезно идентифицировать отдельных графики с помощью легенды. Для добавления легенды используется функция `legend()` при задании аргумента `label` в виде строки в функции `plot()` для каждого графика. Вторая возможность связана с явным заданием меток в виде списка или кортежа строк как аргумента функции `legend()`. Локализация легенды проводится по аргументу `loc)` (см. табл. 3.8).

Таблица 3.8 *Положение легенды графика*

Срока	Код		
'best'	0	'center left'	6
'upper right'	1	'center right'	7
'upper left'	2	'lower center'	8
'lower left'	3	'upper center'	9
'lower right'	4	'center'	10
'right'	5		

Для отображения координатной сетки используется функция `grid()` с возможностью конкретизации свойств линий. Дополнительный контроль шага координатной сетки предоставляют функции `xticks()` и `yticks()`.

Ниже представлена программа, которая иллюстрирует некоторые возможности по оформлению графика (рис. 3.5).

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0., 1., 100)
y1 = np.sin(2*np.pi*x)
y2 = x*x
plt.plot(x, y1, '-', label='$sin(2\pi x)$')
plt.plot(x, y2, '--', label='$x^2$')
plt.title('The fuctions $sin(2\pi x)$ and $x^2$', \
          fontsize='large', va='bottom', ha='right')
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()
```

Выше мы рассмотрели возможности рисования одного графика. Отметим некоторые основные возможности более сложной организации графического материала. Свяжем интерактивное графическое окно или графический файл с фигурой (областью рисования) и будем рассматривать ее как контейнер графиков. В своей вычислительной практике нам может понадобиться

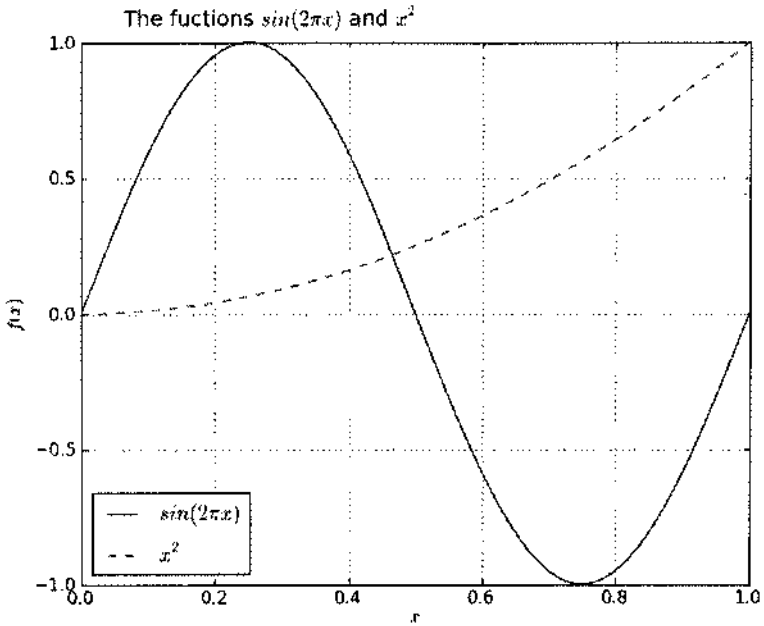

Рис. 3.5 Оформление графика

подготовить несколько фигур и каждая их фигур может включать несколько графиков.

Пример работы с двумя фигурами, показанный на рис. 3.6, реализован следующим образом.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0., 1., 100)
plt.figure(1)
plt.plot(x, x*x)
plt.figure(2)
plt.plot(x, x/(1+x))
plt.show()
```

Рассмотрим теперь проблему размещения нескольких графиков на одной фигуре. Первая возможность связана с использованием функции `subplot()`, которая разделяет фигуру на несколько прямоугольных областей равного размера, расположенных подобно элементам матрицы. Первый аргумент этой функции определяет количество строк (`numrows`), второй — количество колонок (`numcols`) в активной фигуре и третий — номер графика (`fignum`). Если

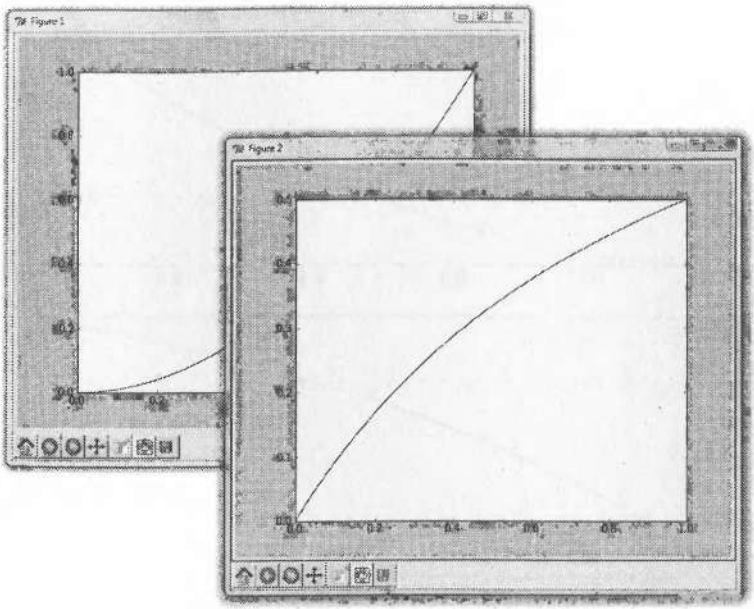

Рис. 3.6 Две фигуры

`numrows*numcols < 10` то запятые ставить не обязательно, так что вместо `subplot(1,2,1)` можно использовать `subplot(121)`.

На рис. 3.7 приведен результат работы программы:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0., 1., 100)
plt.figure(1)
plt.subplot(211)
plt.plot(x, x*x)
plt.subplot(212)
plt.plot(x, x/(1+x))
plt.show()
```

Имеется возможность ручного расположения графиков на фигуре. Для этого нужно воспользоваться функцией `axes()`. Аргументом этой функции является список `[left, bottom, width, height]`, который задает положение графика на фигуре в относительных координатах (все значения изменяются от 0 до 1). Эти возможности иллюстрируются программой

```
import numpy as np
```

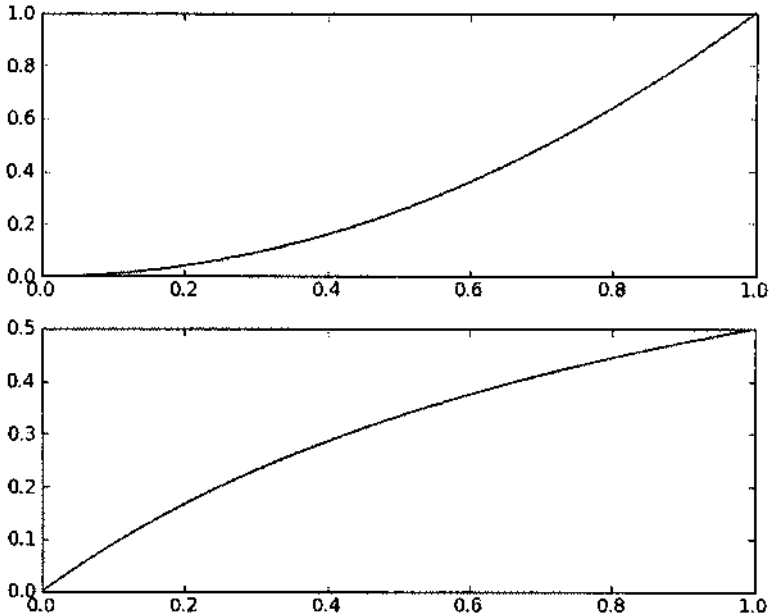

Рис. 3.7 Два графика

```
import matplotlib.pyplot as plt
x = np.linspace(0., 1., 100)
plt.figure(1)
plt.axes([0.05, 0.05, 0.6, 0.6])
plt.plot(x, x*x)
plt.axes([0.35, 0.35, 0.6, 0.6])
plt.plot(x, x/(1+x))
plt.show()
```

результатом работы которой является рис. 3.8.

1D графики

К базовым возможностям любого программного инструментария научной визуализации относятся графическое представлении одномерных и двумерных расчетных данных. Примеры использования функции `plot()`, которые приведены выше, демонстрировали возможность строить графики зависимостей $y = f(x)$ (одномерные функциональные зависимости). Остановимся на других возможностях 1D графики в пакете Matplotlib.

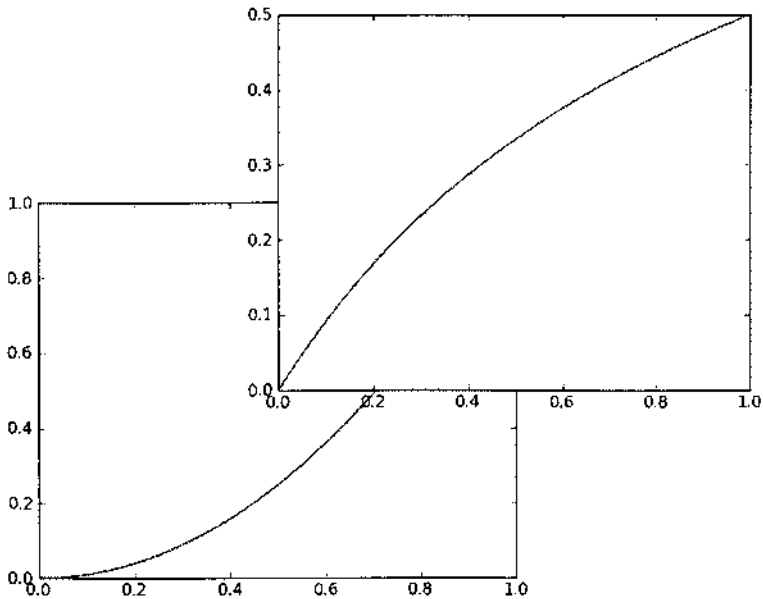

Рис. 3.8 Ручное задание положения графиков

Для отображения графиков величин, который заданы с погрешностью, используется функция `errorbar()`. После массива абсцисс и массива ординат в качестве третьего аргумента передается погрешность ординаты. Расширенные возможности включают погрешности в задании абсциссы. Пример использования функции `errorbar()` приведен ниже, результат — на рис. 3.9.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0., 1., 100)
y = np.exp(-x)
e = 0.1*np.random.normal(0, 1, 100)
plt.errorbar(x, y, e, fmt='.')
plt.show()
```

Простейший способ построения графиков в полярных координатах (ρ, φ) связан с использованием функции `polar()` вместо `plot()`. Первый аргумент этой функции угол φ, второй — ρ. Программа для рисования кардиоиды :

```
import numpy as np
import matplotlib.pyplot as plt
phi = np.linspace(0., 2*np.pi, 100)
ro = 5*(1 + np.cos(phi))
plt.axis('off')
```

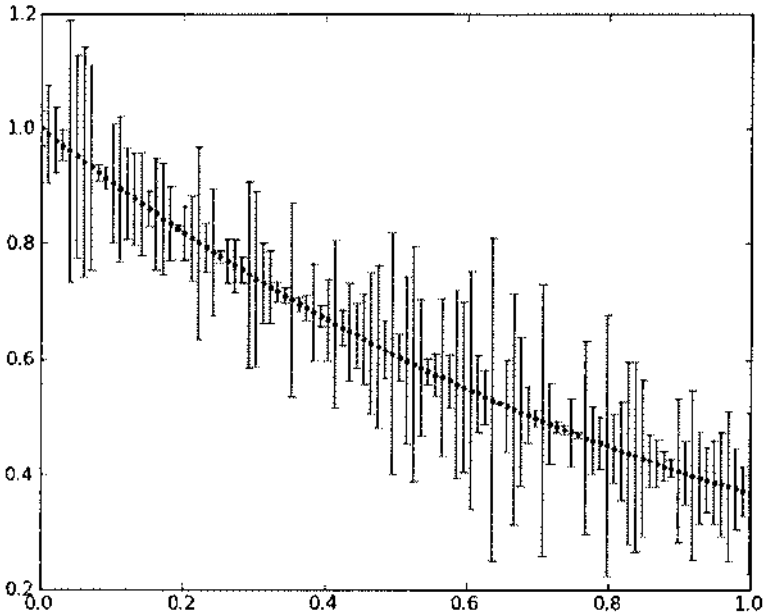

Рис. 3.9 График с погрешностями

```
plt.title('Cardioid')
plt.polar(phi, ro)
plt.show()
```

Функция `axis('off')` используется для того, чтобы не рисовать (рис. 3.10) декартовы оси, которые рисуются по умолчанию.

Для создания более обозримого графика вдоль одной или обеих координатных осей можно выбирать логарифмический масштаб. Для построения таких графиков вместо `plot()` используются функции `loglog()` (логарифмический масштаб по обеим осям), `semilogx()` (логарифмический масштаб по оси x), `semilogy` (логарифмический масштаб по оси y). Примером является график на рис. 3.11, который получен с использованием следующей программы.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0., 1., 100)
y = np.exp(-10*x*x)
plt.semilogy(x,y)
plt.show()
```

Для графическое изображение зависимости частоты попадания элементов выборки от соответствующего интервала группировки используются гисто-

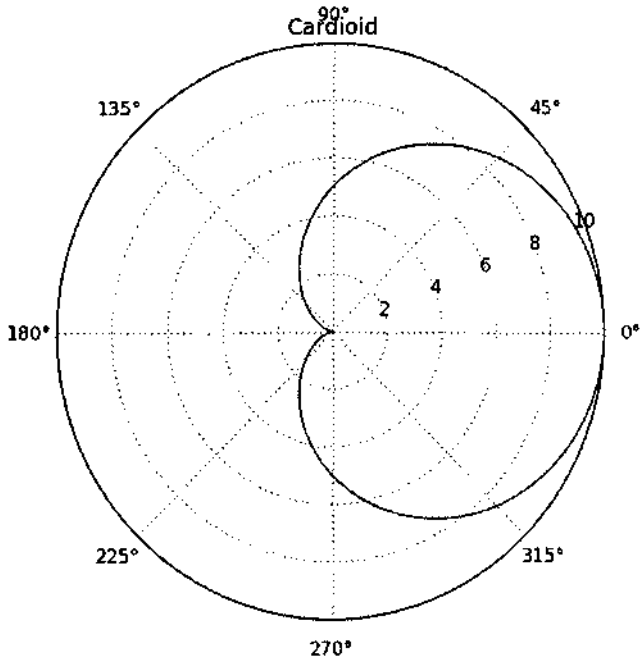

Рис. 3.10 Полярные координаты

граммы. Возможность построения гистограмм реализуется функцией `hist()`, первый аргумент которой есть массив данных, а второй задает число интервалов. Здесь приведен пример трех гистограмм для выборки из 250, 500 и 1000 случайных нормально распределенных величин соответственно (рис. 3.12).

```
import numpy as np
import matplotlib.pyplot as plt
mu = 1
sigma = 0.2
x = mu + sigma*np.random.randn(1000)
plt.hist(x, 25)
plt.show()
```

Иногда приходиться используют для визуализации результатов различного рода диаграммы — столбцовые и круговые. Для рисования столбцовых диаграмм используется функция `bar()` (`barh()` для горизонтальных). Пример использования (см. рис. 3.13) иллюстрируется программой:

```
import numpy as np
import matplotlib.pyplot as plt
pos = np.linspace(0., 1., 10)
```

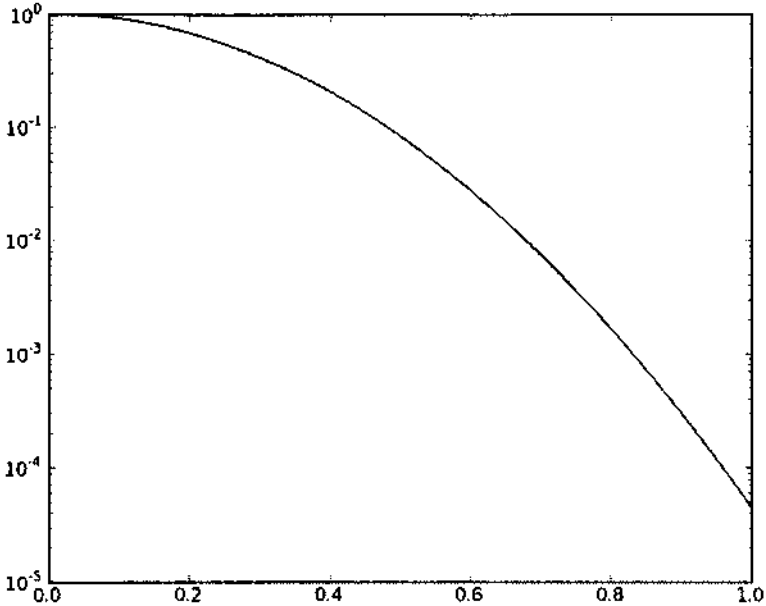

Рис. 3.11 Логарифмический масштаб

```
val = 3 + np.random.randn(10)
plt.bar(pos, val, align='center', width=0.1, color='m')
plt.show()
```

Подобные инструменты характерны для деловой графики при небольшом объеме представляемых данных и не имеют большого значения для научных вычислений.

2D графики

При проведении численных расчетов большое внимание уделяется визуализации двумерных данных. На этой основе реализуются те или иные инструменты для представления и более общих трехмерных данных, используя визуализацию на отдельных поверхностях и сечениях. Здесь мы отметим основные возможности построения 2D графиков в пакете Matplotlib.

Рассматривается задача построения графика двумерной функции $z = f(x, y)$. Считается, что значения заданы на прямоугольной сетке, т.е. двумерный массив (матрица) размерности $m \times n$. Массивы x и y могут быть векторами размерности n и m соответственно или матрицами (с одинаковыми строками или столбцами) той же размерности, что и z.

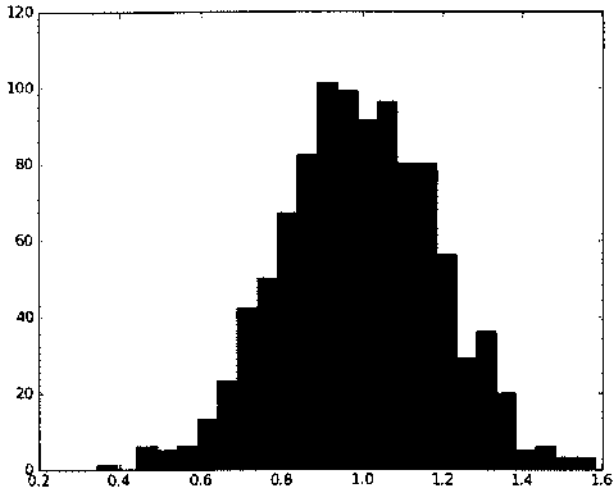

Рис. 3.12 Гистограмма

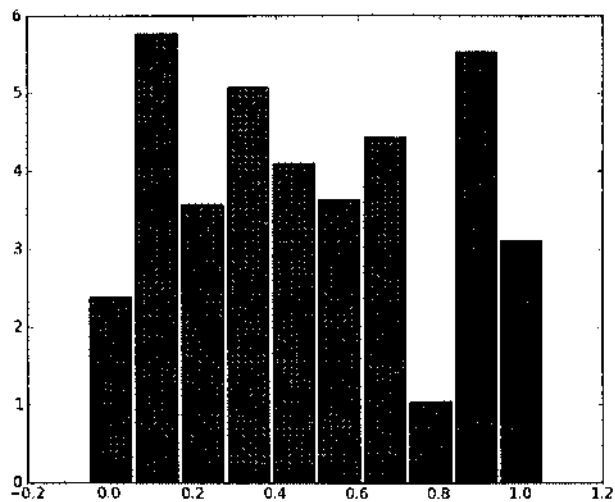

Рис. 3.13 Столбцовая диаграмма

Линиями уровня (изолинии) представляют собой кривые, которые получаются в результате пересечения поверхности $z = f(x, y)$ с плоскостями $z = \text{const}$. Для их построения используется функция `contour()`. Стандартный вызов

этой функции связан с заданием массивов x, y, которые определяют сетку, затем массив значений функции z и число уровней.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-2., 2., 101)
y = np.linspace(-2., 2., 101)
X, Y = np.meshgrid(x, y)
z = X * np.exp(-X**2 - Y**2)
plt.contour(x, y, z, 20)
plt.show()
```

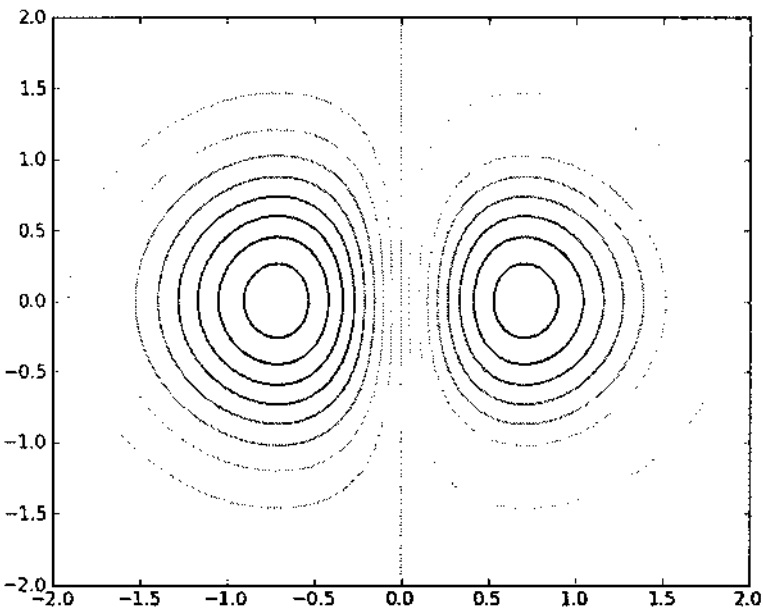

Рис. 3.14 Линии уровня

Результат работы этой программы приведен на рис. 3.14. Здесь функция `meshgrid()` задает сетку на плоскости x, y в виде двумерных массивов X, Y, которые определяются одномерными массивами x и y. Применяя операции над массивами, с помощью этой функции легко вычисляются функции двух переменных.

Можно отобразить значения линий уровня. Для этого используется функция `clabel()`, первым аргументом которой является результат `ContourSet` объект, который возвращается `contour`. Пример работы иллюстрируется рис. 3.15, который мы получаем при программы:

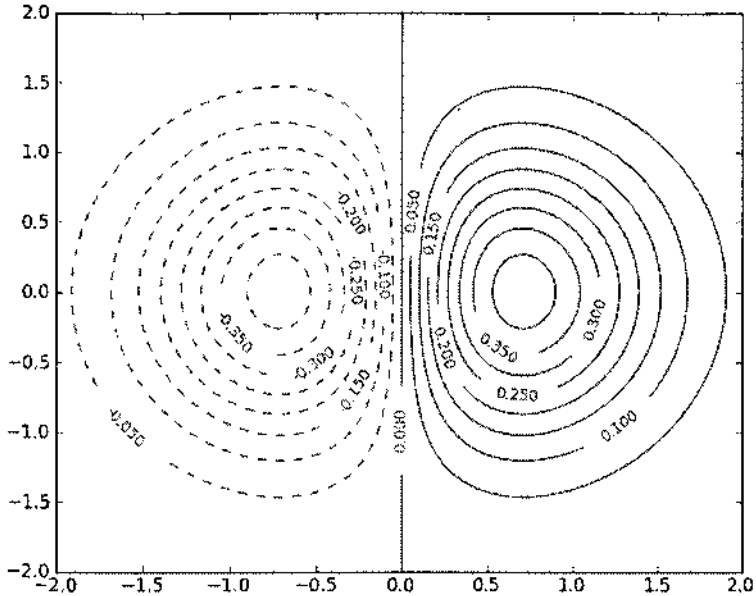

Рис. 3.15 Управление линиями уровня

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-2., 2., 101)
y = np.linspace(-2., 2., 101)
X, Y = np.meshgrid(x, y)
z = X * np.exp(-X**2 - Y**2)
v = np.linspace(-0.5, 0.5, 21)
cs = plt.contour(x, y, z, v, colors='g')
plt.clabel(cs, inline=True, fontsize=10)
plt.show()
```

Для выбора цвета изолиний (см. табл. 3.7) служит параметр `colors`. Для отображения изолиний с отрицательными значениями используются штриховые линии. В данном примере значения изолиний задаются явно (массив `v`).

Для подготовки графиков с заливкой областей между линиями уровня нужно воспользоваться функцией `contourf()`. Пример использования показан на рис. 3.16.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-2., 2., 101)
```

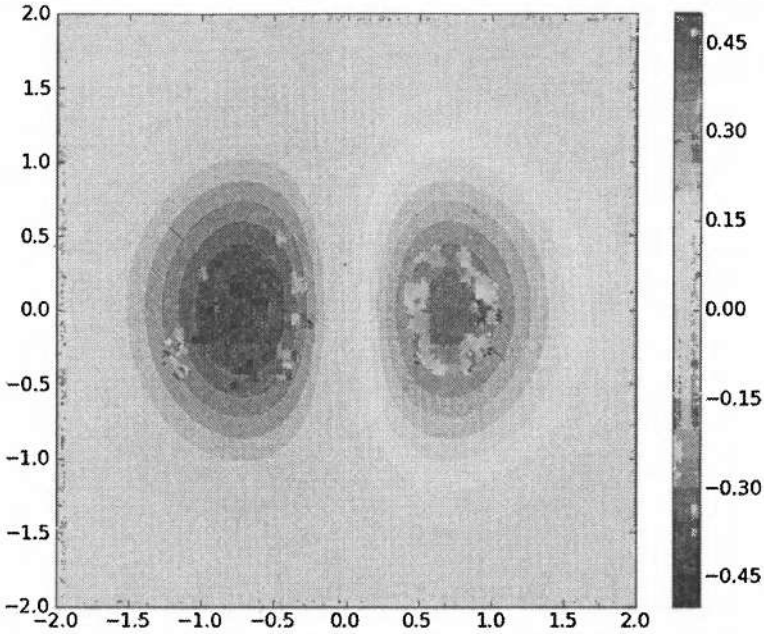

Рис. 3.16 Линии уровня с заливкой

Таблица 3.9 *Палитры*

Название	Описание
'autumn'	оттенки красного и желтого цветов
'cool'	оттенки голубого и фиолетового цветов
'flag'	чередование красного, белого, синего и черного цветов
'gray'	линейная палитра в оттенках серого цвета
'hot'	чередование черного, красного, желтого и белого цветов
'hsv'	цвета радуги
'pink'	розовые цвета с оттенками пастели
'spectral'	спектральная палитра
'spring'	оттенки желтого и фиолетового цветов
'summer'	оттенки зеленого и желтого цветов
'winter'	оттенки синего и зеленого цветов

```
y = np.linspace(-2., 2., 101)
X, Y = np.meshgrid(x, y)
z = X * np.exp(-X**2 - Y**2)
v = np.linspace(-0.5, 0.5, 21)
plt.contourf(x, y, z, v)
plt.colorbar()
plt.show()
```

Шкала палитры отображена со значениями на линиях отображается помощью `colorbar()`. В пакете Matplotlib используются различные палитры (некоторые из них приведены в табл. 3.9, по умолчанию используется `hsv`).

Иногда при вычислениях нужно отобразить матрицу. Для этой цели можно использовать различные инструменты (`imshow()`, `matshow()`). В примере

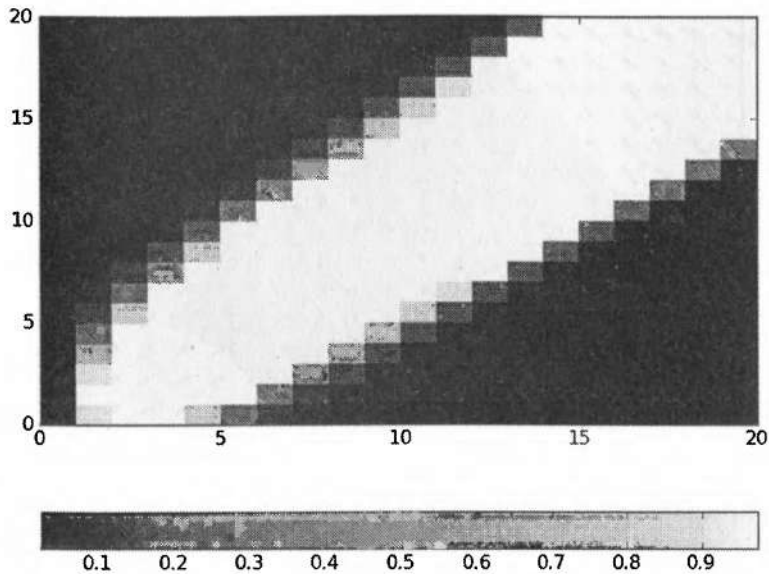

Рис. 3.17 Графическое отображение матрицы

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 20, 21)
y = np.linspace(0, 20, 21)
X, Y = np.meshgrid(x, y)
z = X * np.exp(-0.1*(X-Y)**2)
plt.pcolor(x, y, z, vmin=0.025, vmax=0.975, cmap='gray')
plt.colorbar(orientation='horizontal')
plt.show()
```

используется функция `pcolor()` с выбором палитры (рис. 3.17).

Проиллюстрируем также минимальные возможности пакета Matplotlib по отображению векторных полей. По компонентам вектора в каждом узле рисуются стрелка, направление которой совпадает с направлением вектора, а длина — пропорциональна длине вектора. Такое представление реализуется

функцией `quiver()`. Пример использования дается следующим листингом программы и рис. 3.18).

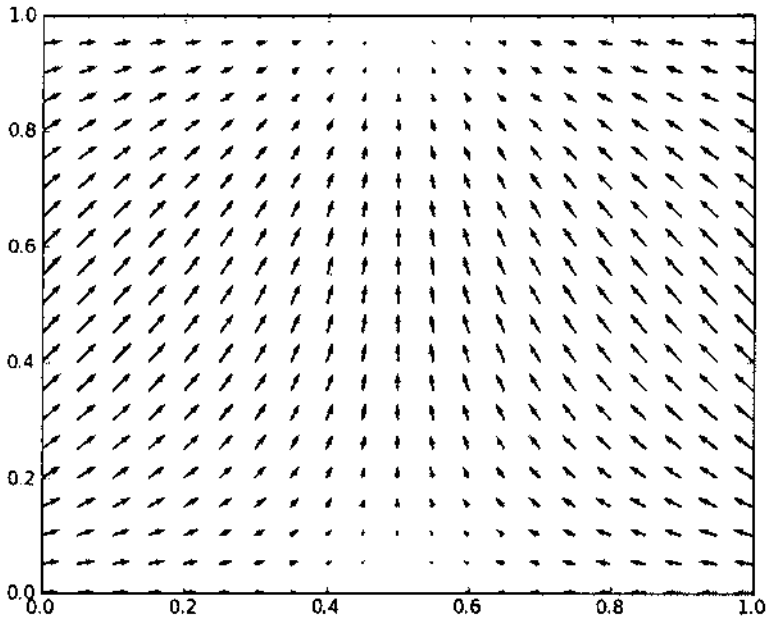

Рис. 3.18 Векторное поле

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0., 1., 21)
y = np.linspace(0., 1., 21)
print x
X,Y = np.meshgrid(x, y)
U = np.cos(np.pi*X)
V = np.sin(np.pi*Y)
plt.quiver(x, y, U, V)
plt.show()
```

Выше мы привели примеры, когда проводится визуализация данных на прямоугольной сетке. Отметим некоторые возможности построения двумерных графиков в более общих условиях.

Изолинии можно строить на произвольных структурированных сетках, когда сетка топологически эквивалентна прямоугольной В этом случае не только сама функция, но и каждая координата узлов сетки представляются в

виде двумерного массива. Для примера рассмотрим представление решения (рис. 3.19), которое строиться на равномерной цилиндрической сетке.

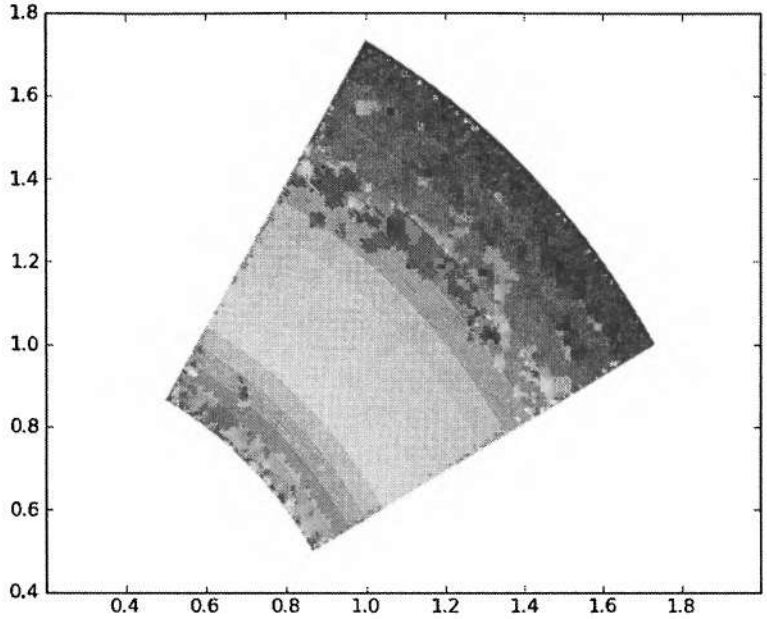

Рис. 3.19 Полярная сетка

```
import matplotlib.pyplot as plt
import numpy as np
ro = np.linspace(1., 2., 21)
phi = np.linspace(np.pi/6, np.pi/3, 21)
Ro, Phi = np.meshgrid(ro, phi)
X = Ro*np.cos(Phi)
Y = Ro*np.sin(Phi)
Z = np.exp(-Ro)
plt.contourf(X, Y, Z, 20)
plt.axis('equal')
plt.show()
```

Для данных на произвольном множестве узлов задача визуализации в пакете Matplotlib решается следующим образом. Сначала по заданному множеству узлов строиться триангуляция Делоне и данные интерполируются на структурированную сетку, после чего и рисуется график. Задача интерполяции решается с использованием функции `griddata()` из модуля `mlab`. Другая воз-

можность связана с использованием расширения Natgrid (mpl_toolkits.natgrid инсталлируется отдельно от пакета Matplotlib).

Пусть x, y, z есть одномерные массивы координат x, y и заданных значений интерполируемой функции z. Прямоугольная сетка задастся одномерными или двумерными массивами X и Y как обычно. Двумерный массив значений на этой новой сетке определяется функцией griddata(x, y, z, X, Y). Пример визуализации данных на нерегулярном множестве узлов дает программа:

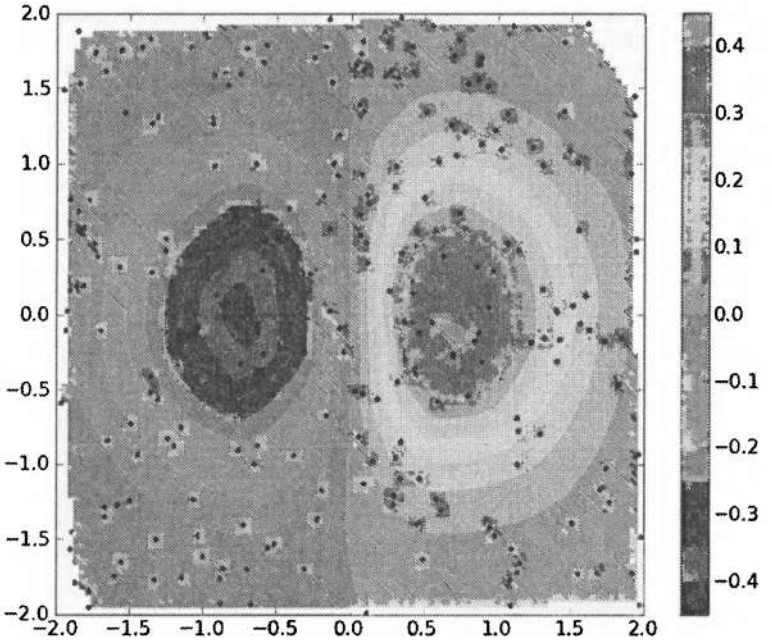

Рис. 3.20 Нерегулярное множество узлов

```
from matplotlib.mlab import griddata
import matplotlib.pyplot as plt
import numpy as np
x = np.random.uniform(-2, 2, 250)
y = np.random.uniform(-2, 2, 250)
z = x*np.exp(-x**2 - y**2)
X = np.linspace(-2., 2., 100)
Y = np.linspace(-2., 2., 100)
Z = griddata(x, y, z, X, Y)
plt.contourf(X, Y, Z, 20, cmap = plt.cm.spectral)
plt.colorbar()
plt.scatter(x, y, marker='o', c='k', s=5)
```

```
plt.xlim(-2,2)
plt.ylim(-2,2)
plt.show()
```

Результат работы программы представлен на рис. 3.20. Здесь функция `scatter` используется для нанесения на график маркеров выбранного размера и цвета (параметры `s` и `c`) в заданных точках.

Трехмерная визуализация

При визуализации двумерных данных значение функции в отдельных точках плоскости x, y мы связывали с цветом, отображая данные на плоскость. Вторая возможность связана с привязкой значения двумерной функции к третьей декартовой координате z. В этом случае объект визуализации есть поверхность, которая является существенно трехмерным объектом — трехмерное представление двумерной функции.

Возможности работы с трехмерными объектами визуализации поддерживаются в модуле mplot3D пакета Matplotlib. Прежде чем обсуждать визуализацию двумерных функций рассмотрим более простые объекты.

Пример рисования отдельных точек (функция `scatter()`) и параметрической кривой в пространстве (функция `plot()`) показан на рис. 3.21. В этом примере использовалась программа

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)
x = np.random.sample(100)
y = np.random.sample(100)
z = np.random.sample(100)
ax.scatter(x, y, z)
theta = np.linspace(0., 4 * np.pi, 100)
x1 = 0.5 * (1 + np.sin(theta))
y1 = 0.5 * (1 + np.cos(theta))
z1 = np.linspace(-2, 2, 100)
ax.plot(x1, y1, z1)
plt.show()
```

Параметры функций `plot()` и `scatter()` близки к тем, которые есть в аналогичных функциях модуля pyplot. Открывающееся интерактивное окно имеет дополнительные возможности по выбору точки зрения на отображаемый объект за счет вращения и изменения масштабов с помощью мыши.

Для отображения каркасной поверхности (wire frame) используется функция `plot_wireframe()`, аргументами которой являются двумерные массивы коор-

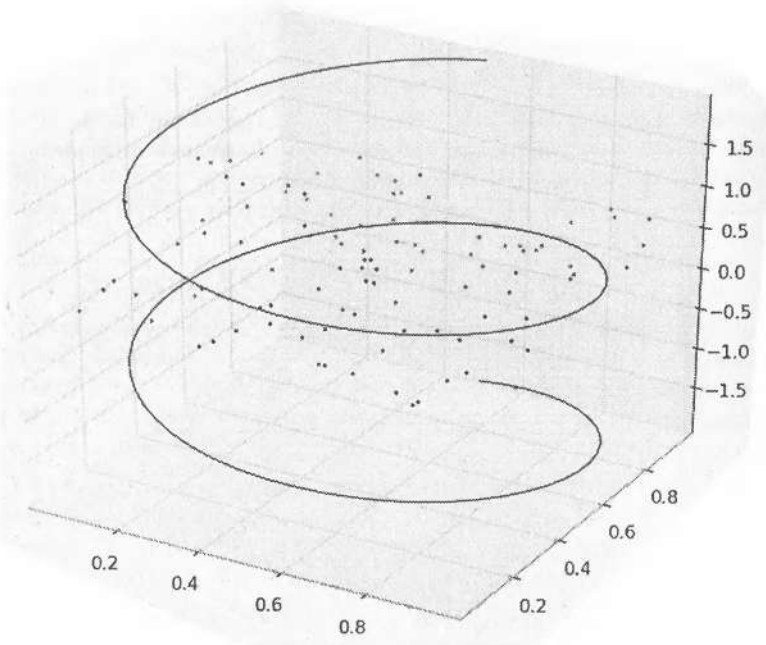

Рис. 3.21 Точки и кривая в пространстве

динат отображаемой поверхности. Тестовая программа дается ниже, результат ее работы показан на рис. 3.22 (рисуется каждая пятый элемент массива по каждому направлению).

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)
x = np.linspace(-2., 2., 101)
y = np.linspace(-2., 2., 101)
X, Y = np.meshgrid(x, y)
Z = X * np.exp(-X**2 - Y**2)
ax.plot_wireframe(X, Y, Z, rstride=5, cstride=5)
plt.show()
```

Для более выразительной визуализации двумерных расчетных данных часто комбинируют привязывание как цвета так и высоты со значениями функции. В этом случае график строиться с помощью функции `plot_surface()` с заливкой четырехугольников каркаса. Результат работы программы

```
import numpy as np
```

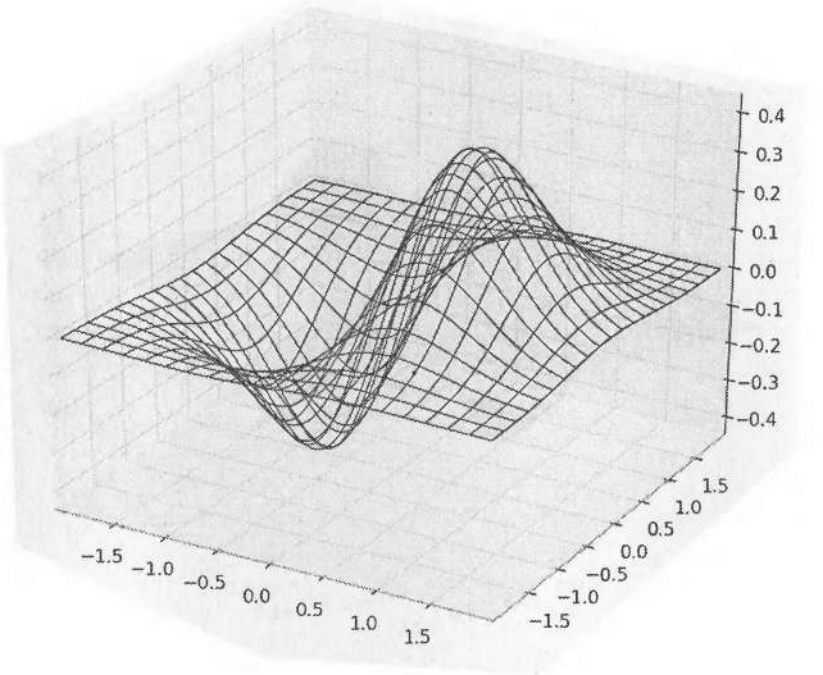

Рис. 3.22 Каркасное представление

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)
x = np.linspace(-2., 2., 101)
y = np.linspace(-2., 2., 101)
X, Y = np.meshgrid(x, y)
Z = X * np.exp(-X**2 - Y**2)
ax.plot_surface(X, Y, Z, rstride=2, cstride=2, cmap='spectral')
plt.show()
```

представлен на рис. 3.23. Здесь конкретизируется выбор палитры аналогично соответствующим функциям модуля pyplot.

Отдельного упоминания заслуживают возможности рисования трехмерных изолиний, которые предоставляются модулем mplot3D. Используются функции `contour()` и `contourf()`, которые вполне аналогичны соответствующим функциям модуля pyplot. Пример использования дает (рис. 3.24) программа

```
import numpy as np
import matplotlib.pyplot as plt
```

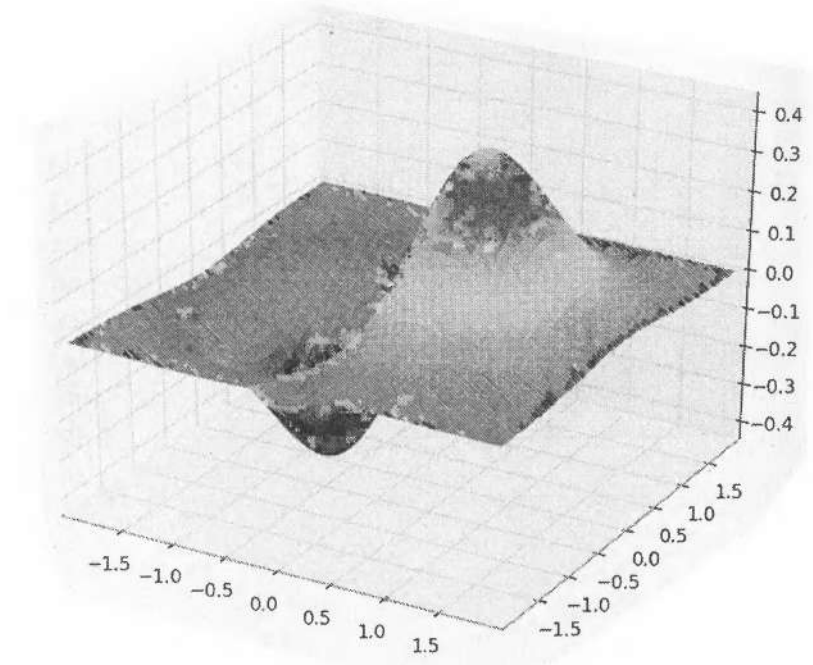

Рис. 3.23 Поверхность

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)
x = np.linspace(-2., 2., 101)
y = np.linspace(-2., 2., 101)
X, Y = np.meshgrid(x, y)
Z = X * np.exp(-X**2 - Y**2)
ax.contourf(X, Y, Z, 15)
plt.show()
```

В целом, пакет Matplotlib) удовлетворяет базовые потребности научной визуализации и на его основе может строится программное обеспечение по постпроцессингу данных расчетов.

3.4 Пакет SciPy

Научные вычисления на Python, общая характеристика, специальные функции, быстрое преобразование Фурье, линейная алгебра, интерполяция, задачи оптимизации, интегрирование и ОДУ.

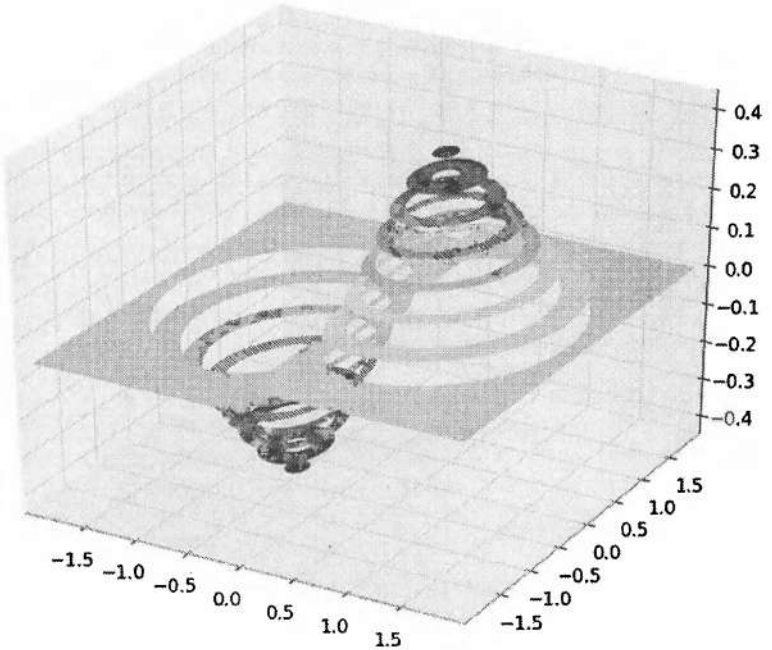

Рис. 3.24 Трехмерные контура

Научные вычисления на Python

Прикладное математическое моделирование проводится на основе численного исследования математических моделей. Математические модели включают в себя нелинейные системы уравнений с частными производными, дополненными системами дифференциальных и алгебраических уравнений. Их приближенное решение основывается на базовых алгоритмах численного анализа, среди которых можно отметить методы решения систем линейных и нелинейных уравнений и алгоритмы аппроксимации.

Среди расширений Python[11] есть много инструментов, которые ориентированы на решение задач вычислительной математики. Из коллекций Python, которые предназначены для научных вычислений, отметим SciPy[12]. Более подробное обсуждение этого пакета дается ниже.

Другой набор модулей Python для научных вычислений известен как ScientificPython[13]. Он включает в себя средства для работы векторами, тензорами и кватернионами, поддерживает преобразования векторных и тензорных по-

[11] Python — http://pypi.python.org/pypi/
[12] SciPy — http://scipy.org/
[13] ScientificPython — http://dirac.cnrs-orleans.fr/plone/software/scientificpython/

лей, автоматическое дифференцирование, интерполяцию, работу с полиномами, задачи элементарной статистики и задачи минимизации, имеется также интерфейсы для MPI (Message Passing Interface), двумерная и трехмерная визуализация. Большая часть этих функций включена в SciPy.

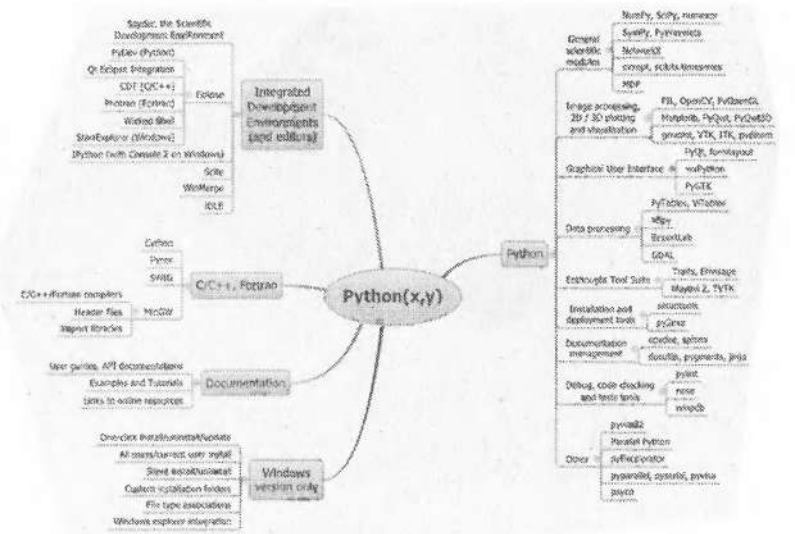

Рис. 3.25 Python(x,y) компоненты

Наверное самая большая и довольно таки разрозненная коллекция Python(x,y)[14] вобрала с себя все основные наработки для научного и инженерного программирования на языке Python. Общее представление о составе коллекции даст рис. 3.25. Она включает не только все основные математические и графические пакеты, но и, например, несколько интегрированных средств разработки, инструменты поддержки разработки графического пользовательского интерфейса, работы с языками C/C++ и Fortran и поддержки вычислений на параллельных вычислительных системах.

Отметим также программное обеспечение, которое ориентировано на решение специальных задач научных вычислений, которое успешно дополнит тот же пакет SciPy. Например, пакет Numexpr[15] работает с массивами существенно быстрее, чем NumPy. В качестве второго примера хорошего дополнения базового пакета научных вычислений SciPy, отметим PyTrilinos[16]. PyTrilinos обеспечивает интерфейс к коллекции библиотек Trilinos, которые решают, в частности, задачи построения и использование векторов, разреженных и

[14] Python(x,y) — http://pythonxy.com/
[15] Numexpr — http://code.google.com/p/numexpr/
[16] PyTrilinos — http://trilinos.sandia.gov/packages/pytrilinos/

плотных матриц, численного решения систем линейных уравнений прямыми и итерационными методами с учетом параллельной архитектуры вычислительной системы.

Для численного решения краевых задач для обыкновенных дифференциальных уравнений можно использовать bvp[17]. Краевые задачи могут решаться методом конечных элементов с использованием SfePy[18]. Это программное обеспечение позволяет решать двумерные и трехмерные задачи механики сплошной среды (теплопроводность, упругость, уравнения Навье—Стокса).

Общая характеристика

Пакет SciPy является программным обеспечением с открытым исходным кодом для математики, естественных наук и инженерии. SciPy обеспечивает удобную и быструю работу с N-мерными массивами NumPy. Кроме этого, пакет SciPy предоставляет большое число эффективных вычислительных процедур численного анализа, такие как программы для численного интегрирования и оптимизация.

Как мы уже отмечали выше, пакет NumPy предоставляет широкие возможности работы с важнейшим для вычислений объектом – N-мерным массивом. Имеются мощные инструменты работы с этими объектами, такие как создание, перестройка, выбор отдельных элементов и групп. Кроме этого пакет NumPy содержит модули численного анализа, которые связаны с задачами линейной алгебры, реализуют быстрое преобразование Фурье и работу со случайными числами.

Сам пакет SciPy организован как коллекции пакетов или, как говорят, subpackages, которые обеспечивают научные вычисления в различных направлениях. С учетом принятой в Python терминологии для программного обеспечения будем говорить об этих пакетах как о модулях пакета SciPy. Именно в этом контексте, в частности, пакет NumPy мы рассматриваем как модуль пакета SciPy.

В состав пакета SciPy входят следующие (в алфавитном порядке) модули.

Clustering package (scipy.cluster). Предоставляются[19] функции кластерного анализа, который связан с разбиением заданной выборки объектов (ситуаций) на непересекающиеся подмножества, называемые кластерами, при условии, что каждый кластер состоит из схожих объектов, а объекты разных кластеров существенно различаются. Задача кластеризации относится к статистической обработке и включает иерархическую генерацию кластеров по расстоянию между объектами, вычисление матрицы расстояний от вектора наблюдений, подсчет

[17] bvpm — http://elisanet.fi/ptvirtan/software/bvp/
[18] SfePy — http://sfepy.kme.zcu.cz/
[19] scipy-cluster — http://code.google.com/p/scipy-cluster/

статистики по группам, визуализацию кластеров с помощью дендрограмм.

Constants (scipy.constants). Обеспечивается набор физических констант и функции перевода значений из одних единиц измерения в другие. Реализован международно принятый набор значений фундаментальных физических констант[20] предоставляется CODATA (Committee on Data for Science and Technology). Доступны значения, единицы измерения и относительная точность.

Fourier transforms (scipy.fftpack). Модуль предназначен для проведения прямого и обратного быстрого преобразования Фурье (БПФ) с целью получения спектра сигналов и восстановления формы сигналов по их спектрам. Дискретное преобразование Фурье трансформирует последовательность комплексных (либо вещественных) чисел в последовательность комплексных чисел. Реализация основана на основе широко известного и популярного пакета FFTPACK[21].

Integration and ODEs (scipy.integrate). Представлены функции для вычисления интегралов при функциональном и табличном задании подынтегральной функции. Вторая часть модуля integrate предназначена для численного решения задачи Коши для системы обыкновенных дифференциальных уравнений. Программная реализация базируется на FORTRAN пакете ODEPACK[22].

Interpolation (scipy.interpolate). В модуле interpolate реализованы алгоритмы интерполяции и сглаживания одномерных и двумерных функций. Особое внимание уделяется аппроксимации и сглаживанию с использованием кубических сплайнов. Используются многие возможности пакета программ на FORTRAN FITPACK[23].

Input and output (scipy.io). Ввод/вывод данных в пакете SciPy дополняет базовые возможности чтения и записи пакета NumPy. В NumPy поддерживаются чтение и запись массивов, которые являются основными объектами при вычислениях. В пакете SciPy реализованы некоторые возможности ввода/вывода для различных других форматов (данные, аудио, видео, изображения, MATLAB и т.д.).

Linear algebra (scipy.linalg). Пакет NumPy обеспечивает решение основных задач линейной алгебры. Пакет SciPy, построенный на основе библиотек линейной алгебры ATLAS LAPACK и BLAS, дает новые возможности. В частности, при работе с матричными объектами поддерживаются вычисления с функциями матриц.

Maximum entropy models (scipy.maxentropy). Содержит процедуры подбора моделей максимальной энтропии в теории информации.

[20] CODATA constants — http://physics.nist.gov/cuu/Constants/
[21] FFTPACK — http://netlib.org/fftpack/
[22] ODEPACK — https://computation.llnl.gov/casc/odepack/
[23] FITPACK — http://www.netlib.org/fitpack/

Miscellaneous routines (scipy.misc). Представлены небольшое число различных полезных функций, которые не отнесены к специализированным модулям пакета SciPy.

Multi-dimensional image processing (scipy.ndimage). Модуль предназначен для анализа и обработки изображений на основе операций над массивами значений. Представлены функции для линейной и нелинейной фильтрации, имеются инструменты для решения задач интерполяции и бинарной морфологии.

Orthogonal distance regression (scipy.odr). Метод наименьших квадратов используется для нахождения параметров модели при условии, что имеются ошибки в зависимых переменных. В модуле odr представлены эффективные алгоритмы оценивания параметров в более общей ситуации, когда ошибки вносятся и в независимые переменные. Реализация основана на пакете ODRPACK[24]

Optimization and root finding (scipy.optimize). Модуль optimize обеспечивает большую часть стандартных алгоритмов оптимизации, задач о нахождении экстремума (минимума или максимума) вещественной функции в некоторой области. Реализованы основные алгоритмы решения задач условной и безусловной оптимизации. Модуль также включает функции нахождения решений систем нелинейных уравнений.

Signal processing (scipy.signal). Представлены функции обработки сигналов, которые понимаются как массивы действительных или комплексных чисел. Модуль содержит некоторые функции для решения задач фильтрации сигналов, а также инструменты для разработки фильтров. Интерполяции для одно- и двумерных данных обеспечивается использованием B-сплайнов.

Sparse matrices (scipy.sparse). К разреженными относятся матрицы с преимущественно нулевыми элементами. Такие типы матриц типичны для научных вычислений при численном решении краевых задач для дифференциальных уравнений. В модуле sparse реализованы функции создания, хранения и трансформаций разреженных матриц. Поддерживаются различные структуры данных, которые включают, в частности, стандартные CSR (Compressed Sparse Row) и CSC (Compressed Sparse Column) форматы для разреженных матриц.

Sparse linear algebra (scipy.sparse.linalg). В модуле sparse.linalg пакета SciPy представлены функции, с помощью которых решаются основные задачи линейной алгебры. Для разреженных матриц реализованы алгоритмы вычисления собственных значений и собственных функций, прямые и основные итерационные методы решения систем уравнений с симметричными и несимметричными разреженными матрицами.

Spatial algorithms and data structures (scipy.spatial). Предназначен для работы с пространственными структурами данных и алгоритма-

[24] ODRPACK — http://netlib.org/odrpack/

ми. Модуль spatial содержит функции работы с KD-деревьями, в частности, для нахождение ближайшего соседа, примерного соседа и поиск всех ближайших пар.

Special functions (scipy.special). Модуль special обеспечивает работу пакета SciPy с многими специальными функциями математической физики (функции Эйри, Бесселя, гипергеометрические функции и т.д.).

Statistical functions (scipy.stats). Модуль stats пакета SciPy имеет большое количество основных процедур обработки статистических данных. Реализованы множество генераторов случайных величин как для дискретных и для непрерывных распределений.

Image Array Manipulation and Convolution (scipy.stsci). Поддерживаются вычисления средних и экстремальных величин, свертки для массивов изображений.

C/C++ integration (scipy.weave). С помощью модуля weave обеспечивается включение кода C/C++ внутри кода Python, что позволяет существенно ускорить вычисления.

Многие из модулей пакета SciPy непосредственно ориентированы на научные вычисления, связанные, прежде всего, с решением задач вычислительной математики. С учетом этой направленности ниже приведено более детальное описание отдельных модулей.

Специальные функции

Специальные функции рассматриваются как классы функций, которые обычно возникают в прикладных задачах при решении дифференциальных уравнений. Наибольшего внимания заслуживают ортогональные многочлены. сферические и цилиндрические функции, гипергеометрические функции, вырожденные гипергеометрические функции, гамма-функция, эллиптические функции, интегральный синус, косинус, показательная функция, логарифм, интеграл вероятности. В модуле special пакета SciPy представлены все основные специальные функции. Мы проиллюстрируем работу с некоторые из них, для более полного знакомства воспользуйтесь документацией к модулю special.

Одними из широко используемых специальных функций являются ортогональные многочлены (полиномы): система многочленов $\{p_n(x)\}$, $n=0,1,2,\ldots$ ортогональных с весом $\rho(x)$ на отрезке $I=[a,b]$. В пакете SciPy представлены основные из них (табл. 3.10).

С помощью программы

```
import scipy.special as spec
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-1., 1., 100)
```

Таблица 3.10 *Ортогональные полиномы*

Функция	Описание
legendre(n)	полиномы Лежандра $P_n(x)$ ($\rho(x) = 1$, $I = [-1, 1]$)
chebyt(n)	полиномы Чебышева первого рода $T_n(x)$ ($\rho(x) = (1 - x^2)^{-1/2}$, $I = [-1, 1]$)
chebyu(n)	полиномы Чебышева второго рода $U_n(x)$ ($\rho(x) = (1 - x^2)^{1/2}$, $I = [-1, 1]$)
chebyc(n)	полиномы Чебышева первого рода $C_n(x)$ ($\rho(x) = (1 - (x/2)^2)^{1/2}$, $I = [-2, 2]$)
chebys(n)	полиномы Чебышева второго рода $S_n(x)$ ($\rho(x) = (1 - (x/2)^2)^{1/2}$, $I = [-2, 2]$)
jacobi(n,α,β)	полиномы Якоби $P_n^{\alpha,\beta}(x)$ ($\rho(x) = (1 - x)^\alpha (1 + x)^\beta$, $I = [-1, 1]$)
laguerre(n)	полиномы Лагерра $L_n(x)$ ($\rho(x) = \exp(-x)$, $I = [0, \infty]$)
genlaguerre(n,α)	обобщенные полиномы Лагерра $L_n^\alpha(x)$ ($\rho(x) = \exp(-x)x^\alpha$, $I = [0, \infty]$)
hermite(n)	полиномы Эрмита $H_n(x)$ ($\rho(x) = \exp(-x^2)$, $I = [-\infty, \infty]$)
hermitenorm(n)	нормированные полиномы Эрмита $He_n(x)$ ($\rho(x) = \exp(-(x/2)^2)$, $I = [-\infty, \infty]$)
gegenbauer(n,α)	полиномы Гегенбауэра (ультрасферические) $C_n^\alpha(x)$ ($\rho(x) = (1 - x^2)^{\alpha - 1/2}$, $I = [-1, 1]$)
sh_legendre(n)	смещенные полиномы Лежандра $P_n^*(x)$ ($\rho(x) = 1$, $I = [0, 1]$)
sh_chebyt(n)	смещенные полиномы Чебышева первого рода $T_n^*(x)$ ($\rho(x) = (x - x^2)^{-1/2}$, $I = [0, 1]$)
sh_chebyu(n)	смещенные полиномы Чебышева второго рода $U_n^*(x)$ ($\rho(x) = (x - x^2)^{1/2}$, $I = [0, 1]$)
sh_jacobi(n,α,β)	полиномы Якоби $P_n^{\alpha,\beta}(x)$ ($\rho(x) = (1 - x)^{\alpha - \beta} x^{\beta - 1}$, $I = [0, 1]$)

```
y1 = spec.chebyt(1)(x)
plt.plot(x, y1, '-', label="$T_1(x)$")
y2 = spec.chebyt(2)(x)
plt.plot(x, y2, '--', label="$T_2(x)$")
y3 = spec.chebyt(3)(x)
plt.plot(x, y3, '-.', label="$T_3(x)$")
y4 = spec.chebyt(4)(x)
plt.plot(x, y4, ':', label="$T_4(x)$")
plt.xlabel('x')
plt.legend(loc=9)
plt.grid(True)
plt.show()
```

на рис. 3.26 показаны графики полиномов Чебышева.

Модуль special предоставляет большие возможности по работе с функциями Бесселя (цилиндрическими функциями). Имеется возможность не только вычислять значения цилиндрических функций, но и интегралы и производные от них, а также найти корни. Например, функция `jn(n,x)` (`yn(n,x)`) вычисляет значение функции Бесселя первого (второго) рода целого порядка `n` в точке `x`. В случае нецелого порядка `v` для комплексного аргумента `z`

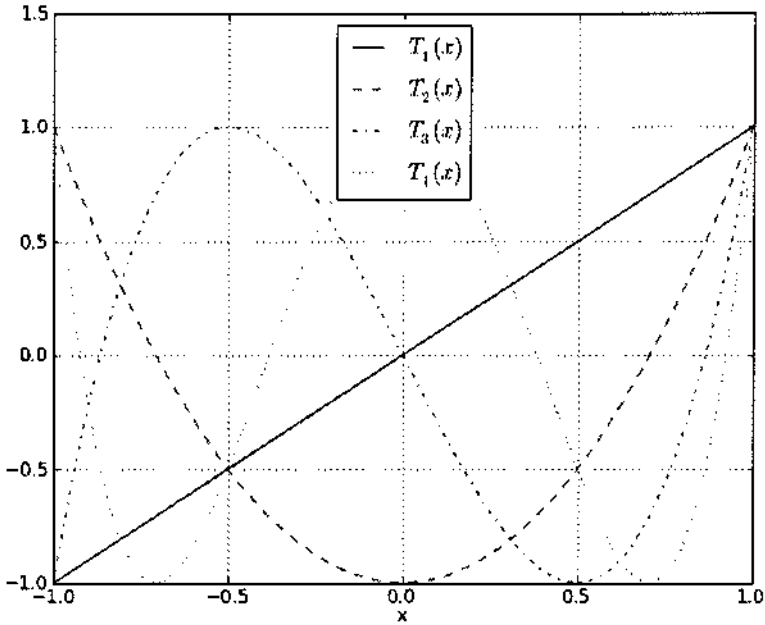

Рис. 3.26 Полиномы Чебышева первого рода

используются функции `jv(v,z)` (`yv(v,z)`) соответственно. Для вычисления корней используются функции `jn_zeros(n,nt)` (`yn_zeros(n,nt)`), где `nt` - число корней.

Ниже представлена программа для рисования функций Бесселя первого рода $J_n(x), n = 0, 1, 3$ и первых 6 корней функции $J_0(x)$ (см. рис. 3.27).

```
import scipy.special as spec
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0., 20., 100)
y1 = spec.jn(0,x)
plt.plot(x, y1, '-', label="$J_0(x)$")
y2 = spec.jn(1,x)
plt.plot(x, y2, '--', label="$J_1(x)$")
y3 = spec.jn(2,x)
plt.plot(x, y3, '-.', label="$J_2(x)$")
zeros = spec.jn_zeros(0, 6)
for xz in zeros:
    plt.scatter(xz, 0)
plt.xlabel('x')
```

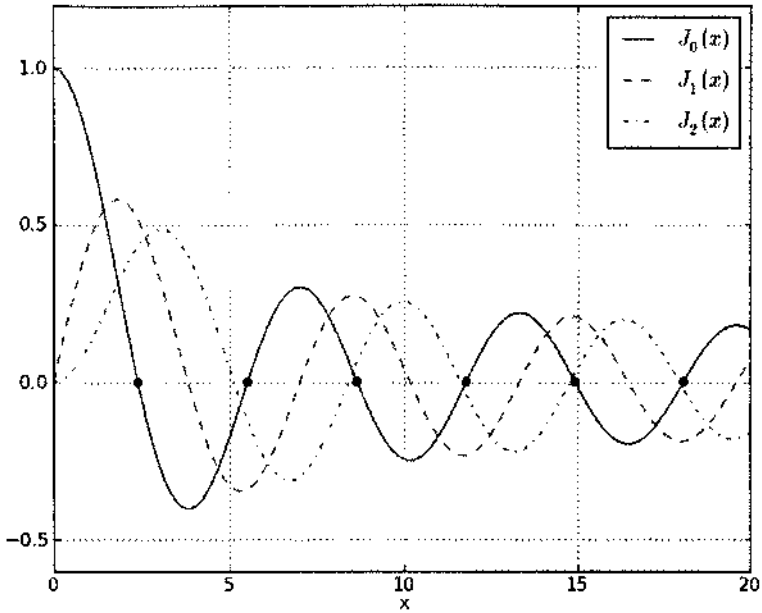

Рис. 3.27 Функции Бесселя первого рода

```
plt.legend()
plt.grid(True)
plt.show()
```

В качестве еще одного примера отметим гамма-функцию

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt,$$

которая определена как для вещественных, так и для комплексных z. График функции $\Gamma(x)$ для $-5 \leq x \leq 5$ приведен на рис. 3.28. Расчет выполнен по следующей программе.

```
import scipy.special as spec
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-5,5,0.01)
y = spec.gamma(x)
for i in x:
    if y[i] > 1.e20: y[i] = 1.e20
    if y[i] < -1.e20: y[i] = -1.e20
plt.plot(x, y, label="$\\Gamma(x)$")
```

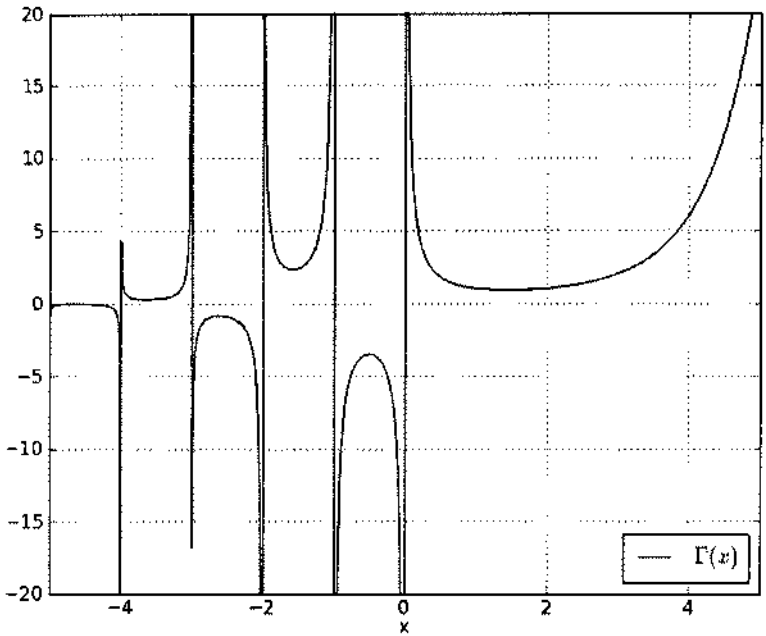

Рис. 3.28 Гамма-функции

```
plt.axis([-5.,5.,-20.,20.])
plt.xlabel('x')
plt.legend(loc=4)
plt.grid(True)
plt.show()
```

В данном примере для корректной прорисовки слишком большие по модулю значения функции обрезаются.

Быстрое преобразование Фурье

На основе преобразования Фурье решаются многие задачи обработки сигналов. Задачи прямого и обратного преобразования Фурье возникают также при применении метода разделения переменных для краевых задач математической физики. Отметим возможности модуля fftpack пакета SciPy по преобразованию Фурье сеточных периодических функций.

Будем считать, что комплексная периодическая функция $y_j, j = 0, \pm 1, ...$ с периодом N определяется по значениям в узлах $j = 0, 1, ..., N - 1$, так что $y_j = y_{j+mN}$, где m — любое целое. Функция $y_j, j = 0, \pm 1, ...$ представляется в

виде

$$y_j = \sum_{k=0}^{N-1} x_k e^{\frac{2k\pi j}{N} i}, \tag{3.1}$$

где $i = \sqrt{-1}$. Здесь $x_k, k = 0, 1, ..., N-1$ — коэффициенты Фурье, для которых имеет место представление

$$x_k = \frac{1}{N} \sum_{j=0}^{N-1} y_j e^{-\frac{2k\pi j}{N} i}. \tag{3.2}$$

Вычисление коэффициентов Фурье по (3.2) есть прямое преобразование Фурье функции $y_j = 0, 1, ..., N-1$, а восстановление самой функции по коэффициентам $x_k, k = 0, 1, ..., N-1$ согласно (3.1) — обратное преобразование Фурье.

В общем случае прямое и обратное преобразование Фурье требует $O(N^2)$ арифметических действий. Вычислительную работу удается снизить до величины $O(N \log N)$, например, при $N = 2^n$ — алгоритмы быстрого преобразования Фурье.

В модуле fftpack для прямого преобразования Фурье используется функция `fft()`, обязательным аргументом которой является массив значений сеточной функции. Обратное преобразование Фурье обеспечивается функцией `ifft()`.

Приведенная ниже программа иллюстрирует использование функций `fft()` и `ifft()` при Фурье-преобразовании разрывной функции. На рис. 3.29 приведены модули коэффициентов Фурье, а на рис. 3.30 представлены точные значения сеточной функции и полученные после применения прямого, а затем обратного преобразования Фурье.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import fftpack
def f (x):
    f = x
    if x > 0.5: f = 0.
    return f
N = 32
y = np.zeros((N), 'float')
t = np.zeros((N), 'float')
for k in range(N):
    tk = np.float(k) / N
    t[k] = tk
    y[k] = f(tk)
x = fftpack.fft(y)
z = np.sqrt(x.real**2 + x.imag**2)
plt.figure(1)
plt.bar(range(N), z, align='center', width=0.5, color='g')
```

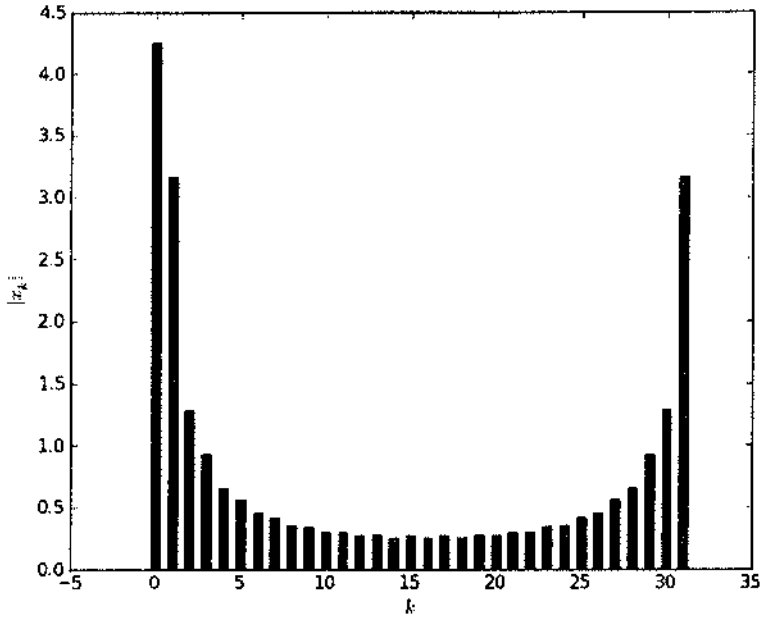

Рис. 3.29 Модули коэффициентов Фурье

```
plt.xlabel('$k$')
plt.ylabel('$|x_k|$')
plt.figure(2)
y1 = fftpack.ifft(x)
plt.plot(t, y, ':', label='exact')
plt.plot(t, y1, '--', label='direct-invers')
plt.xlabel('$t$')
plt.legend(loc=1)
plt.show()
```

Для двумерного преобразования Фурье используются функции `fft2()` и `ifft2()`, а для Фурье-преобразований многомерных массивов имеются функции `fftn()` и `ifftn()`.

В модуле `fftpack` предусмотрена также возможность работы с сеточными функциями, которые принимают только вещественные значения — функции `rfft()` и `irfft()` для прямого и обратного преобразований Фурье соответственно.

На основе дискретного преобразования Фурье реализованы дополнительные операции дифференцирования и интегрирования периодических сеточных функций (функция `diff()`), некоторые псевдодифференциальные операции

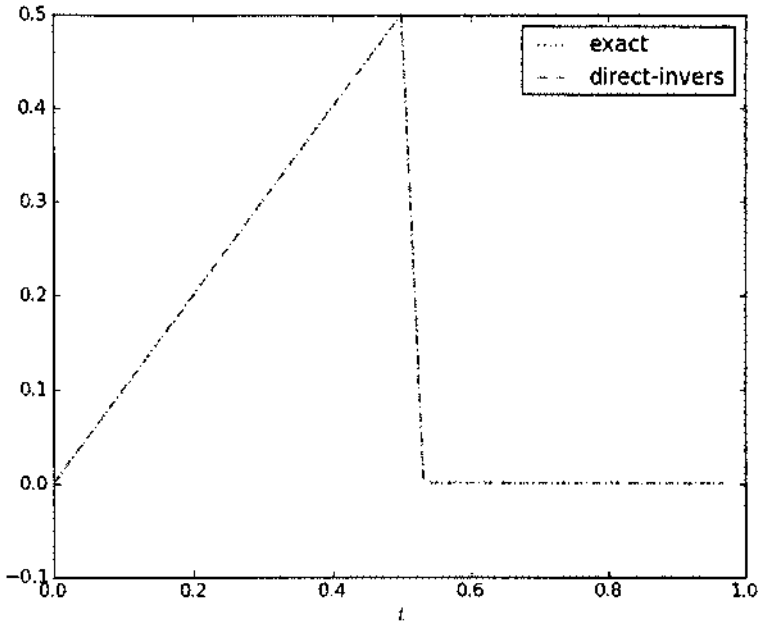

Рис. 3.30 Сеточные функции

Поддерживается также возможность быстрого вычисления свертки двух сеточных функций (функция `convolve()`). Имеется также возможность использования прямого и обратного преобразований Гильберта (функции `hilbert()` и `ihilbert()`), которые тесно связаны с преобразованиями Фурье.

Линейная алгебра

Пакет SciPy предоставляет богатые возможности по решению задач линейной алгебры. Как мы уже отмечали выше, эти задачи представлены в пакете NumPy, здесь же мы коротко опишем инструменты, предоставляемые модулем linalg. Этот модуль содержит функции для вычисления норм матриц и векторов, обращения и факторизации матриц, решения систем линейных уравнений, позволяет находить собственные значения и собственные вектора, вычислять функции матриц. Для решения задач линейной алгебры с разреженными матрицами, которые особенно важны при научных вычислениях, предназначен модуль sparse.linalg.

В SciPy, как и пакете NumPy, мы можем работать как с массивами, так и с матрицами. При задании матриц поддерживается синтаксис MATLAB (инициализация по строкам). Напомним базовые операции над матрицами.

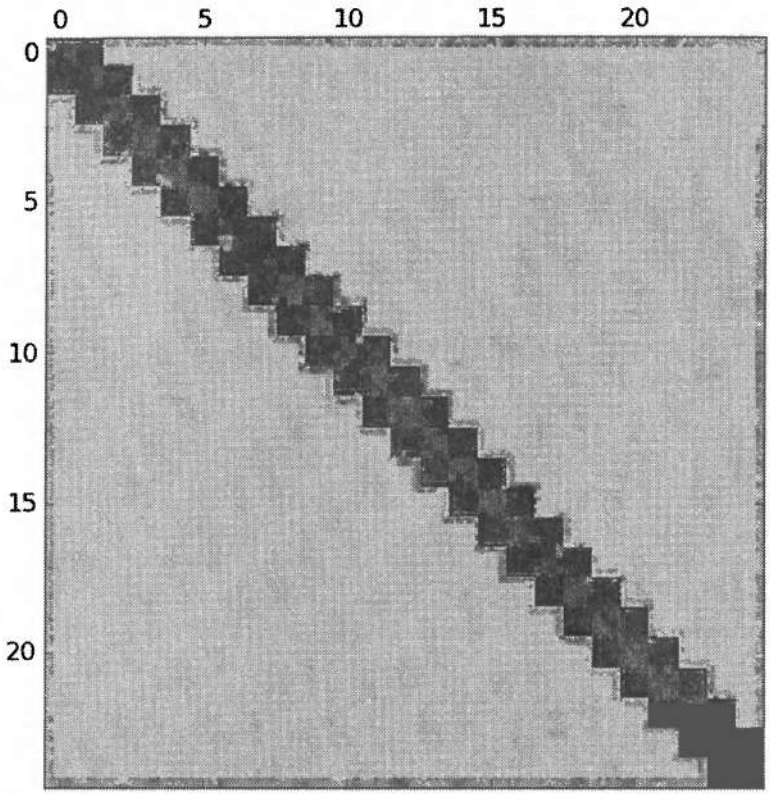

Рис. 3.31 Портрет матрицы A

Для вычисления определителя матрицы имеется функция `det()`. Для нахождения нормы матрицы или вектора предназначена функция `norm()`. Второй аргумент этой функции принимает значения 1, 2 и `inf` (2 по умолчанию) и соответствует выбору нормы вектора в l_p при $p = 1, 2, \infty$ соответственно и подчиненной нормы матрицы. Здесь `inf` означает бесконечность (определено в пакете NumPy).

Для примера рассмотрим матрицу, которая соответствует разностной производной второго порядка с обратным знаком на интервале $0 \le x \le 1$ для функций, обращающихся в ноль на концах. Используется равномерная сетка с шагом h — внутренние узлы $x_i, i = 0, 1, ..., m-1$ и $h = 1/(m+1)$. Для модельной матрицы A размерности $m \times m$ нарисуем портрет, сосчитаем определитель и различные нормы матрицы, проведем разложение Холецкого, найдем обратную матрицу и получим решение уравнения $Ay = b$ при $b = 1$.

```
import numpy as np
```

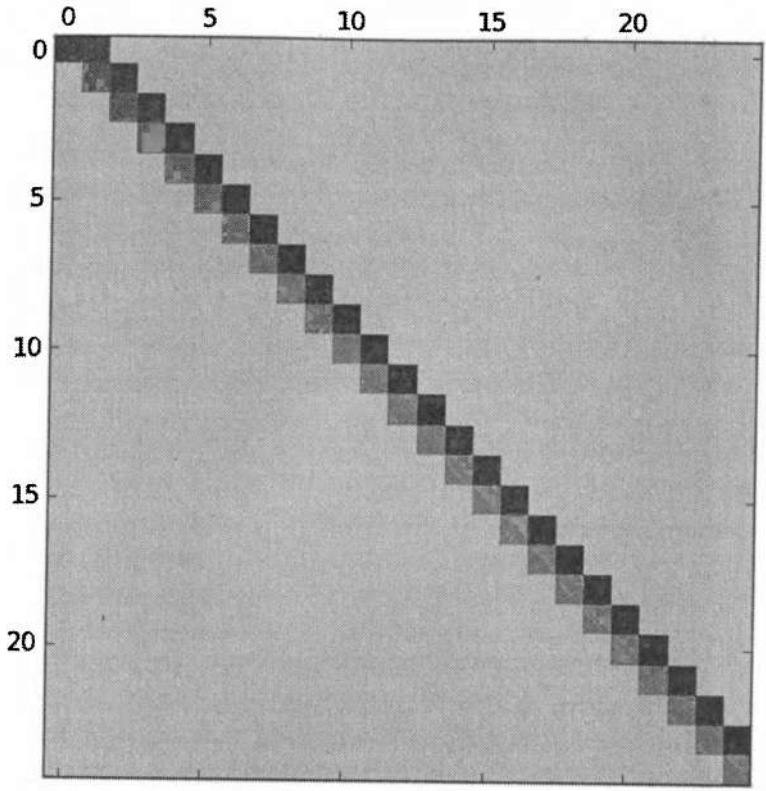

Рис. 3.32 Матрица U ($A = UU^*$)

```
from scipy import linalg
import matplotlib.pyplot as plt
m = 25
h = 1. / (m+1)
A = np.mat(np.zeros((m, m), 'float'))
for i in range(1, m):
    A[i-1,i] = - 1. / h**2
    A[i,i] = 2. / h**2
    A[i,i-1] = - 1. / h**2
A[0,0] = 2. / h**2
A[m-1,m-1] = 2. / h**2
plt.matshow(A)
print 'det:', linalg.det(A)
print 'norm-1:', linalg.norm(A, 1)
```

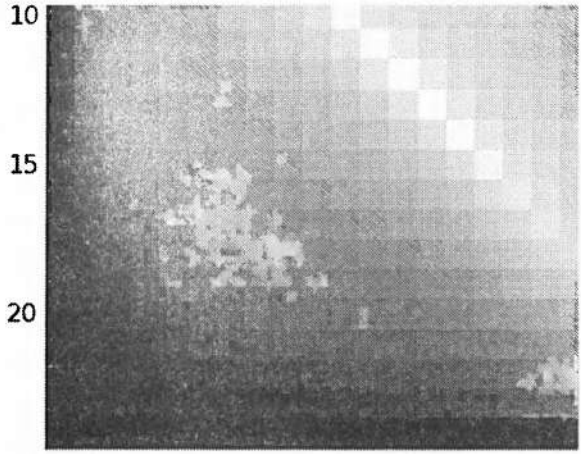

Рис. 3.33 Портрет матрицы $B = A^{-1}$

```
print 'norm-2:', linalg.norm(A, 2)
print 'norm-inf:', linalg.norm(A, np.inf)
L = linalg.cholesky(A)
plt.matshow(L)
B = linalg.inv(A)
plt.matshow(B, cmap='gray')
b = np.mat(np.ones((m), 'float'))
y = B*b.T
plt.figure(4)
x = np.linspace(h, m*h, m)
plt.plot(x, y, '--', x, x*(1-x)/2, ':')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```

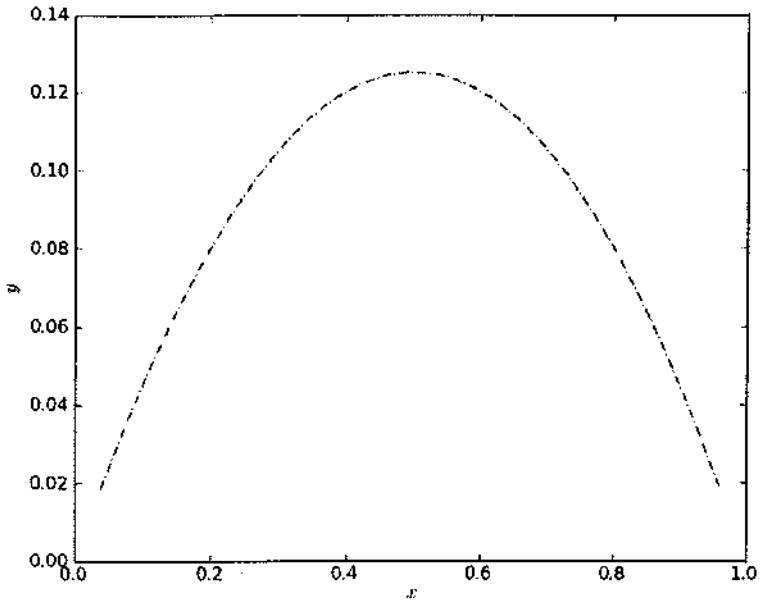

Рис. 3.34 Решение уравнения

```
det  1.45760801099e+72
norm-1  2704.0
norm-2  2694.14239778
norm-inf  2704.0
```

На рис. 3.31 приведен портрет матрицы A (использовалась функция `matshow()` из пакета Matplotlib). Матрица A является симметричной и трехдиагональной.

В нашем примере решение системы уравнений проведено с явным использованием обратной матрицы. Вторая возможность связана с применением функции `solve()` (вместо `y = B*b.T` положим `y = linalg.solve(A, b.T)`). Функция `solve_banded()` предназначена для решения систем уравнений с ленточной матрицей.

С учетом того, что матрица A симметрична и положительно определена мы представили ее в факторизованном виде $A = LL^* = U^*U$ (ленточность множителей матрицы наблюдается на рис. 3.32). Обратная матрица (рис. 3.33), конечно, является полной. Сравнение с точным решением показано на рис. 3.34 — разностный оператор второй производной является точным для полиномов второй степени.

Для более общих матриц в модуле linalg имеются функции LU-разложения. Более того, для работы с любыми невырожденными матрицами реализовано более общее разложение $A = PLU$ — функция `lu()`. Здесь L — нижняя треугольная матрица с диагональными элементами, равными единице, U — верхняя треугольная матрица, а P — матрица перестановок. Другие возможности LU-разложения реализованы в функциях `lu_factor()`, `lu_solve()` (см. документацию по модулю linalg).

Имеются также функции для других разложений матриц:

- разложение Холецкого — функции `cholesky()`, `cholesky_banded()`, `cho_factor()`, `cho_solve()`;
- QR-разложение — функция `qr()`;
- сингулярное (SVD) разложение — функции `svd()`, `svdvals()`, `diagsvd()`, `diagsvd()`;
- разложение Шура — функции `schur()`, `rsf2csf()` .

Использование метода наименьших квадратов проиллюстрируем на задаче подбора коэффициентов сглаживающего полинома. Пусть в точках $x_i = ih$, $i = 0, 1, ..., m$ известны значения функции y_i, $i = 0, 1, ..., m$. Эти данные будем приближать с помощью полинома $z(a; x) = a_0 + a_1 x + ... + a_n x^n$, причем $n < m$. Коэффициенты $a_k, k = 0, 1, ..., n$ находятся из условия минимума

$$J(a) = \sum_{i=1}^{m} (z(a; x_i) - y_i)^2.$$

В матричной записи имеем $J(a) = \|Aa - b\|^2$.

Реализация метода наименьших квадратов с использованием функции `lstsq()` дается следующей программой.

```
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
m = 20
n = 3
h = 1./m
A = np.mat(np.zeros((m+1, n+1), 'float'))
x = np.zeros((m+1), 'float')
y = np.zeros((m+1), 'float')
y1 = np.zeros((m+1), 'float')
for i in range(0, m+1):
    xx = i*h
    x[i] = xx
    y[i] = np.exp(-xx**2)
    for k in range(0, n):
        A[i,k] = xx**k
a, resid, rank, sigma = linalg.lstsq(A, y)
for i in range(0, m+1):
```

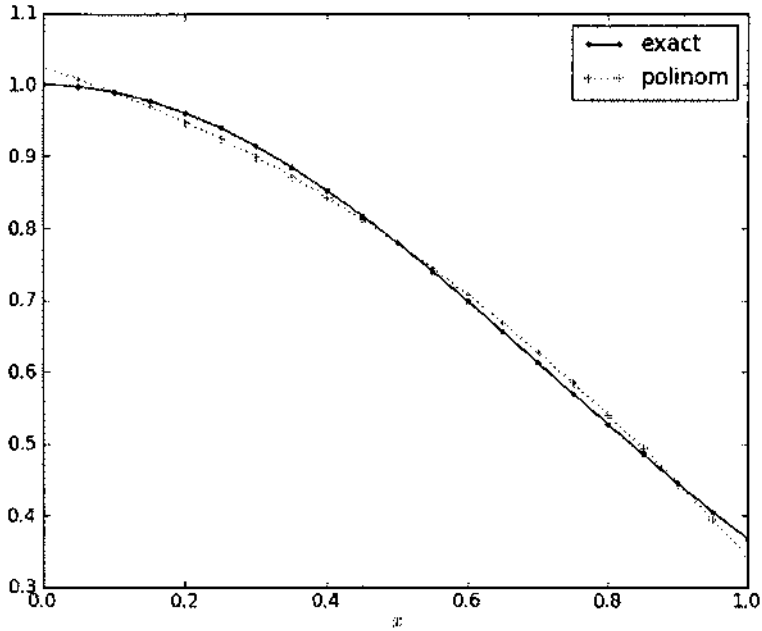

Рис. 3.35 Аппроксимация полиномом

```
    xx = i*h
    sum = 0.
    for k in range(0, n):
        sum += a[k]*xx**k
    y1[i] = sum
plt.plot(x, y, '.-', label='exact')
plt.plot(x, y1, '+:', label='polinom')
plt.xlabel('$x$')
plt.legend()
plt.show()
```

Рис. 3.34 демонстрирует точность аппроксимации функции e^{-x^2} полиномом третьей степени при $0 \leq x \leq 1$. Вместо функции `lstsq()` можно использовать `pinv()` или `pinv2()`, которые позволяют найти псевдорешение системы линейных уравнений.

Приведем также пример нахождения собственных значений матриц. Будем искать собственные значения оператора центральной производной для пери-

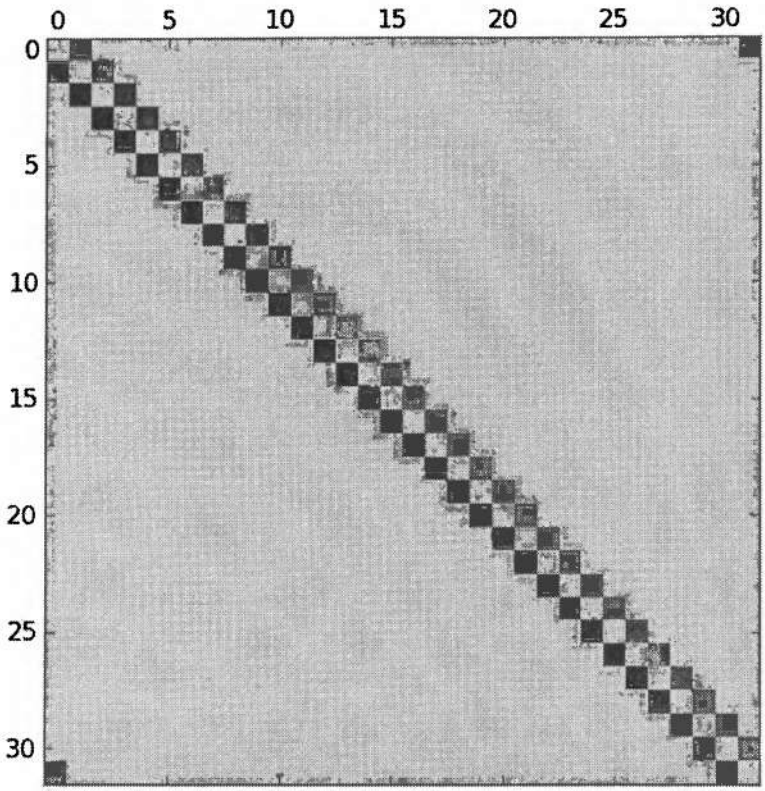

Рис. 3.36 Портрет матрицы A

одических сеточных функций:

$$Ay = \frac{y_{i+1} - y_{i-1}}{2h}, \quad i = 0, 1, ..., m-1,$$

при $y_{i+m} = y_i$, $h = 1/m$. Собственными значения кососимметричной матрицы A являются чисто мнимые:

$$\lambda_k = \frac{\sqrt{-1}}{h} \sin(2\pi k h), \quad k = 0, 1, ..., m-1.$$

Ниже представлена программа вычисления собственных значений этой матрицы (на рис. 3.36 показан портрет матрицы) с использованием функции `eigvals()` модуля linalg. На рис. 3.37 дано сравнение точных и вычисленных собственных значений.

```
import numpy as np
```

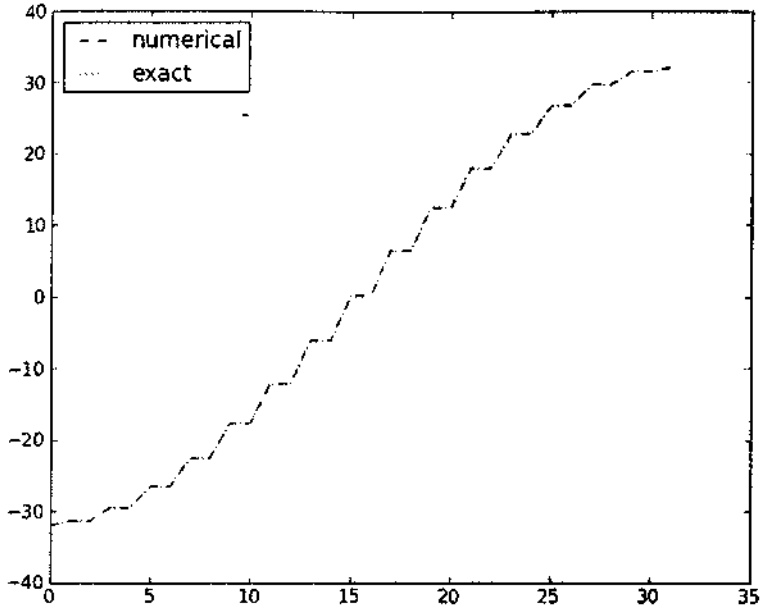

Рис. 3.37 Точные и вычисленные собственные значения

```
from scipy import linalg
import matplotlib.pyplot as plt
m = 32
h = 1. / m
A = np.mat(np.zeros((m, m), 'float'))
for i in range(1, m):
    A[i-1,i] = 1. / (2*h)
    A[i,i-1] = -1. / (2*h)
A[0,m-1] = -1. / (2*h)
A[m-1,0] = 1. / (2*h)
plt.matshow(A)
plt.figure(2)
la = linalg.eigvals(A)
laI = np.sort(la.imag)
ii = np.linspace(0, m-1, m)
plt.plot(ii, laI, '--', label='numerical')
laE = np.sort(1./h * np.sin(2*np.pi*ii*h))
plt.plot(ii, laE, ':', label='exact')
plt.legend(loc=0)
plt.show()
```

Для нахождения как собственных значений так и собственных функций используется функция `eig()`. Реализованы также алгоритмы решения спектральных задач для специальных классов матриц: функции `eigvalsh()` и `eigh()` для комплексных эрмитовых и вещественных симметричных матриц, функции `eigvals_banded()` и `eig_banded()` для ленточных матриц.

В модуле linalg имеется поддержка работы с функциями матриц. Для вычисления матричной экспоненты

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k$$

используются три функции: `expm()` — аппроксимация Паде, `expm2()` — спектральное разложение, `expm3()` — ряд Тейлора. Для вычисления логарифма имеется функция `logm()`. Реализованы также матричный синус, косинус и тангенс (функции `sinm()`, `cosm()` и `tanm()`), гиперболический синус, косинус и тангенс (функции `sinhm()`, `coshm()` и `tanhm()`). Для вычисления матричная sgn (sign) функции используется `signm()`, квадратного корня — `sqrtm()`. Отметим также возможность вычисления функций матриц (с помощью `funm()`), которые связываются с любой функцией Python.

Рассмотрим пример использования матричной экспоненты при решении задачи Коши:

$$\frac{dy}{dt} + Ay = 0, \quad 0 < t \le T,$$

$$y(0) = u.$$

При постоянной (не зависящей от t) матрице A решение представляется в виде

$$y(t) = e^{-At}u.$$

В программе матрица A соответствует второй разностной производной сеточных функций, обращающихся в ноль на концах интервала $[0, 1]$. Точное решение дискретной и непрерывной задачи представляется в виде разложения по собственным функциям $\varphi_k = \sqrt{2}\sin(\pi k x), k = 1, 2, \ldots$.

```
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
n = 3
m = 19
t = 0.1
h = 1. / (m+1)
A = np.mat(np.zeros((m, m), 'float'))
for i in range(1, m):
    A[i-1,i] = - 1. / h**2
    A[i,i] = 2. / h**2
```

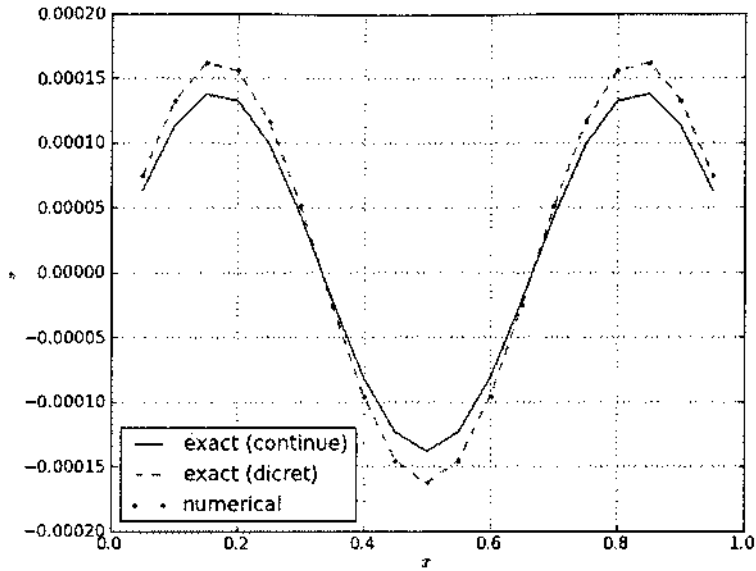

Рис. 3.38 Решение задачи Коши при $t = 0.1$

```
    A[i,i-1] = - 1. / h**2
A[0,0] = 2. / h**2
A[m-1,m-1] = 2. / h**2
x = np.zeros((m), 'float')
y = np.zeros((m), 'float')
for i in range(0, m):
    xx = (i+1)*h
    x[i] = xx
    y[i] = np.sin(np.pi*n*xx)
y0 = np.exp(-(np.pi*n)**2 * t) * y
y1 = np.exp(-4/h**2*(np.sin(np.pi*n*h/2))**2 * t) * y
B = np.mat(linalg.expm(-A*t))
y2 = B*np.mat(y).T
plt.plot(x, y0, '-', label='exact (continue)')
plt.plot(x, y1, '--', label='exact (dicret)')
plt.plot(x, y2, '.', label='numerical')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.legend(loc=0)
plt.grid(True)
plt.show()
```

Здесь

$$\lambda_k = \pi^2 k^2, \quad \tilde{\lambda}_k = \frac{4}{h^2} \sin^2\left(\frac{\pi k h}{2}\right)$$

есть собственные значения непрерывной и сеточной задачи соответственно. Использование функции expm2() вместо expm() даст те же результаты, а вот expm3() — ничего разумного (из-за плохой сходимости ряда Тейлора при вычислении матричной экспоненты).

Для полной поддержки вычислительных алгоритмов линейной алгебры можно добавить итерационные методы решения систем линейных уравнений. Соответствующие функции можно найти в модуле решения задач с разреженными матрицами sparse.linalg (ранее они находились в linalg). Пример использования метода сопряженных градиентов для ранее рассмотренной задачи (см. рис. 3.34) дается следующей программой.

```
import numpy as np
from scipy.sparse import linalg
import matplotlib.pyplot as plt
m = 25
h = 1. / (m+1)
A = np.mat(np.zeros((m, m), 'float'))
for i in range(1, m):
    A[i-1,i] = - 1. / h**2
    A[i,i] = 2. / h**2
    A[i,i-1] = - 1. / h**2
A[0,0] = 2. / h**2
A[m-1,m-1] = 2. / h**2
b = np.mat(np.ones((m), 'float')).T
help(linalg.cg)
y, info = linalg.cg(A, b, tol=1.e-06)
x = np.linspace(h, m*h, m)
plt.plot(x, y, '--', x, x*(1-x)/2, ':')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```

Модуль sparse.linalg помимо метода сопряженных градиентов (функция cg()) включает другие итерационные методы решения систем линейных уравнений:

- cgs() — квадратичный метод сопряженных градиентов;
- qmr() — метод квазиминимальных невязок;
- gmres() — обобщенный метод минимальных невязок;
- bicg() — метод бисопряженного градиента;
- bicgstab() — метод бисопряженного градиента со стабилизацией.

Эти итерационные методы применяются при решении задач как с полными, так и с разреженными матрицами.

Интерполяция

Модуль interpolate пакета SciPy предоставляет инструменты для решения стандартных задач интерполяции и сглаживания полиномами и сплайнами сеточных функций одной переменной. Имеются также функции для интерполяции двумерных данных, заданных на регулярной сетке или в произвольных точках плоскости.

Пусть в узлах $x_0, x_1, ..., x_m$ функция $y = y(x)$ принимает заданные значения $y_0, y_1, ..., y_m$. При полиномиальной интерполяции строиться полином $p(x) = a_0x^m + a_1x^{m-1} + ... + a_m$, для которого $p(x_k) = y_k, k = 0, 1, ..., m$. Для построения интерполирующего полинома можно воспользоваться функцией `lagrange()`. Более устойчивые алгоритмы вычисления значений интерполирующего полинома реализованы в функциях `barycentric_interpolate()` и `krogh_interpolate()`. Приведем пример (рис. 3.39) построения интерполирующего полинома для функции Рунге $y = (1 + 25x^2)^{-1}$.

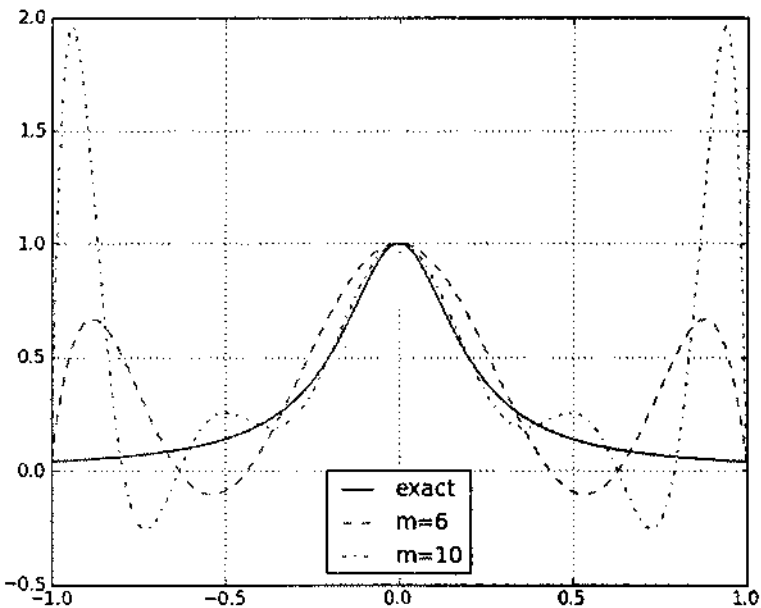

Рис. 3.39 Аппроксимация полиномами

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
x = np.linspace(-1, 1., 200)
```

```
plt.plot(x, 1./(1 + 25*x*x), '-', label='exact')
x1 = np.linspace(-1, 1., 7)
y1 = 1./(1 + 25*x1*x1)
p1 = interpolate.lagrange(x1, y1)
plt.plot(x, p1(x), '--', label='m=6')
x2 = np.linspace(-1, 1., 11)
y2 = 1./(1 + 25*x2*x2)
yb = interpolate.barycentric_interpolate(x2, y2, x)
plt.plot(x, yb, '-.', label='m=10')
plt.grid(True)
plt.legend(loc=0)
plt.show()
```

В практике научных вычислений гораздо большее внимание уделяется кусочно-полиномиальным аппроксимациям (сплайнам). Основной функцией одномерной интерполяции является `interp1d()`, первый аргумент которой есть массив узлов, второй — массив значений. Параметр `kind` задает тип кусочно-полиномиальной интерполяции. При `kind = 'linear'` (значение по умолчанию) используется кусочно-линейная интерполяция (линейные сплайны), `kind = 'cubic'` — кубические сплайны, значения `kind = 'nearest', 'zero'` связаны с кусочно-постоянными аппроксимациями. Целое значение `kind` соответствует степени интерполирующего полинома между узлами. При задании в узлах интерполяции производных кусочно-полиномиальная аппроксимация строится с помощью функции `piecewise_polynomial_interpolate()`.

Приведем пример использования функции `interp1d()` при интерполяции функции Рунге (см. рис. 3.40).

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
x = np.linspace(-1, 1., 200)
plt.plot(x, 1./(1 + 25*x*x), ':', label='exact')
x1 = np.linspace(-1, 1., 11)
y1 = 1./(1 + 25*x1*x1)
plt.scatter(x1, y1)
f1 = interpolate.interp1d(x1, y1, kind='nearest')
plt.plot(x, f1(x), '-', label='nearest')
f2 = interpolate.interp1d(x1, y1, kind=3)
plt.plot(x, f2(x), '--', label='cubic')
plt.grid(True)
plt.legend(loc=0)
plt.show()
```

Модуль `interpolate` содержит много инструментов для работы со сплайнами. В частности, можно не только вычислить значение сплайна в заданной точке, но и найти коэффициенты сплайна, вычислить производную сплайна, най-

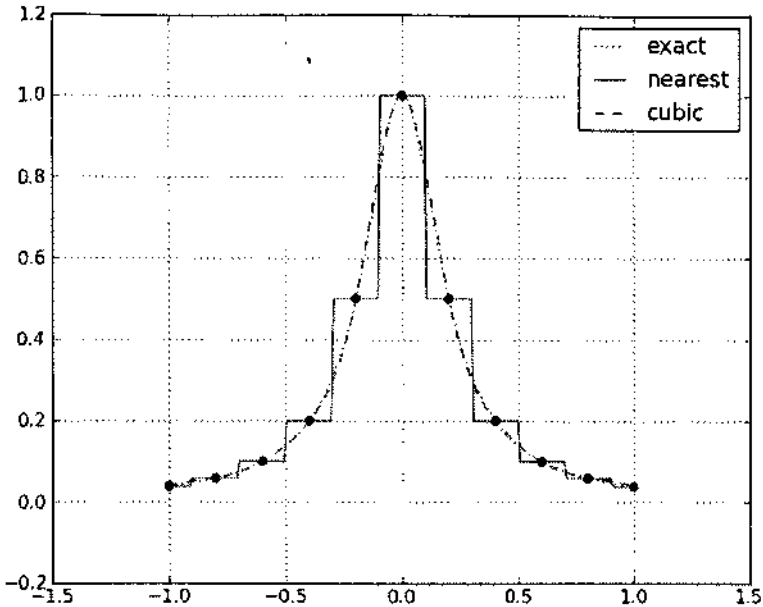

Рис. 3.40 Кусочно-полиномиальная аппроксимация

ти интеграл на заданном интервале, корни сплайна. В качестве первого иллюстративного примера рассмотрим построение сглаживающего B-сплайна (рис. 3.41). B-сплайн строиться с помощью функции `splrep()`, для вычисления значений сплайна используется `splev()`. Параметр `s` есть параметр сглаживания.

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
x = np.linspace(0., 1., 200)
plt.plot(x, np.cos(3*np.pi*x)*np.exp(-x), ':', label='exact')
x1 = np.linspace(0., 1., 11)
y1 = np.cos(3*np.pi*x1)*np.exp(-x1)
plt.scatter(x1, y1)
tck1 = interpolate.splrep(x1, y1, s=0)
s1 = interpolate.splev(x, tck1, der=0)
plt.plot(x, s1, '--', label='s=0')
tck2 = interpolate.splrep(x1, y1, s=0.05)
s2 = interpolate.splev(x, tck2, der=0)
plt.plot(x, s2, '-.', label='s=0.05')
```

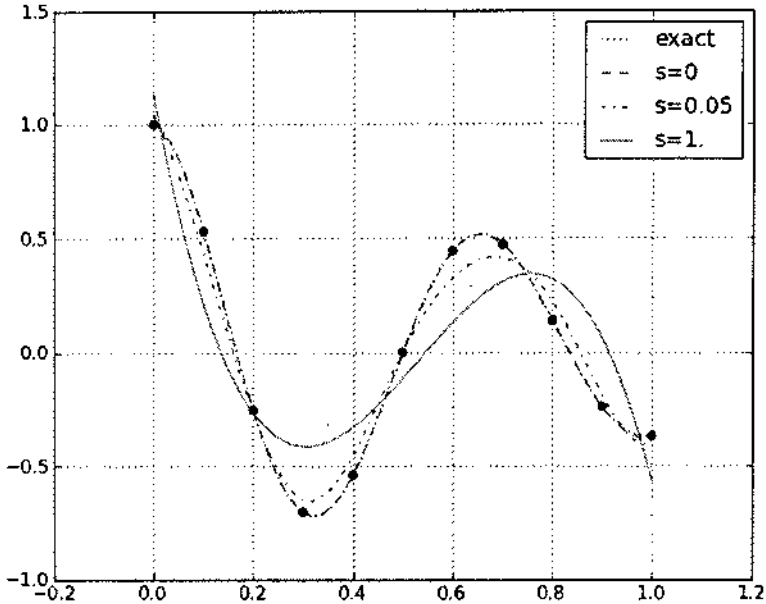

Рис. 3.41 Сглаживание сплайнами

```
tck3 = interpolate.splrep(x1, y1, s=1.)
s3 = interpolate.splev(x, tck3, der=0)
plt.plot(x, s3, '-', label='s=1.')
plt.grid(True)
plt.legend(loc=0)
plt.show()
```

Для интерполяции двумерных данных на двумерной сетке можно воспользоваться функциями `bisplrep()` (нахождение B-сплайна) и `bisplev()` (вычисление значений двумерного B-сплайна). На рис. 3.42 показана исходная сеточная функция на сетке 11×11, результат интерполяции (сетка $101 \times$ — на рис. 3.43.

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
x, y = np.mgrid[0:1:11j, 0:1:11j]
z = y*(1-y)*x
plt.figure(1)
plt.pcolor(x, y, z, cmap='gray')
plt.colorbar()
```

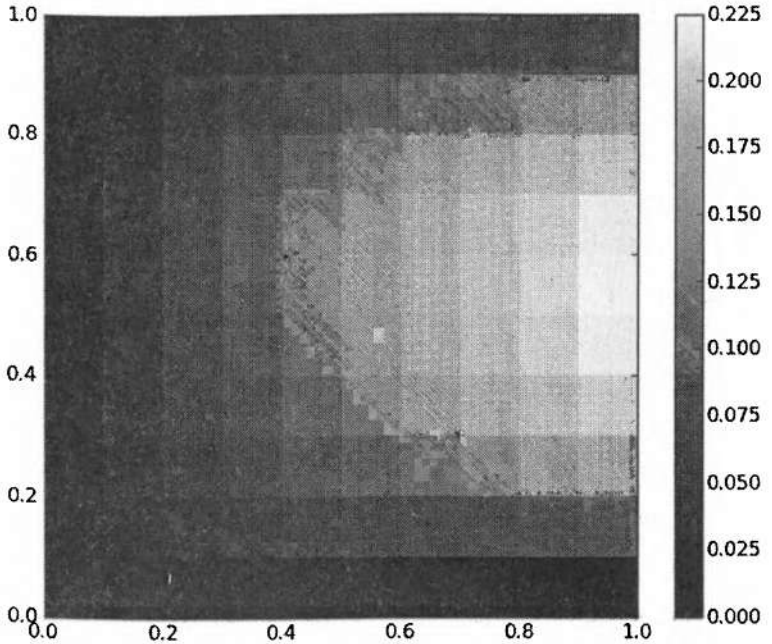

Рис. 3.42 Сеточная функция

```
x1, y1 = np.mgrid[0:1:101j, 0:1:101j]
tck = interpolate.bisplrep(x, y, z, s=0)
z1 = interpolate.bisplev(x1[:,0], y1[0,:], tck)
plt.figure(2)
plt.pcolor(x1, y1, z1, cmap='gray')
plt.colorbar()
plt.show()
```

Для работы с данными на нерегулярной сетке можно применять интерполяцию или сглаживание на основе радиальных базисных функций. Пример использования функции `Rbf()` представлен ниже (см. рис. 3.44).

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import Rbf
x = np.random.rand(100)
y = np.random.rand(100)
z = np.sin(2*np.pi*x)*y*(1-y)
rbf = Rbf(x, y, z, epsilon=2)
x1 = np.linspace(0., 1., 101)
```

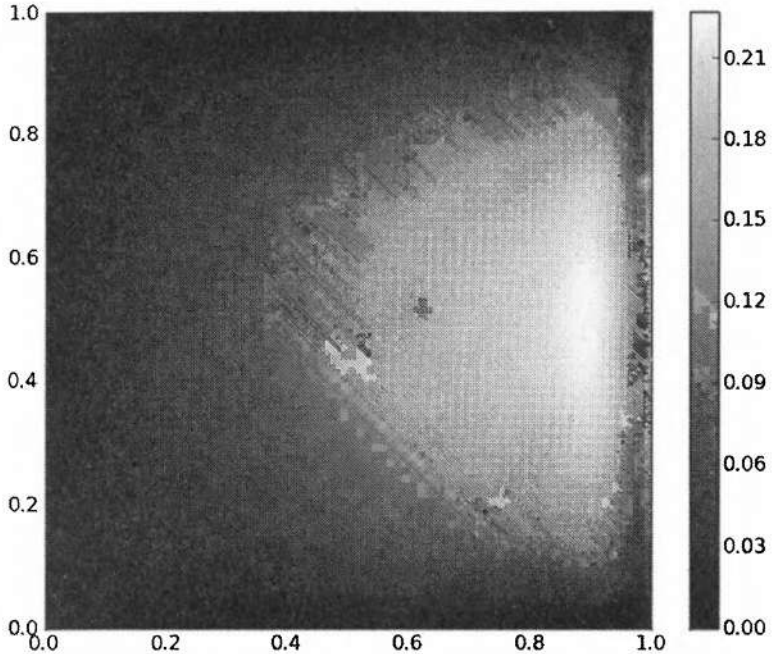

Рис. 3.43 Сплайн-интерполяция

```
y1 = np.linspace(0., 1., 101)
X1, Y1 = np.meshgrid(x1, y1)
Z1 = rbf(X1, Y1)
plt.pcolor(X1, Y1, Z1, cmap='gray')
plt.colorbar()
plt.scatter(x, y)
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.show()
```

Другой вариант интерполяции данных на произвольной сетке(кусочно-линейное восполнение при триангуляции Делоне) рассмотрен нами ранее (рис. 3.20) при обсуждении возможностей графического пакета Matplotlib.

Задачи оптимизации

В модуле optimize пакета SciPy представлены средства для решения нелинейных уравнений и их систем, минимизации одномерных функций и функций многих переменных без ограничений и с ограничениями.

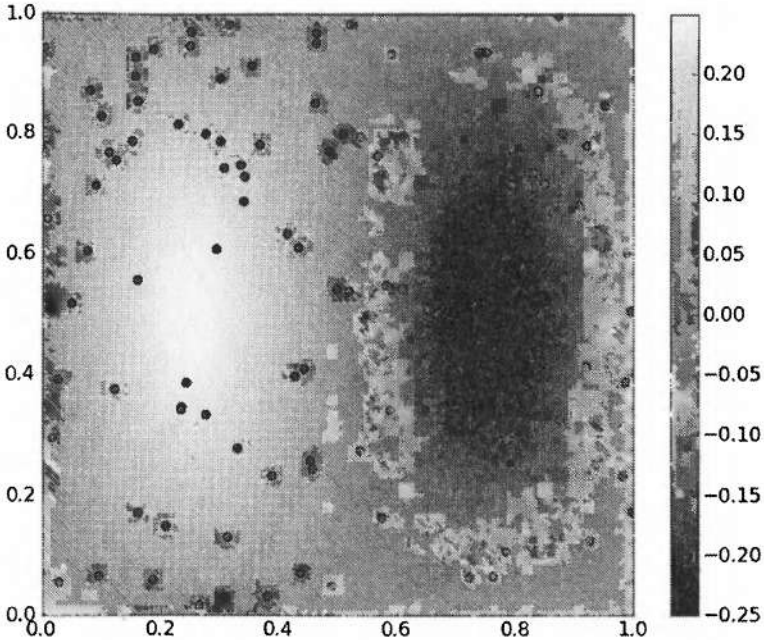

Рис. 3.44 Интерполяция на нерегулярной сетке

Для приближенного решения задачи нахождения решения нелинейного уравнения $f(x) = 0$ можно использовать различные методы. Для поиска корня на интервале $[a, b]$, на котором функция $f(x)$ меняет знак ($f(a)f(b) < 0$), можно применить метод Брента (функция `brentq()`). Ниже (см. рис. 3.45) ищется решение уравнения $2x - 1 + 2\cos(\pi x) = 0$ на интервалах $[0, 1]$ и $[1, 2]$.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as optimize
def f(x):
    return 2*x - 1 + 2*np.cos(np.pi*x)
x = np.linspace(0., 2., 201)
y = f(x)
plt.plot(x, y, label='$2x-1+2\\cos(\\pi x) = 0$')
x0 = optimize.brentq(f, 0., 1.)
x1 = optimize.brentq(f, 1., 2.)
print 'x:', x0, ',', x1
plt.scatter(x0, 0)
plt.scatter(x1, 0)
plt.legend(loc=0)
```

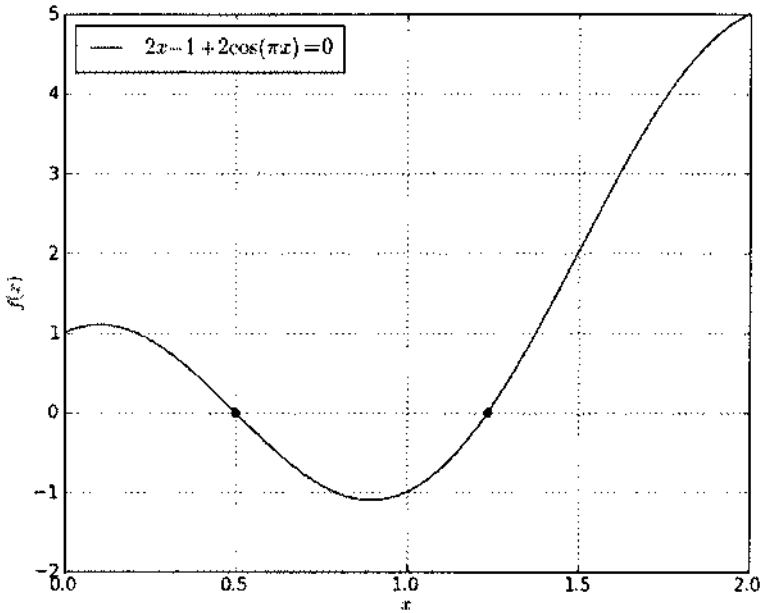

Рис. 3.45 Решение нелинейного уравнения

```
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
plt.grid(True)
plt.show()
```

```
x. 0.5 , 1.23648444824
```

Для решения систем нелинейных уравнений используется функция `fsolve` в которой нужно задать уравнения системы и начальное приближение к шению. Рассмотренное выше уравнение $2x-1+2\cos(\pi x)=0$ будем рассм ривать как систему

$$1-2x_0-x_1=0,$$
$$x_1-2\cos(\pi x_0)=0.$$

В следующей программе решение (см. рис. 3.46) получено при использова функции `fsolve()`.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as optimize
def F(x):
    out = [1 - 2*x[0] - x[1]]
```

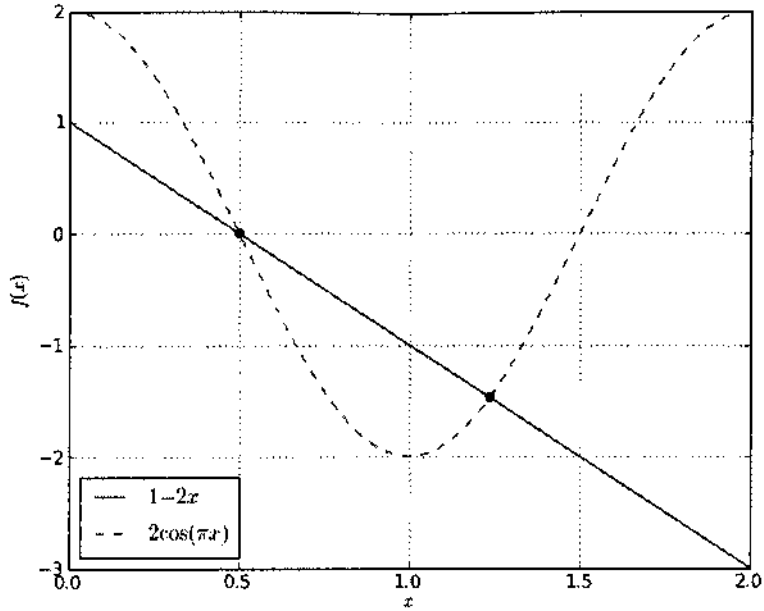

Рис. 3.46 Решение системы уравнений

```
    out.append(x[1] - 2*np.cos(np.pi*x[0]))
    return out
x = np.linspace(0., 2., 201)
y1 = 1 - 2*x
y2 = 2*np.cos(np.pi*x)
plt.plot(x, y1, '-', label='$1 - 2x$')
plt.plot(x, y2, '--', label='$2\\cos(\\pi x)$')
x0 = optimize.fsolve(F, [0.5, 0.])
x1 = optimize.fsolve(F, [1.5, 0.])
print 'x:', x0, ',', x1
plt.scatter(x0[0], x0[1])
plt.scatter(x1[0], x1[1])
plt.legend(loc=0)
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
plt.grid(True)
plt.show()
```

Для минимизации одномерных функций используются различные алгоритмы: например, в функции `golden()` — метод золотого сечения, в `brent()` метод Брента (метод обратной параболической интерполяции). В нижеприведенной программе используется функция `fminbound()` для минимизации функции $f(x) = x^2(x^2 - x - 6)$ на интервалах $[-3, 0]$ и $[0, 3]$ (рис. 3.47).

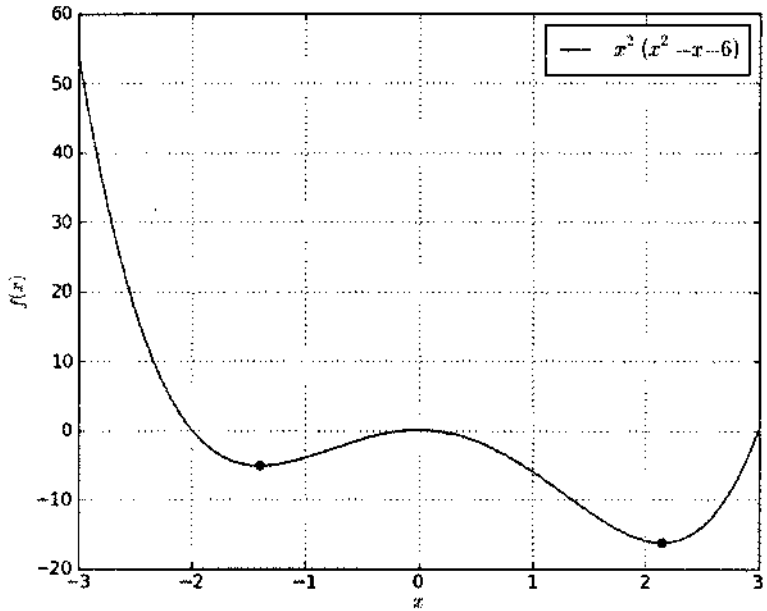

Рис. 3.47 Минимизация одномерной функции

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as optimize
def f(x):
    return x*x*(x*x - x - 6)
x = np.linspace(-3., 3., 201)
y = f(x)
plt.plot(x, y, label='$x^2(x^2 - x -6)$')
x0 = optimize.fminbound(f, -3., 0.)
x1 = optimize.fminbound(f, 0., 3.)
print 'x:', x0, ',', x1
plt.scatter(x0, f(x0))
plt.scatter(x1, f(x1))
plt.legend(loc=0)
```

```
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
plt.grid(True)
plt.show()
```

```
x: [-1.39718014] , [ 2.14718187]
```

При минимизации функций многих переменных обычно используются квазиньютоновские методы. В этом случае мы можем задавать производные минимизирующей функции точно или же они будут вычисляться приближенно в процессе вычислений. Мы ограничимся примером использования функции `fmin()`, в которой реализован симплекс-метод. Будем искать минимум функции $F(x) = 100(x_1 - x_0^2)^2 + (1 - x_0)^2$ (двумерная функция Розенброка).

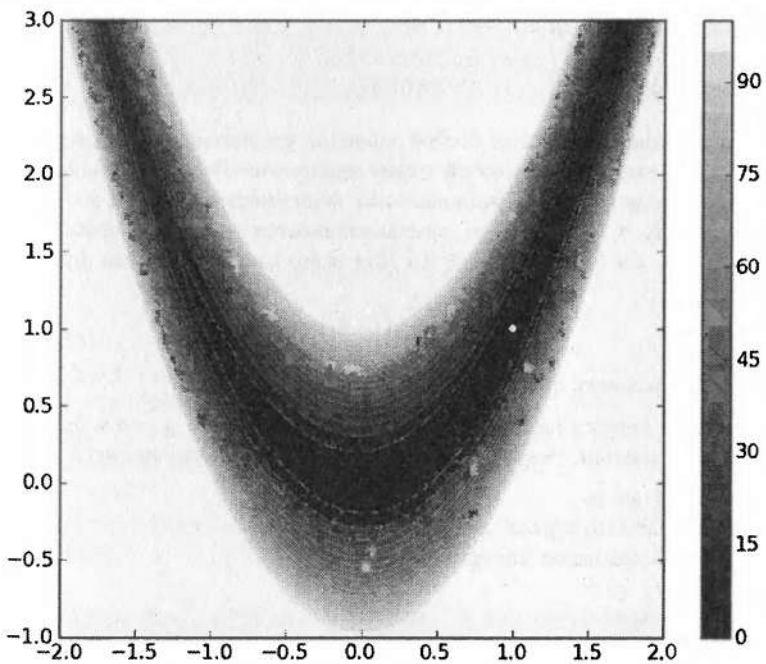

Рис. 3.48 Минимизация функции Розенброка

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as optimize
def f(x):
    return 100*(x[1] -x[0]**2)**2 + (1 - x[0])**2
x = np.linspace(-2., 2., 101)
```

```
y = np.linspace(-1., 3., 101)
X, Y = np.meshgrid(x, y)
z = f([X, Y])
v = np.linspace(0, 100, 21)
plt.contourf(x, y, z, v, cmap=plt.cm.gray)
plt.colorbar()
x0 = [0, 0]
xmin = optimize.fmin(f, x0)
plt.scatter(xmin[0], xmin[1], c='w')
print 'xmin:', xmin
plt.show()
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 79
         Function evaluations: 146
xmin = [1.00000439  1.00001064]
```

В вычислительной практике особое значение уделяется минимизации функции, которая представляет собой сумму квадратов. Типичной является ситуация, когда нелинейная функциональная зависимость функции $y = f(p;x)$ от параметров $p_k = k = 0,1,...,n$ восстанавливается на набору приближенных данных $x_i, \tilde{y}_i$, $i = 0,1,...,m$ $(m \geq n)$. Для этого минимизируется функционал

$$J(p) = \sum_{i=0}^{m} (f(p;x_i) - \tilde{y}_i)^2.$$

Можно использовать функцию `leastsq()`.

В программе ищутся параметры a,b,c в представлении $y = a + be^{-cx}$ по зашумленным данным. Результаты расчетов иллюстрируются рис. 3.49.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize
def res(p, y, x):
    a, b, c = p
    err = y - a - b * np.exp(-c*x)
    return err
x = np.linspace(0., 1., 21)
a, b, c = 2, 1, 3
y = a + b * np.exp(-c*x)
np.random.seed(0)
y0 = y + 0.1 * np.random.randn(len(x))
plt.plot(x, y, '-', label='exact')
plt.plot(x, y0, '.', label='perturbation')
p0 = [0, 0, 0]
plsq = optimize.leastsq(res, p0,args=(y0, x))
```

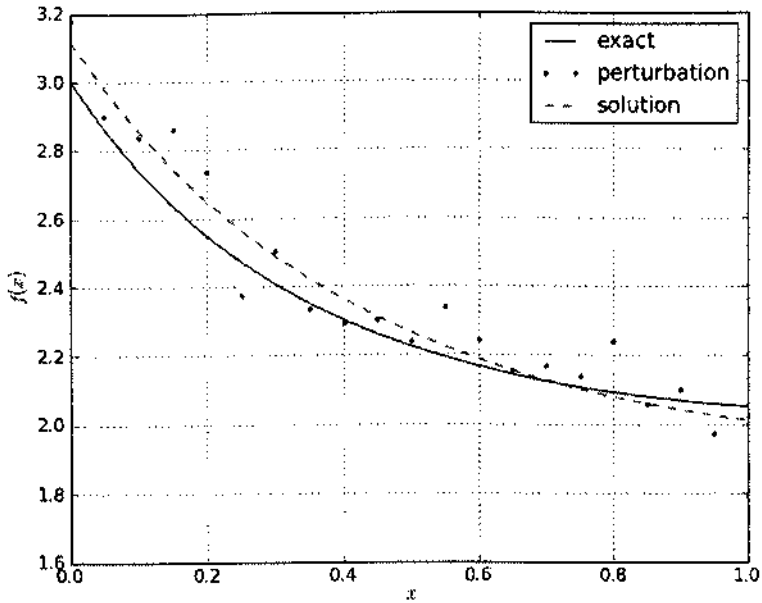

Рис. 3.49 Параметрическая идентификация

```
p1 = plsq[0]
print 'a, b, c:', p1
y1 = p1[0] + p1[1]*np.exp(-p1[2]*x)
plt.plot(x, y1, '--', label='solution')
plt.legend(loc=0)
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
plt.grid(True)
plt.show()
```

```
a, b, c: [ 1.90512141  1.21298056  2.43970231]
```

В более общем случае рассматриваются задачи минимизации с ограничениями, которые обычно формулируются в виде неравенств. В простейшем случае имеются ограничения на диапазоны независимых переменных. Для условной минимизации можно использовать функцию `fmin_cobyla()`.

Интегрирование и ОДУ

В модуле integrate пакета SciPy представлено программное обеспечение для решения двух классов задач вычислительной математики. Рассматриваются

проблемы вычисления интегралов непрерывных и сеточных функций. Второй класс задач связан с численным решением задачи Коши для системы обыкновенных дифференциальных уравнений.

Основная функция вычисления определенных интегралов от функции непрерывного аргумента есть `quad()`. В нашем примере (см. рис. 3.50) для заданной функции $f(x) = e^{-4x}\sin(4\pi x)$ вычисляется интеграл $j(x) = \int_0^x f(t)dt$ на равномерной сетке интервала $[0, 1]$.

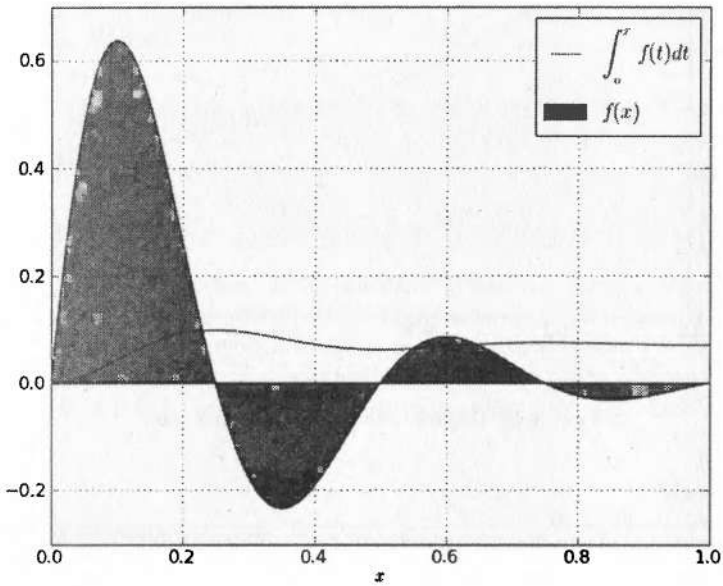

Рис. 3.50 Интегрирование функции $f(x) = e^{-4x}\sin(4\pi x)$

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
def f(x):
    return np.exp(-4*x)*np.sin(4*np.pi*x)
m = 101
x = np.zeros((m), 'float')
y = np.zeros((m), 'float')
y1 = np.zeros((m), 'float')
for i in range(0, m):
    x[i] = i/(m-1.)
    y[i] = integrate.quad(f, 0, x[i])[0]
    y1[i] = f(x[i])
plt.fill(x, f(x), 'g', label='$f(x)$')
```

```
plt.plot(x, y, label='$\\int_0^x f(t) dt$')
I0 = integrate.quad(f, 0, 1)[0]
I1 = integrate.trapz(y1, x)
I2 = integrate.simps(y1, x)
print 'int(quad): ', I0
print 'int(trapz):', I1
print 'int(simps):', I2
plt.legend(loc=0)
plt.xlabel('$x$')
plt.grid(True)
plt.show()
```

```
int(quad)   0.0709329489651
int(trapz)  0.0708301283866
int(simps)  0.0709330243013
```

Для сеточных функций используются квадратурные формулы трапеций и Симпсона (функции `trapz()` и `simps()` соответственно). На основе процедур интегрирования по одной переменной реализованы так же функции интегрирования по двум и трем переменным (`dblquad()` и `tplquad()`).

Отдельного внимания заслуживают возможности решения задачи Коши для систем обыкновенных дифференциальных уравнений. В модуле integrate для таких задач используются функции `ode()` и `odeint()`. Они предназначены для решения задач как для нежестких, так и жестких систем ОДУ.

Рассмотрим задачу Коши (модель Лотка–Вольтерра)

$$\frac{dy_0}{dt} = y_0 - y_0 y_1, \quad \frac{dy_1}{dt} = -y_1 + y_0 y_1, \quad 0 < t \le T, \quad y_0(0) = y_0^0, \quad y_1(0) = y_1^0.$$

Система уравнений записывается в векторном виде

$$\frac{dY}{dt} = F(Y,t), \quad 0 < t \le T$$

при

$$Y = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}, \quad F(Y,t) = \begin{bmatrix} y_0 - y_0 y_1 \\ -y_1 + y_0 y_1 \end{bmatrix}.$$

В программе решение получено с помощью функции `odeint()` на интервале $[0, 10]$ (см. рис. 3.50).

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
def f(y, t):
   return [y[0] - y[0]*y[1],
           -y[1] + y[0]*y[1]]
t = np.linspace(0, 10,  1000)
```

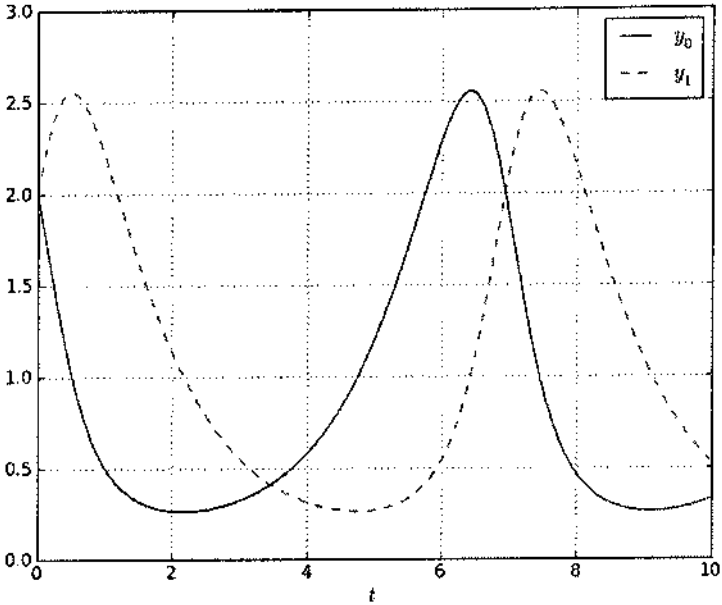

Рис. 3.51 Задача Коши для системы ОДУ

```
y0 = [2, 2]
Y = integrate.odeint(f, y0, t)
r0, r1 = Y.T
plt.plot(t, r0, '-', label='$y_0$')
plt.plot(t, r1, '--',  label='$y_1$')
plt.legend(loc=0)
plt.xlabel('$t$')
plt.grid(True)
plt.show()
```

При использовании функции `odeint()` для решения задачи Коши для систем ОДУ можно повысить вычислительную эффективность за счет аналитического задания матрицы Якоби (якобиан, производные правых частей системы).

3.5 Другие математические пакеты

Вы многое найдете, пакет символьных вычислений SymPy, разреженные матрицы (SciPy.sparse), PyAMG — многосеточный метод, элементы графического интерфейса пользователя (formlayout).

Вы многое найдете

В научных вычислениях могут понадобиться инструменты, которые не представлены в рассмотренном пакете SciPy или же возможности модулей этого пакета окажутся недостаточными. Прежде чем программировать что-то свое (программу на языке Python, включение сторонних программ на других языках) нужно посмотреть, кто и как подобные задачи уже решал.

Поиски во всемирной сети уместно начать с репозитария пакетов Python[25]. Вторая общая возможность связана с тенденцией обеспечения интерфейса на языке Python к прикладному программному, написанному на других языках программирования. Поэтому нужно обратиться к ресурсам по интересующему вас программному обеспечению. Здесь мы отметим некоторые пакеты Python, которые расширяют возможности SciPy[26] прежде всего по решению задач вычислительной математики.

Начнем мы с систем аналитических вычислений, которые традиционно хорошо представлены в таких общих математический инструментах, как пакет прикладных программ для решения задач технических вычислений MATLAB[27], программа для выполнения и документирования инженерных и научных расчётов Mathcad[28], системы компьютерной алгебры Mathematica[29] и Maple[30]. Наиболее полная поддержка символьных вычислений обеспечивается Sage[31]. Этот программный продукт позиционируется как свободно-программная альтернатива тех же Maple, Mathematica и MATLAB. Для иллюстрации поддержки символьных вычислений при работе на Python мы ограничимся возможностями пакета SymPy.

В пакете SciPy мы выделили модуль optimize, в котором решаются нелинейные уравнения и минимизируются функции. Более широкие возможности по задачам оптимизации предоставляет специализированный пакет OpenOpt[32]. Помимо всего в этом пакете реализован интерфейс с другими программными продуктами решения задач оптимизации. В частности, к пакету решения задач выпуклой оптимизации CVXOPT[33].

При численном решении задачи Коши для системы обыкновенных дифференциальных уравнений использовался модуль integrate пакета SciPy. Дополнительные возможности работы с такими математическими объектами предоставляет пакет Model Builder[34]. С использованием графической оболочки строятся математические модели, находиться численное решение и ви-

[25] Репозитарий пакетов — http://pypi.python.org/pypi
[26] Тематическое ПО — http://scipy.org/Topical_Software
[27] MATLAB — http://www.mathworks.com/
[28] Mathcad — http://www.ptc.com/appserver/mkt/products/home.jsp?k=3901
[29] Mathematica — http://www.wolfram.com/
[30] Maple — http://www.maplesoft.com/
[31] Sage — http://www.sagemath.org/index.html
[32] OpenOpt — http://openopt.org/Welcome
[33] CVXOPT — http://abel.ee.ucla.edu/cvxopt/
[34] Model Builder — http://model-builder.sourceforge.net/

зуализируются данные расчетов. Пакет PyDDE[35] позволяет найти решение систем обыкновенных дифференциальных уравнений с запаздыванием.

Для научных вычислений и прикладных исследований наибольший интерес представляют программные продукты, которые предназначены для решения краевых задач для уравнений с частными производными. В настоящее время необходимо ориентироваться на приближенное решение многомерных нелинейных задач в нерегулярных расчетных областях. В силу этого необходимо иметь возможность подготовки входных данных (область, уравнения, граничные условия), генерации расчетной сетки, аппроксимации (построения дискретной задачи), решения дискретных задач, обработки и визуализации расчетных данных.

Для подготовки сложных трехмерных геометрических моделей (расчетной области) используются CAD инструменты. Среди свободного программного обеспечения упомянем набор библиотек и программного обеспечения для 3D моделирования Open CASCADE[36]. На ее основе строятся многие как платные, так и бесплатные программные продукты. Отметим в этой связи открытую интегрируемую платформу для численного моделирования SALOME[37]. На основе Open CASCADE реализована Python 3D CAD библиотека pythonOCC[38].

При численном решении задач математической физики используются регулярные и нерегулярные (структурированные, блочно-структурированные и неструктурированные) сетки. Сеточные генераторы являются важной составной частью любого программного продукта, ориентированного на решение задач для уравнений с частными производными и они включены в общие инструменты численного моделирования. Среди пакетов Pethon, которые ориентированы на подготовку расчетных сеток, отметим MeshPy[39] Имеется возможность строить 2D и 3D неструктурированные сетки на основе разбиения Вороного (триангуляция Делоне).

Особое внимание уделяется программным продуктам для решения дискретных задач. При решении многомерных задач мы должны учитывать основную специфику таких проблем, которая связана прежде всего с разреженной структурой расчетных данных. Мы упоминали выше о модуле sparse в пакете SciPy, который дает инструменты для работы с разреженными матрицами. В частности, в модуле sparse.linalg представлены функции для решения систем уравнений с такими матрицами. Аналогичные возможности предоставляет пакет Pysparse[40]. Из других пакетов линейной алгебры, ориентированных на численное решение задач математической физики, стоит упомянуть

[35] PyDDE — http://users.ox.ac.uk/ clme1073/python/PyDDE/
[36] Open CASCADE — http://www.opencascade.org/
[37] SALOME — http://www.salome-platform.org/
[38] pythonOCC — http://www.pythonocc.org/
[39] MeshPy — http://mathema.tician.de/software/meshpy
[40] Pysparse — http://pysparse.sourceforge.net/

PyAMG[41], в котором реализован алгебраический многосеточный итерационный метод для решения сеточных задач.

Для решения задач большой размерности необходимо использовать компьютеры с параллельной архитектурой. Разработка программного обеспечения для решения задач на таких вычислительных многопроцессорных и/или многоядерных комплексах базируется на тех или иных алгоритмах распределенных вычислений. Среди специализированных инструментов Python можно отметить пакет mpi4py[42], который обеспечивает поддержку MPI (Message Passing Interface) стандарт для многопроцессорных систем. Для разбиения задачи на подзадачи используется пакет PyMetis[43].

Среди общих инструментов прикладного численного анализа отметим коллекцию библиотек Trilinos[44]. В настоящее время пакет PyTrilinos[45] обеспечивает интерфейс на языке Python к отдельным библиотекам этой коллекции. Пакет PySUNDIALS[46] дает возможность работы с другой популярной коллекцией программного обеспечения численного анализа SUNDIALS[47] (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers). В этой связи отметим, что предоставление интерфейса для работы на языке Python разрабатываемому прикладному программному обеспечению является общей тенденцией.

При проведении научных вычислений отдельного обсуждения заслуживает проблема визуализации расчетных данных. Особенно это актуально при оперировании с большими массивами данных на компьютерах параллельной архитектуры. Для большинства случаев может быть достаточно возможностей графической библиотеки Matplotlib, которая достаточно подробно обсуждалась нами выше, и им подобным (Gnuplot, DISLIN, Chaco). Для научной визуализации трехмерных данных (скаляров, векторов) можно ориентироваться на более мощные инструменты, такие, например, как Mayavi2[48] и Visualization Toolkit (VTK)[49]. Вторая возможность связана с записью расчетных данных в надлежащем формате и использованию сторонних программных продуктов для их визуализации. Хорошим примером программы визуализации данных расчетов является ParaView[50], – свободно распространяемая программа с открытым кодом для параллельной и интерактивной научной визуализации.

Имеется ряд интегрированных программных продуктов, которые обеспечивают численное решение краевых задач для уравнений с частными производными. Так например, DOLFIN[51] обеспечивает интерфейс на языке Python

[41] PyAMG — http://code.google.com/p/pyamg/
[42] mpi4py — http://code.google.com/p/mpi4py/
[43] PyMetis — http://mathema.tician.de/node/423
[44] Trilinos — http://trilinos.sandia.gov/
[45] PyTrilinos — http://trilinos.sandia.gov/packages/pytrilinos/index.html
[46] PySUNDIALS — http://pysundials.sourceforge.net/
[47] SUNDIALS — https://computation.llnl.gov/casc/sundials/main.html
[48] Mayavi2 — http://code.enthought.com/projects/mayavi/
[49] VTK — http://www.vtk.org/
[50] ParaView — http://www.paraview.org/
[51] DOLFIN — http://www.fenics.org/wiki/DOLFIN

для коллекции свободных программных продуктов FEniCS[52] — инструменты для работы с сетками, конечно-элементными аппроксимациями уравнений с частными производными и решения дискретных задач. В качестве второго примера программного инструментария для решения многомерных нелинейных задач на основе метода конечных элементов отметим Escript[53]. В пакете FiPy[54] для решения стационарных и нестационарных задач математической физики применяется метод конечных объемов.

При разработке современных программных продуктов большое внимание уделяется графическому интерфейсу пользователя. На этот элемент программирования при проведении научных расчетов практически часто не обращается внимания. Однако совсем отказываться от удобств, который предоставляет GUI (Graphical user interface) вряд ли целесообразно. На примере пакета formlayout мы проиллюстрируем разработку программ на языке Python с минимальным графическим интерактивным вводом параметров.

Пакет символьных вычислений SymPy

Научные вычисления базируются на получении приближенного численного решения задачи. Они могут дополняться символьными вычислениями, которые подразумевают преобразования и работу с математическими равенствами и формулами, которые рассматриваются как последовательность символов. Системы аналитических вычислений (системы компьютерной алгебры) используются для символьного интегрирования и дифференцирования, вычисления пределов, подстановки одних аналитических выражений в другие, упрощения формул и т.д.

Символьные вычисления в Python поддерживаются пакетом SymPy[55]. Это полнофункциональная и одновременно простая для использования и легко расширяемая система компьютерной алгебры. Пакет SymPy полностью написан на Python и при использовании не требует никаких внешних библиотек.

Для задания одной символьной переменной используется функция `Symbol()`, нескольких — `symbols()` или `var()`. При работе с числами с плавающей запятой имеется возможность выбора точности вычислений с использованием функции `N()` (арифметика произвольной точности). Поддерживаются вычисления с комплексными числами (в SymPy $\sqrt{-1} = \mathrm{I}$, $e = \mathrm{E}$). Для представления дробей используется функция `Rational()`. Дополнительные возможности при выводе результатов предоставляет функция `pprint()`. Следующий пример иллюстрирует отмеченные возможности пакета SymPy.

```
import sympy as sm
x, y = sm.symbols ('xy')
```

[52] FEniCS — http://www.fenics.org/wiki/FEniCS_Project
[53] Escript — https://launchpad.net/escript-finley
[54] FiPy — http://www.ctcms.nist.gov/fipy/
[55] SymPy — http://code.google.com/p/sympy/

```
print 'pi:', sm.N(sm.pi, 50)
print 'complex:', sm.N(1/(2 + sm.I), 50)
x = sm.Rational(1, 2)
y = sm.Rational(3, 7)
sm.pprint(x + y)
```

```
pi: 3.1415926535897932384626433832795028841971693993751
complex: 0.4 - 0.2*I
13
--
14
```

Поддерживаются основные символьные операции с выражениями. Для упрощения символьного выражения применяется функция `simplify()`. Для упрощения тригонометрических функций используется `trigsimp()`, для численных упрощений — `nsimplify()`.

```
import sympy as sm
x = sm.Symbol('x')
s1 = (1-2*x)*(3+x) + x**2
print 'simplify:', sm.simplify(s1)
s2 = sm.sin(x)**2 + sm.cos(x)**2
print 'trigsimp:', sm.trigsimp(s2)
s3 = sm.nsimplify(sm.pi*7, tolerance=0.01)
print 'nsimplify:', s3
```

```
simplify: 3 - 5*x - x**2
trigsimp: 1
nsimplify: 22
```

Раскрытие скобок в символьных выражениях обеспечивается функцией `expand()` (ниже приведен пример для полиномов и тригонометрических функций комплексного переменного). Отметим также функцию `apart()` для разложения на простые дроби.

```
import sympy as sm
x = sm.Symbol('x')
y = sm.Symbol('y')
s1 = (x+y)**3
print 'expand (basic):', sm.expand(s1)
s2 = sm.sin(x + sm.I*y)
print 'expand (complex):', sm.expand(s2, complex=True)
s3 = 2/( (x-1)*(x+1) )
print 'apart:', sm.apart (s3, x)
```

```
expand (basic): 3*x*y**2 + 3*y*x**2 + x**3 + y**3
expand (complex): cosh(im(x) + re(y))*sin(-im(y) + re(x))
 + I*cos(-im(y) + re(x))*sinh(im(x) + re(y))
```

```
apart   -1/(1 + x) - 1/(1 - x)
```

В пакете Sympy поддерживаются основные операции математического анализа. Для аналитического дифференцирования используется функция `diff()`, для интегрирования — `integrate()`.

```
import sympy as sm
x = sm.Symbol('x')
s1 = sm.log(1+x) + 1/(1+x)
print 'diff:', sm.diff(s1, x)
s2 = sm.log(x**2)
print 'integrate:', sm.integrate(s2, x)
s3 = x / (x**2 + 2*x+1)
print 'integrate (0,1):', sm.integrate(s3, (x, 0, 1))
```

```
diff   1/(1 + x) - 1/(1 + x)**2
integrate: -2*x + 2*x*log(x)
integrate (0,1)  -1/2 + log(2)
```

Вычисление пределов обеспечивается функцией `limit()`. Для разложения функции в ряд Тейлора в заданной точке используется `series()`.

```
import sympy as sm
x = sm.Symbol('x')
s1 = x**x
print 'limit (x->0):', sm.limit(s1, x, 0)
s2 = sm.cos(x)
print 'series:', s2.series(x, 0, 8)
```

```
limit (x->0): 1
series  1 - x**2/2 + x**4/24 - x**6/720 + O(x**8)
```

Для решения алгебраических уравнений и систем в пакете Sympy используется функция `solve()` (уравнения записываются в виде $f_i(x_0, x_1, x_m), i = 0, 1, ..., m$). Кроме того имеется возможность получения аналитического решения задачи Коши для обыкновенных дифференциальных уравнений — функция `dsolve()`. В примере ищется общее решение уравнения

$$\frac{d^2u}{dx^2} + u = 0.$$

```
import sympy as sm
x, y = sm.symbols('xy')
u = sm.Function('u')
s1 = x**4 - 1
print 'solve:', sm.solve(s1, x)
s2 = (x**2 + y**2 - 1, x - y + 1)
print 'solve (system):', sm.solve(s2, x, y)
eq = sm.Derivative(u(x), x, x) + u(x)
```

```
print 'dsolve:', sm.dsolve(eq, u(x))
```

```
solve: [1, -1, -I, I]
solve (system): [(-1, 0), (0, 1)]
dsolve: C1*sin(x) + C2*cos(x)
```

Отметим некоторые возможности аналитического решения задач линейной алгебры. Матрицы могут задаваться на основе явного вычисления элементов матрицы (функция `Matrix()`) и с использованием функций `eye()`, `zeros()`, `ones()`. Имеются функции для выполнения LU и QR разложения, поддерживаются возможности решения систем линейных уравнений в аналитическом виде.

```
import sympy as sm
def f(i,j):
    return 4 - (i-j)**2
M1 = sm.ones(3)
print 'M1:\n', M1
M2 = sm.Matrix(3, 3, f)
print 'M2:\n', M2
s = M1.dot(M2)
print 'M1.dot(M2):', s
d = M1.det()
print 'det(M1):', d
M3 = M2.inv()
print 'M3:\n', M3
Q, R = M3.QRdecomposition()
print 'Q:\n', Q
print 'R:\n', R
```

```
M1:
[1, 1, 1]
[1, 1, 1]
[1, 1, 1]
M2:
[4, 3, 0]
[3, 4, 3]
[0, 3, 4]
M1.dot(M2): 24
det(M1): 0
M3:
[-7/8,  3/2, -9/8]
[ 3/2,   -2,  3/2]
[-9/8,  3/2, -7/8]
Q:
[-7*274**(1/2)/274,    15*274**(1/2)/274,    0]
[ 6*274**(1/2)/137,    14*274**(1/2)/685, 3/5]
```

```
[-9*274**(1/2)/274, -21*274**(1/2)/1370, 4/5]
R.
[274**(1/2)/8, -24*274**(1/2)/137, 135*274**(1/2)/1096]
[           0,   5*274**(1/2)/274,  -12*274**(1/2)/685]
[           0,                  0,                 1/5]
```

В пакете Sympy поддерживается работа с полиномами. Например, функция `div()` предназначена для деления полиномов с остатком. Для разложения в виде множителей с рациональными коэффициентами используется функция `factor()`. Корни полиномов находятся с помощью `factor()`.

```
import sympy as sm
x = sm.Symbol('x')
a, b, c = sm.symbols('abc')
f = a*x**2 + b*x + c
g = 2*x + 1
q, r = sm.div(f, g, x)
print 'q:', q
print 'remainder:'
sm.pprint(r)
p1 = x**3 - 1
p2 = sm.factor(p1)
print 'factor(x^3 - 1):\n', p2
p3 = x**3 + 2*x + 3
print 'roots:', sm.solve(p3, x)
```

```
q: b/2 - a/4 + a*x/2
remainder:
    b   a
c - - + -
    2   4
factor(x^3 - 1):
-(1 - x)*(1 + x + x**2)
roots [1/2 - I*11**(1/2)/2, -1, 1/2 + I*11**(1/2)/2]
```

Пакет Sympy (модуль statistics) имеет средства для работы со случайными величинами. Функция `Normal()` генерирует нормальное распределение с заданным средним значением и заданным стандартным отклонением. Функция плотности вероятности рассчитывается с помощью функция `Uniform()`. Для равномерно распределенных величин используется `pdf()`. На рис.3.52 представлены графики нормально распределенных случайных величин и соответствующая функция распределения, которые получены следующей программой.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy.statistics as st
```

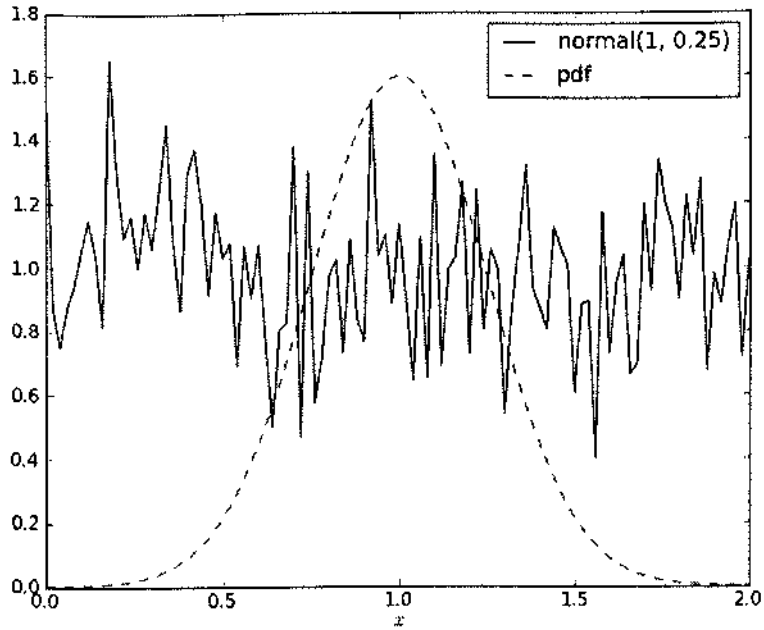

Рис. 3.52 Нормально распределенные случайные величины

```
N = st.Normal(1, 0.25)
m = 101
x = np.linspace(0., 2., m)
y1 = np.zeros((m), 'float')
y2 = np.zeros((m), 'float')
for i in range(m):
    y1[i] = N.random()
    y2[i] = N.pdf(x[i])
plt.plot(x, y1, '-', label='normal(1, 0.25)')
plt.plot(x, y2, '--', label='pdf')
plt.xlabel('$x$')
plt.legend(loc=1)
plt.show()
```

Геометрический модуль geometry предоставляет возможности аналитической работы с точками, линиями, сегментами, эллипсами и полигонами. Можно находить площади элементов, решать некоторые задачи вычислительной геометрии, такие как нахождение точек пересечения отрезков. Здесь приведен пример создание треугольника и круга, вычисления их площадей. Более сложные примеры вы найдете в документации по пакету Sympy.

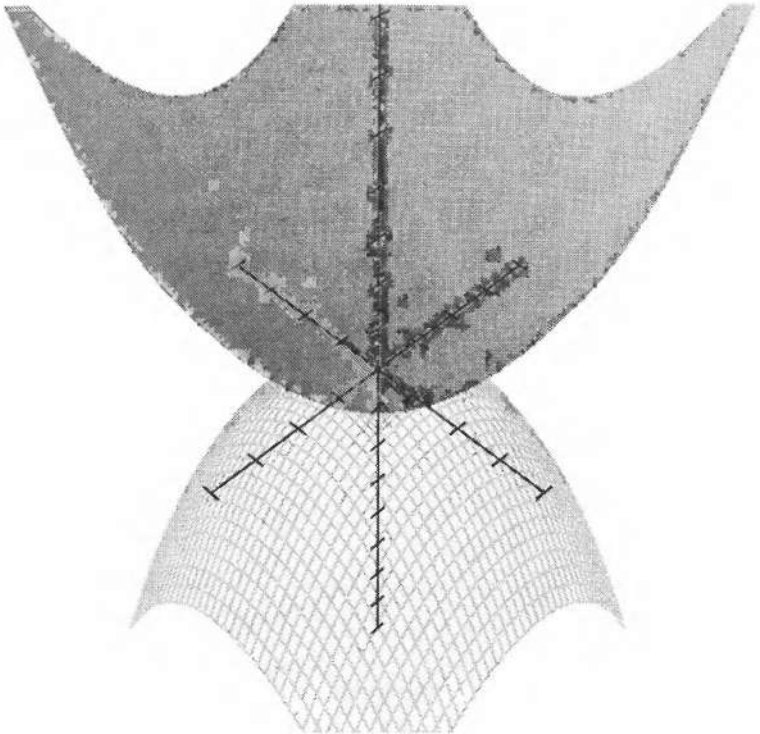

Рис. 3.53 Визуализация в Sympy

```
import sympy as sm
import sympy.geometry as gm
r = sm.Symbol ('r')
x = gm.Point(0, 0)
y = gm.Point(0, 2)
z = gm.Point(1, 0)
t = gm.Triangle(x, z, y)
c = gm.Circle(x, r)
print 'triangle area:', t.area
print 'circle area:', c.area
```

```
triangle area  1
circle area. pi*r**2
```

Многоплановость аналитически решаемых в пакете Sympy задач дополняется графическими возможностями. Отличительная особенность связана с работой с аналитическими объектами. Имеется возможность визуализации в интерактивном окне одномерных и двумерных функций.

```
import sympy as sm
sm.var('x y')
p = sm.Plot(visible=False)
p[1] =  x**2 + y**2
p[1].style = 'solid'
p[2] = - x**2 - y**2
p[2].style = 'wireframe'
p.show()
```

Рис. 3.53 получен как копия экрана (горячая клавиша F8) при установленном пакете Python Imaging Library (PIL)[56].

Разреженные матрицы (SciPy.sparse)

Разреженные матрицы, которые характеризуются большим числом нулевых элементов, возникают при численном решении краевых задач для уравнений с частными производными. В этом случае при использовании конечно-разностных или конечно-элементных аппроксимаций имеется зависимость решения дискретной задачи только от небольшого числа соседей. Вычислительные алгоритмы линейной алгебры должны учитывать отмеченную специфику разреженных матриц. Здесь мы отметим возможности пакета SciPy (модуль sparse).

В качестве иллюстративного примера рассмотрим разностную задачу Дирихле для уравнения Пуассона в единичном квадрате. Функция $u(\mathbf{x})$, $\mathbf{x} = (x_1, x_2)$, удовлетворяет в $\Omega = \{\mathbf{x} \mid 0 < x_\alpha < 1,\ \alpha = 1, 2$ уравнению

$$-\frac{\partial^2 u}{\partial x_1^2} - \frac{\partial^2 u}{\partial x_1^2} = \varphi(\mathbf{x}), \quad \mathbf{x} \in \Omega$$

и однородным граничным условиям

$$u(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\Omega.$$

Для приближенного решения введем равномерную разностную сетку с шагом h ($Nh = 1$) по каждому направлению. Разностное решение $y_{ij}, i = 0, 1, ..., N$, $j = 0, 1, ..., N$ определяется как решение сеточной задачи

$$4y_{i,j} - y_{i-1,j} - y_{i+1,j} - y_{i,j-1} - y_{i,j+1} = f_{i,j}, \quad i = 1, 2, ..., N-1,\ j = 1, 2, ..., N-1,$$

$$u_{i,j} = 0, \quad i = 1, N,\ j = 0, N,$$

где $f_{i,j} = h^2 f(ih, jh)$.

Сформулируем матричную задачу для определения разностного решения во внутренних узлах сетки. Нумерацию узлов проведем по правилу

$$j = 1, 2, ..., N-1: \quad i = 1, 2, ..., N-1: \quad v_k = y_{ij}, \quad k = i + (N-1)j.$$

[56] Python Imaging Library — http://www.pythonware.com/products/pil/

Общее число неизвестных равно $(N-1)^2$. В представлении

$$Av = b$$

разреженная матрица имеет диагональную структуру (ненулевые элемент содержат только пять диагоналей). Вместо $O(N^4)$ нужно хранить толь $O(N^2)$ ненулевых элементов.

Для работы с разреженными матрицами используются специальные форм ты хранения матриц. Простейший из них COOrdinate format (COO) связан использованием трех массивов. Первый из них содержит значения ненулеви элементов ($a_{ij} \neq 0$), второй — номер строки (i), а третий — номер столби (i), т.е. храниться тройка (a_{ij}, i, j).

Для выполнения основных операций над разреженными матрицами (слож ние, умножение, транспонирование, решение систем линейных уравнений) б лее удобен разреженный строчный формат — Compressed Sparse Row forma (CSR). В этом случае значения ненулевых элементов и номера столбцов хр нятся по строкам в двух массивах: a_{ij} (массив AA) и j (массив JA). Трети массив (обозначим его IA) отвечает за номер строки: его i-й элемент указы вает, с какой позиции в массивах элементов и столбцов (AA и JA) начинает i-я строка. В силу этого $IA(i+1) - IA(i)$ есть число элементов в i-й строк

Имеются также варианты разреженного формата для хранения по столбца — Compressed Sparse Column format (CSC), по блокам, для симметричных ленточных матриц. Основные форматы поддерживаются в модуле sparse п кета SciPy.

Первая проблема при работе с разреженными матрицами связана с заданис матрицы в нужном формате. Мы можем начать с простейшего способа, к торый связан с заданием матрицы как полной и последующей конвертации нужный формат. Аналогично стандартному массиву для инициализации ра реженной матрицы используется `lil\_matrix()`, после этого можно задава отдельные ненулевые элементы. Для больших матриц нужно ориентироват ся на явное задание матрицы в разреженном формате.

Для работы с Compressed Sparse Row форматом в модуле sparse предназначе класс `csr\_matrix()`, для Compressed Sparse Column — `csc\_matrix()`, дл DIAgonal format — `dia\_matrix()`, для COOrdinate format — `coo\_matrix()`.

```
from scipy import sparse
import matplotlib.pyplot as plt
import time
def poisson_lil(n):
    A = sparse.lil_matrix((n*n, n*n))
    for j in range(n):
        for i in range(n):
            k = i + n*j
            A[k,k] = 4
```

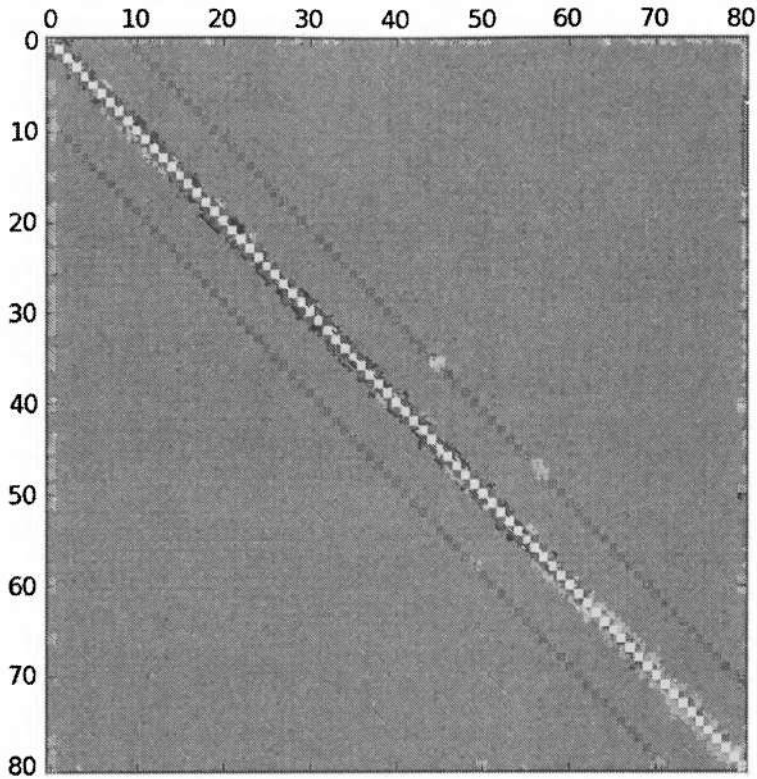

Рис. 3.54 Портрет разностного оператора Лапласа

```
            if i > 0:
                A[k,k-1] = -1
            if i < n-1:
                A[k,k+1] = -1
            if j > 0:
                A[k,k-n] = -1
            if j < n-1:
                A[k,k+n] = -1
    return A
def poisson_coo(n):
    V = []
    I = []
    J = []
    for j in range(n):
```

```
        for i in range(n):
            k = i + n*j
            V.append(4.)
            I.append(k)
            J.append(k)
            if i > 0:
                V.append(-1.)
                I.append(k)
                J.append(k-1)
            if i < n-1:
                V.append(-1.)
                I.append(k)
                J.append(k+1)
            if j > 0:
                V.append(-1.)
                I.append(k)
                J.append(k-n)
            if j < n-1:
                V.append(-1.)
                I.append(k)
                J.append(k+n)
    return sparse.coo_matrix((V, (I,J)))
N = 10
tst = time.clock()
A = poisson_lil(N-1)
dt = time.clock() - tst
print 'time (lil):', dt
tst = time.clock()
B = poisson_coo(N-1)
dt = time.clock() - tst
print 'time (coo):', dt
C = B.todense()
R = A.todense() - C
print R
plt.matshow(C)
tst = time.clock()
D = sparse.lil_matrix(B)
dt = time.clock() - tst
print 'time (coo->lil):', dt
plt.show()
```

```
time (lil): 0.0123241920412
time (coo) 0.000795631847064
[[ 0.  0.  0. ,  0.  0.  0.]
 [ 0.  0.  0. ,  0.  0.  0.]
```

```
 [ 0.   0.   0.     ,  0.   0.   0.]
  ..,
 [ 0.   0    0.     ,  0.   0.   0.]
 [ 0.   0    0      ,  0.   0    0 ]
 [ 0.   0.   0.     ;  0.   0.   0 ]]
time (coo->lil)  0.00115880649636
```

Здесь разреженная матрица соответствует разностному оператору Лапласа на стандартном пятиточечном шаблоне. Как показывает этот пример (см. портрет матрицы на рис. 3.54), формирование матрицы в формате COO (функция `poisson\_coo()`) значительно быстрее, чем на основе полной матрицы (`poisson\_lil()`). После инициализации можно записать разреженную матрицу в необходимом формате.

Проиллюстрируем возможности решения систем линейных уравнений с разреженными матрицами методами модуля sparse.linalg на примере нашей модельной сеточной эллиптической задачи (рис. 3.55).

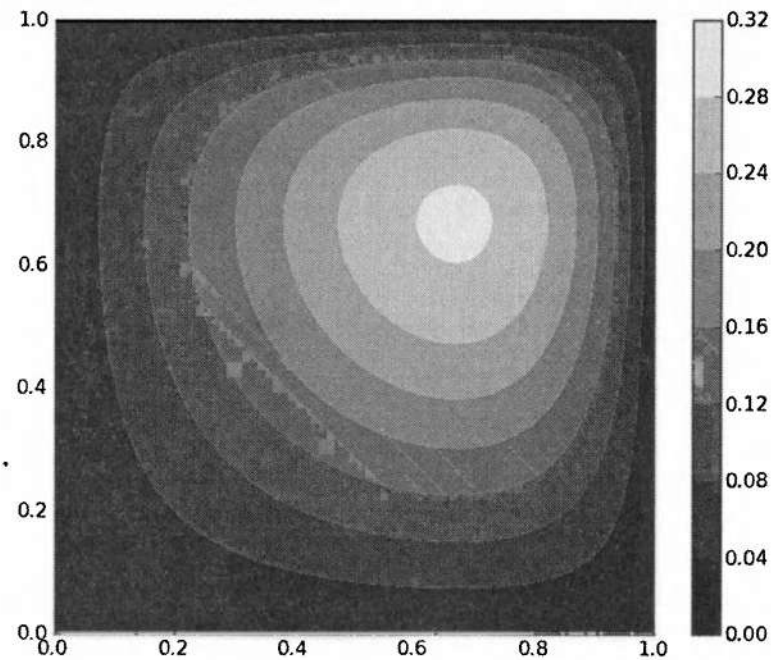

Рис. 3.55 Разностное решение на сетке 800×800

```
import numpy as np
from scipy import sparse
from scipy.sparse import linalg
```

```
import matplotlib.pyplot as plt
import time
def f(x, y):
    return 20*x*y*(x**2+y**2)*(1-x**2*y**2)
def g(x, y):
    return x*(1-x**4)*y*(1-y**4)
def poisson(n, h):
    V = []
    I = []
    J = []
    b = []
    u = []
    for j in range(n):
        for i in range(n):
            k = i + n*j
            b.append(h**2*f(i*h+h, j*h+h))
            u.append(g(i*h+h, j*h+h))
            V.append(4.)
            I.append(k)
            J.append(k)
            if i > 0:
                V.append(-1.)
                I.append(k)
                J.append(k-1)
            if i < n-1:
                V.append(-1.)
                I.append(k)
                J.append(k+1)
            if j > 0:
                V.append(-1.)
                I.append(k)
                J.append(k-n)
            if j < n-1:
                V.append(-1.)
                I.append(k)
                J.append(k+n)
    return sparse.coo_matrix((V, (I,J))), b, u
Nlist = [100, 200, 400, 800]
for N in Nlist:
    h = 1. / N
    A, b, u = poisson(N-1, h)
    B = A.tocsr()
    tst = time.clock()
    v, info = linalg.cg(B, b, tol=1.e-08)
    dt = time.clock() - tst
```

```
    er = np.linalg.norm(v-u, ord=np.inf)
    if info == 0:
        print "N = %i, time solution = %1.3e, error = %1.3e" % \
              (N, dt, er)
x = np.linspace(h/2, 1.-h/2, N-1)
y = np.linspace(h/2, 1.-h/2, N-1)
X, Y = np.meshgrid(x, y)
Z = np.reshape(v, (N-1,N-1))
plt.contourf(X, Y, Z, cmap=plt.cm.gray)
plt.colorbar()
plt.show()
```

```
N = 100, time solution = 1.226e-01, error = 3.299e-05
N = 200, time solution = 1.225e+00, error = 8.247e-06
N = 400, time solution = 1.148e+01, error = 2.062e-06
N = 800, time solution = 1.157e+02, error = 5.155e-07
```

Для приближенного решения сеточной задачи используется стандартный итерационный метод сопряженных градиентов. Точное решение дифференциальной задачи задается функцией $g(x, y)$, соответствующая правая часть уравнения Пуассона — $f(x, y)$. Точность численного решения в равномерной норме на последовательности расчетных сеток хорошо согласуется с теоретической оценкой ($\varepsilon = O(N^{-2})$).

В табл. 3.11 сравниваются временные затраты (в секундах) на решения модельной задачи при использовании прямого метода на основе LU-разложение (функция `spsolve()`), обобщенного метода минимальных невязок (функция `gmres()`) и метода сопряженных градиентов (функция `cg()`).

Таблица 3.11 *Решение сеточной задачи*

N	spsolve	gmres	cg
100	4.930e+00	1.277e+00	1.226e-01
200	7.079e+01	2.233e+01	1.225e+00

На рассматриваемой задаче с симметричной положительно определенной матрицей максимальная скорость сходимости наблюдается при использовании итерационного метода сопряженных градиентов. Потенциальное преимущество прямого метода при относительно небольших N проявляется на задачах с более общими матрицами.

PyAMG — многосеточный метод

К классу наиболее эффективных итерационных методов для приближенного решения сеточных задач, к которым мы приходим при конечно-разностной или конечно-элементной аппроксимации краевых задач для эллиптических

уравнений, относятся многосеточные методы. Они базируются на последовательном проведении итераций на различных, вложенных друг в друга сетках. Особый интерес представляют алгебраические многосеточные методы, которые непосредственно не привязываются к расчетным сеткам, а оперируют с элементами матрицы и их окружением.

В пакете PyAMG[57] реализован алгебраический многосеточный итерационный метод на языке Python. Пакет PyAMG базируется на модуле sparse пакета SciPy и дополняет его возможности по итерационному решению систем линейных уравнений.

Многосеточный метод основан на выборе последовательности сеток, операторов интерполяции и сглаживания для перехода на новую сетку. На каждой сетке делается несколько итераций с использованием, например, метода Гаусса–Зейделя. Подготовка иерархии сеток и соответствующих процедур перехода с одной сетки на другую в пакете PyAMG проводится с помощью функции `ruge_stuben_solver()`, которая имеет один обязательный аргумент – матрицу системы, которая записана в разреженном строчном формате CSR. Здесь могут задаваться также, например, максимальное число уровней многосеточного метода, максимальная размерность матрицы на самом низком уровне.

Итерационное решение системы линейных уравнений проводится с использованием `solve()` по задаваемому на входе массиву правой части уравнения. Можно задавать относительную невязку решения (`tol`) для завершения итерационного процесса и максимальное число итераций (`maxiter`). Алгоритм перехода на новую итерацию контролируется параметром выбора цикла вычислений `cycle`, который может принимать значения `'W'`, `'V'`, `'F'` и соответствуют W-циклу, V-циклу и F-циклу многосеточного метода (по умолчанию используется V-цикл). Дополнительное ускорение итераций многосеточного метода может осуществляться на основе метода сопряженных градиентов (параметр `accel` равен `'cg'`) или на основе обобщенного метода минимальных невязок (`accel='gmres'`).

В приведенном примере решается та же сеточная эллиптическая задача Дирихле для уравнения Пуассона в квадрате, которая рассматривалась нами выше при иллюстрации возможностей модуля sparse пакета SciPy.

```
import numpy as np
from scipy import sparse
import matplotlib.pyplot as plt
import pyamg as amg
import time
def f(x, y):
    return 20*x*y*(x**2+y**2)*(1-x**2*y**2)
def poisson(n, h):
```

[57] PyAMG — http://code.google.com/p/pyamg/

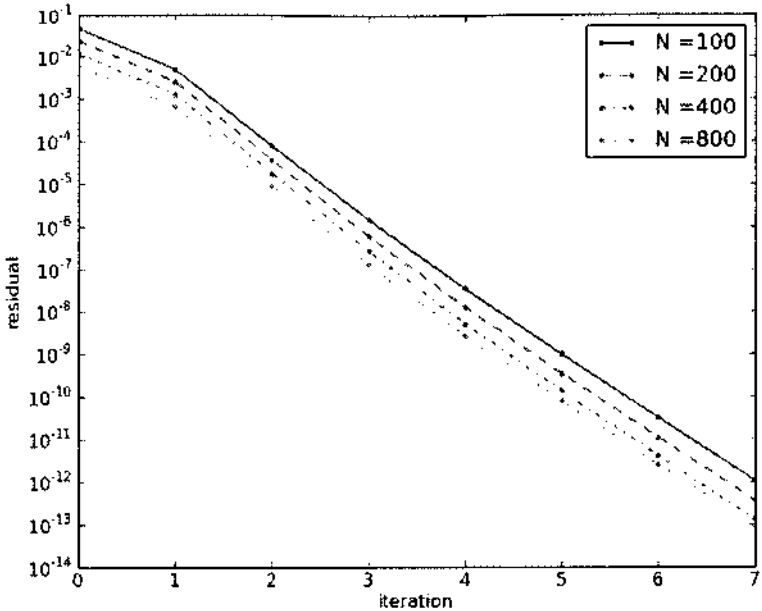

Рис. 3.56 Сходимость многосеточного метода на различных сетках

```
V = []
I = []
J = []
b = []
for j in range(n):
    for i in range(n):
        k = i + n*j
        b.append(h**2*f(i*h+h, j*h+h))
        V.append(4.)
        I.append(k)
        J.append(k)
        if i > 0:
            V.append(-1.)
            I.append(k)
            J.append(k-1)
        if i < n-1:
            V.append(-1.)
            I.append(k)
            J.append(k+1)
        if j > 0:
```

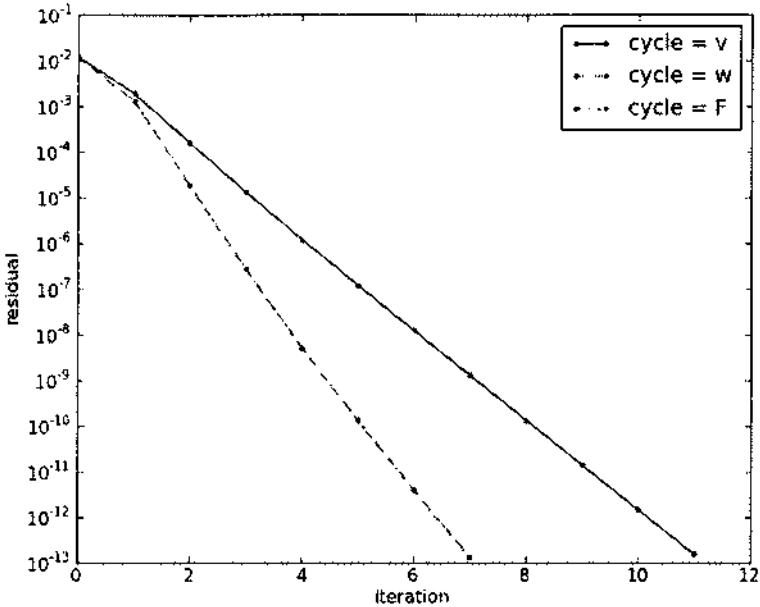

Рис. 3.57 Сходимость при использовании различных циклов

```
                V.append(-1.)
                I.append(k)
                J.append(k-n)
            if j < n-1:
                V.append(-1.)
                I.append(k)
                J.append(k+n)
    return sparse.coo_matrix((V, (I,J))), b
plt.figure(1)
Nlist = [100, 200, 400, 800]
sglist = ['.-', '.--', '.:', '.-.']
for k in range(len(Nlist)):
    N = Nlist[k]
    h = 1. / N
    A, b = poisson(N-1, h)
    A = A.tocsr()
    b = np.array(b)
    tst = time.clock()
    ml = amg.ruge_stuben_solver(A)
    dt = time.clock() - tst
    res = []
```

```
    x = ml.solve(b, tol=1.0e-8, residuals=res, cycle='W')
    dt1 = time.clock() - tst
    print "N = %i, time = %1.3e (%1.3e + %1.3e)" % \
          (N, dt1, dt, dt1-dt)
    res = np.array(res)
    sl = 'N =' + str(N)
    sg = sglist[k]
    plt.semilogy(res, sg, label=sl)
plt.xlabel('iteration')
plt.ylabel('residual')
plt.legend(loc=0)
plt.figure(2)
N = 400
cclist = ['v', 'w', 'F']
for kk in range(len(cclist)):
    cc = cclist[kk]
    h = 1. / N
    A, b = poisson(N-1, h)
    A = A.tocsr()
    b = np.array(b)
    tst = time.clock()
    ml = amg.ruge_stuben_solver(A)
    dt = time.clock() - tst
    res = []
    x = ml.solve(b, tol=1.0e-8, residuals=res, cycle=cc)
    dt1 = time.clock() - tst
    sl = 'cycle = ' + cc
    sl1 = sl + ','
    print sl1, "time = %1.3e (%1.3e + %1.3e)" % \
          (dt1, dt, dt1-dt)
    res = np.array(res)
    sg = sglist[kk]
    plt.semilogy(res, sg, label=sl)
plt.xlabel('iteration')
plt.ylabel('residual')
plt.legend(loc=0)
plt.show()
```

```
N = 100, time = 4.037e-001 (3.255e-002 + 3.712e-001)
N = 200, time = 8.019e-001 (1.037e-001 + 6.983e-001)
N = 400, time = 2.308e+000 (4.424e-001 + 1.865e+000)
N = 800, time = 1.054e+001 (2.087e+000 + 8.457e+000)
cycle = v, time = 2.155e+000 (6.138e-001 + 1.541e+000)
cycle = w, time = 2.388e+000 (4.980e-001 + 1.890e+000)
cycle = F, time = 2.559e+000 (5.461e-001 + 2.013e+000)
```

На рис. 3.56 показана абсолютная невязка на отдельных итерациях многосеточного метода на различных расчетных сетках. Приведенные данные иллюстрируют независимость числа итераций от сетки — оптимальность итерационного метода. Влияние способа организации вычислений иллюстрируется рис. 3.57 (в нашем примере использование W-цикла и F-цикла дает практически одни и те же результаты).

Элементы графического интерфейса пользователя (formlayout)

Современные программы, которые ориентированы на стороннего пользователя, базируются на использовании графического интерфейса пользователя (GUI, Graphical user interface). В GUI такие элементы интерфейса как меню, кнопки, списки, представлены пользователю на дисплее и выполнены в виде графических изображений. Пользователь с помощью устройств ввода (клавиатуры, мыши и т.п.) имеет доступ ко всем видимым экранным объектам (элементам интерфейса) и осуществляет непосредственное манипулирование ими. В программах, предназначенных для научных вычислений, этому моменту часто не придается большого значения и до сих пор часто ограничиваются простейшим пользовательским интерфейсом командной строки.

В стандартной библиотеке Python возможности построения графического интерфейса пользователя реализованы в модуле Tkinter. Tkinter — это встроенная графическая библиотека на основе средств Tk, которая широко распространена в UNIX подобных систем и портирована на Windows, Mac OS X. Реализованы основные элементы GUI, которые часто называют виджетами (widget): окно верхнего уровня (Toplevel), рамка (Frame), меню (Menu), кнопка-меню (Menubutton) надпись (Label), поле ввода (Entry), рисунок (Canvas), кнопка (Button), селекторная кнопка (Radiobutton), флажок (Checkbutton), шкала (Scale), список (Listbox), полоса прокрутки (Scrollbar), форматированный текст (Text), сообщение (Message).

Как альтернативу Tkinter можно рассматривать WxPython[58]. Это бесплатный кроссплатформенный программный инструментарий для создания графический интерфейс на языка программирования Python. WxPython реализован в виде модуля расширения Python (родной код) популярной и мощной кроссплатформенной графической библиотеки WxWidgets, которая написана на C++. Для более удобной работы можно использовать редакторы интерфейса (дизайнеры), Примером GUI-дизайнера для библиотеки wxPython является wxGlade[59].

Отметим также кроссплатформенный инструментарий Qt[60] для разработки программного обеспечения на языке программирования C++. Библиотека

[58] WxPython — http://wxpython.org/
[59] wxGlade — http://wxglade.sourceforge.net/
[60] Qt — http://qt.nokia.com/

Qt включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, в том числе и элементы графического интерфейса пользователя. Привязка к языку Python осуществляется в PyQt[61] и работает на всех платформах, поддерживаемых Qt, включая Windows, Mac OS X и GNU/Linux. Привязка реализована в виде набора модулей Python, содержится более 300 классов и более 6000 функций и методов. Имеется визуальная среда разработки графического интерфейса Designer, которая позволяет создавать диалоги и формы в визуальном режиме (GUI-дизайнер), справочная система Assistant упрощает работу с документацией по библиотеке и позволяет создавать кроссплатформенную справку для разрабатываемого на основе Qt программного обеспечения.

Упрощение коммуникации между пользователем и компьютером является основной задачей графического интерфейса. При разработке программного обеспечения для поддержки научных вычислений типичной является ситуация с параметрическим вводом параметров используемой математической модели и вычислительного алгоритма. Поэтому минимальный графический интерфейс должен обеспечить эту базовую возможность. В дальнейшем такие интерактивные возможности ввода параметров можно расширить до более полноценного GUI.

Модуль FormLayout[62] решает задачу создания диалогов для редактирования параметров. Он базируется на PyQt и обеспечивает графический интерфейс без написания какого-либо кода для соответствующих виджетов.

Приведем пример программы с графическим интерфейсом для ввода параметров. Будем рассматривать краевую задачу для одномерного нелинейного параболического уравнения второго порядка. Пусть $u(x,t)$ определяется из решения уравнения

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2} + f(u,t), \quad 0 < x < l, \quad 0 < t \le T,$$

с постоянным k, которое дополняется граничными и начальными условиями:

$$u(0,t) = g_0(t), \quad u(l,t) = g_1(t), \quad 0 < t \le T,$$

$$u(x,0) = v(x), \quad 0 < x < l.$$

Для численного решения этой краевой задачи будем использовать метод прямых. Дискретизацию по пространству проведем на равномерной сетке с h ($Nh = l$). Решение $y_i, i = 0,1,...,N$ в узлах $x_i, i = 0,1,...,N$ определяется из следующей задачи Коши для системы обыкновенных дифференциальных уравнений:

$$\frac{dy_i}{dt} = k\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + f(y_i,t), \quad i = 1,2,...,N-1, \quad 0 < t \le T,$$

[61] PyQt — http://www.riverbankcomputing.com/software/pyqt/intro
[62] FormLayout — http://code.google.com/p/formlayout/

$$y_0(t) = g_0(t), \quad y_N(t) = g_1(t), \quad 0 < t \le T,$$

$$y_i(0) = v(x_i), \qquad i = 1, 2, ..., N-1.$$

Правую часть зададим в виде

$$f(u,t) = su(1-u), \quad s = \text{const}.$$

В этом случае рассматриваемое нелинейное уравнение диффузии-реакции известно как уравнение Фишера (Колмогорова—Петровского—Пискунова). Пусть

$$g_0(t) = 1, \quad g_1(t) = 0, \quad v(x) = \begin{cases} 1, & 0 < x < l_0, \\ 0, & l_0 \le x < l. \end{cases}$$

Поставленная задача характеризуются параметрами k, s, l, l_0, N, T. Эти параметры (а также число временных слоев для вывода графика решения) будем вводить с использованием диалога, который реализуется модулем FormLayout (функция `fedit()`) на основе списка параметров.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
from formlayout import fedit
datalist = [(None, '<b>Physical</b>'),
            ('diffusion', 0.2),
            ('souce', 5.),
            (None, '<b>Geometric</b>'),
            ('lenth', 10.),
            ('l0', 1.),
            (None, '<b>Calculated</b>'),
            ('nodes', 51),
            ('tmax', 4.),
            ('ntime', 5) ]
d, s, l, l0, m, tmax, ntime = fedit(datalist, title="Parameters")
h = l / m
def f(x):
    return s*x*(1-x)
def ff(y, t):
    r = np.zeros((m+1), 'float')
    for i in range(1, m):
        r[i] = d*(y[i+1] - 2*y[i] + y[i-1]) / h**2 + f(y[i])
    return r
t = np.linspace(0, tmax,  ntime)
x = np.linspace(0, l,  m+1)
v = np.zeros((m+1), 'float')
for i in range(0, m+1):
```

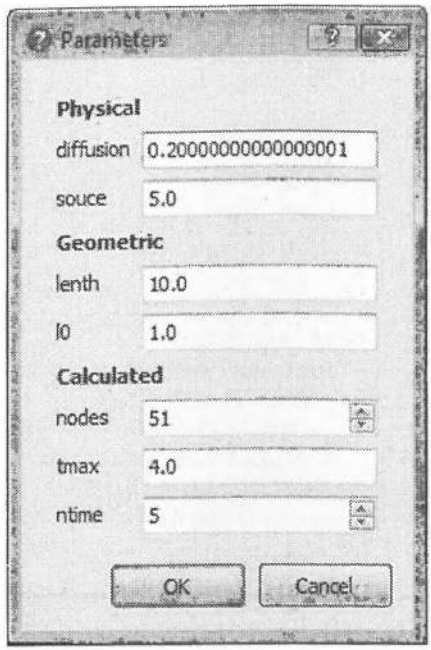

Рис. 3.58 Диалог ввода данных

```
    if x[i] < l0:
        v[i] = 1.
Y, infodict = integrate.odeint(ff, v, t, full_output=True)
sglist = ['-', '--', ':', '-.', '.']
for n in range(0, ntime):
    r = Y[n, :].T
    st = 'time =' + str(n*tmax/(ntime-1))
    plt.plot(x, r, sglist[n], label=st)
plt.legend(loc=0)
plt.xlabel('$x$')
plt.grid(True)
plt.show()
```

Помимо реализованного здесь (см. рис. 3.58) ввода числовых параметров (виджеты QDateEdit и QSpinBox из PyQt) и надписей (QLabel) обеспечивается интерактивный ввод текстовой информации, распознается ввод списков (реализация на основе виджета QComboBox), логических переменных — селекторная кнопка (QCheckBox), информация о цвете и шрифтах. Параметры также можно группировать на отдельных вкладках.

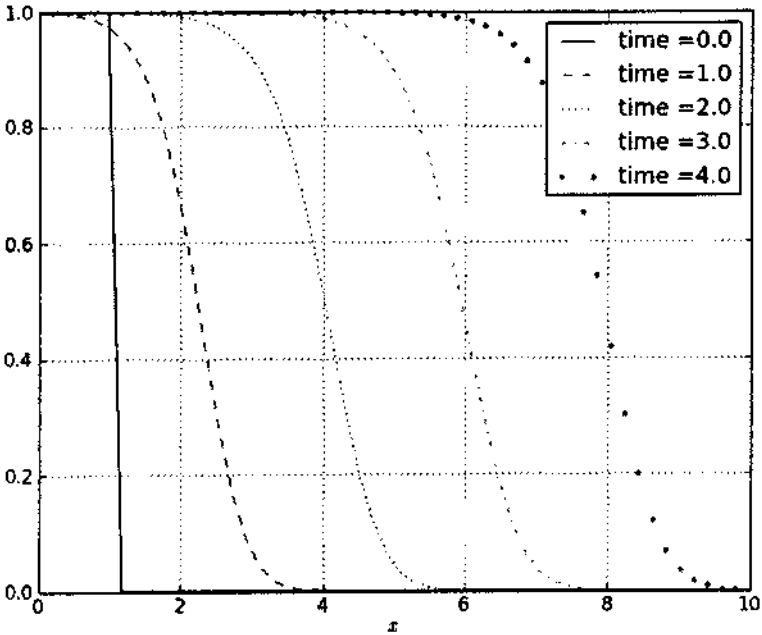

Рис. 3.59 Решение задачи для уравнения Фишера

На рис. 3.59 представлены графики решения задачи при параметрах по умолчанию. Само решение имеет вид бегущей волны со скоростью равной (Колмогоров, Петровский, Пискунов) $2\sqrt{ks} = 2$. Для решения задачи Коши используется модуль **integrate** пакета **SciPy**.

Прямые методы линейной алгебры

Одной из основных задач вычислительной математики является проблема решения систем линейных алгебраических уравнений с вещественными коэффициентами. Для нахождения приближенного решения систем уравнений используются прямые и итерационные методы. Математический аппарат линейной алгебры базируется на понятиях нормы вектора и матрицы, числа обусловленности. Рассматриваются классические методы исключения неизвестных, отмечаются особенности решения задач с симметричной вещественной матрицей.

Основные обозначения

$x = \{x_i\} = \{x_1, x_2, \ldots, x_n\}$	—	n-мерный вектор
$A = \{a_{ij}\}$	—	матрица с элементами a_{ij}
E	—	единичная матрица
$D = \mathrm{diag}\{d_1, d_2, \ldots, d_n\}$	—	диагональная матрица
$\|x\|$	—	норма вектора x
$\|A\|$	—	норма матрицы A
$\det(A)$	—	определитель матрицы A
$\mathrm{cond}(A)$	—	число обусловленности матрицы A

4.1 Задачи решения систем линейных уравнений

Рассматривается задача нахождения решения системы линейных алгебраических уравнений

$$Ax = f. \tag{4.1}$$

Здесь A — квадратная матрица $n \times n$ с вещественными коэффициентами a_{ij}, $i, j = 1, 2, \ldots, n$, f — заданный вектор с вещественными компонентами, x — искомый вектор. Будем считать, что определитель матрицы A отличен от нуля и поэтому система уравнений (4.1) имеет единственное решение. Для

сго нахождения будем использовать прямые (точные) методы, в которых решение находится за конечное число арифметических действий.

Входные данные (коэффициенты матрицы A и компоненты вектора f) заданы с погрешностью, т.е. вместо (4.1) решается система уравнений

$$\widetilde{A}\widetilde{x} = \widetilde{f}. \tag{4.2}$$

Необходимо оценить влияние погрешностей в задании коэффициентов и правой части на решение задачи. Близость решения задачи к решению задачи с точными входными данными связывается с числом обусловленности матрицы.

4.2 Алгоритмы решения систем линейных уравнений

Рассматриваются основные понятия линейной алгебры: норма вектора, согласованная норма матрицы. Дается оценка погрешности решения системы линейных уравнений при возмущении правой части и коэффициентов матрицы на основе привлечения понятия числа обусловленности. Среди прямых методов выделяется метод Гаусса с и без выбора главного элемента, который связан с разложением матрицы на произведение треугольных матриц. Для задач с симметричными вещественными матрицами выделяется метод квадратного корня (метод Холецкого).

Обусловленность матрицы и оценки точности

Среди норм векторов наиболее употребительны нормы:

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|,$$

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2},$$

$$\|x\|_\infty = \max_{1 \le i \le n} |x_i|.$$

Матричная норма $\|A\|$ подчинена векторной норме $\|x\|$, если

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}.$$

Для квадратной невырожденной матрицы A существует единственная матрица A^{-1}, называемая обратной, для которой $AA^{-1} = A^{-1}A = E$. Число обусловленности матрицы A есть

$$\mathrm{cond}(A) = \|A\|\|A^{-1}\|.$$

При рассмотрении близости решений уравнений (4.1), (4.2) для погрешности в задании матрицы, решения и правой части используем обозначения

$$\delta A = \tilde{A} - A, \quad \delta x = \tilde{x} - x, \quad \delta f = \tilde{f} - f.$$

Если матрица A имеет обратную и выполнено условие

$$\|\delta A\|\|A\| < 1,$$

тогда для относительной погрешности справедлива оценка

$$\frac{\|\delta x\|}{\|x\|} \le \frac{\mathrm{cond}(A)}{1 - \mathrm{cond}(A)\|\delta A\|\|A\|^{-1}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta f\|}{\|f\|} \right). \tag{4.3}$$

Оценка (4.3) выражает устойчивость решения при возмущении правой части и коэффициентов уравнения (4.1) (корректность задачи).

Метод Гаусса для решения систем линейных уравнений

Классический алгоритм исключения неизвестных (метод Гаусса) связывается с использованием представления исходной матрицы A в виде произведения треугольных матриц. Матрица A называется нижней (левой) треугольной матрицей, если ее элементы $a_{ij} = 0$ при $i < j$, для верхней (правой) треугольной матрицы A — $a_{ij} = 0$, если $i > j$.

Если все главные миноры матрицы A отличны от нуля, т.е.

$$a_{11} \neq 0, \quad \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \neq 0, \quad \det|A| \neq 0,$$

тогда матрица A представима в виде

$$A = LU, \tag{4.4}$$

где L — нижняя треугольная матрица с единичной диагональю и U — верхняя треугольная матрица с ненулевыми диагональными элементами.

Приведем рекуррентные формулы для определения треугольных матриц L и U:

$$u_{11} = a_{11},$$

$$u_{1j}, \quad l_{j1} = \frac{a_{j1}}{u_{11}}, \quad j = 2, 3, \ldots, n,$$

$$u_{ii} = a_{ii} - \sum_{k=1}^{i-1} l_{ik} u_{ki}, \quad i = 2, 3, \ldots, n,$$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, \quad l_{ji} = \frac{1}{u_{ii}} \left(a_{ji} - \sum_{k=1}^{j-1} l_{jk} u_{ki} \right),$$

$$i = 2, 3, \ldots, n, \quad j = i+1, i+2, \ldots, n.$$

Эти формулы определяют компактную схему метода Гаусса для разложения матрицы на множители.

После того как разложение (4.4) проведено решение системы уравнений сводится к последовательному решению двух систем уравнений с треугольными матрицами:

$$Ly = f, \tag{4.5}$$

$$Ux = y. \tag{4.6}$$

Разложение (4.4) и решение системы (4.5) связывается с прямым ходом в методе исключения неизвестных, а решение системы (4.6) — с обратным ходом.

В методе Гаусса с выбором главного элемента на очередном шаге исключается неизвестное, коэффициент по модулю при котором является наибольшим. В этом случае метод Гаусса применим для любых невырожденных матриц A, т.е. матриц, для которых $\det(A) \neq 0$).

Матрицей перестановок P называется квадратная матрица, у которой в каждой строке и в каждом столбце только один элемент отличен от нуля и равен единице. При $\det(A) \neq 0$ существует матрица перестановок P такая, что справедливо разложение

$$PA = LU.$$

Тем самым метод Гаусса с выбором главного элемента соответствует применению обычного метода Гаусса, который применяется к системе, полученной из исходной системы перестановкой некоторых уравнений.

Метод квадратного корня

При решении системы уравнений (4.1) с симметричной вещественной невырожденной матрицей A используется разложение

$$A = S^* D S,$$

где S – верхняя треугольная матрица с положительными элементами на главной диагонали, S^* – транспонированная к ней ($s^*_{ij} = s_{ji}$), а D — диагональная матрица с элементами d_i, $i = 1, 2, \ldots, n$, равными ± 1. Вычисления на основе этого разложения определяют метод квадратного корня (метод Холецкого).

Для элементов матриц S и D используются расчетные формулы:

$$d_1 = \operatorname{sign} a_{11}, \quad s_{11} = |a_{11}|^{1/2}, \quad s_{1j} = a_{1j}/s_{11}, \quad j = 2, 3, \ldots, n,$$

$$d_i = \operatorname{sign}\left(a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2 d_k\right),$$

$$s_{ii} = \left| a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2 d_k \right|^{1/2},$$

$$s_{ij} = \frac{1}{s_{ii} d_i} \left(a_{ij} - \sum_{k=1}^{i-1} s_{ki} s_{kj} d_k \right),$$

$$i = 2, 3, \ldots, n, \quad j = i+1, i+2, \ldots, n.$$

В методе квадратного корня вычислительные затраты примерно в два раза меньше, чем в стандартном методе Гаусса (эффект учета симметрии матрицы задачи).

4.3 Упражнения

Ниже приведены примеры построения программ на языке Python для численного решения систем линейных уравнений прямыми методами линейной алгебры. Мы ограничиваемся основными возможностями базового математического пакета NumPy (поддержка работы с многомерными массивами).

Упражнение 4.1 *Напишите программу, реализующую решение системы линейных алгебраических уравнений на основе LU-разложения. С ее помощью найдите решение системы*

$$Ax = f$$

с матрицей

$$a_{ij} = \begin{cases} 1, & i = j, \\ -1, & i < j, \\ 0, & i > j \neq n, \\ 1, & j = n, \end{cases} \qquad i = 1, 2, ..., n, \quad j = 1, 2, ..., n$$

и правой частью

$$f_i = 1, \quad i = 1, 2, ..., n$$

при $n = 10$.

В модуле lu функция `decLU()` проводит LU-разложение входной матрицы A и записывает результат в матрицу LU. Реализация (4.5), (4.6) для решения системы уравнений проводится функцией `solveLU()`.

Модуль lu

```
import numpy as np
def decLU(A):
    """
    Returns the decompositon LU for matrix A.
    """
```

```
    n = len(A)
    LU = np.copy(A)
    for j in range(0,n-1):
        for i in range(j+1,n):
            if LU[i,j] != 0.:
                u = LU[i,j] / LU[j,j]
                LU[i,j+1:n] = LU[i,j+1:n] - u*LU[j,j+1:n]
                LU[i,j] = u
    return LU
def solveLU(A, f):
    """
    Solve the linear system Ax = b.
    """
    n = len(A)
    # LU decomposition
    LU = decLU(A)
    x = np.copy(f)
    # forward substitution process
    for i in range(1,n):
        x[i] = x[i] - np.dot(LU[i,0:i], x[0:i])
    # back substitution process
    for i in range(n-1,-1,-1):
        x[i] = (x[i] - np.dot(LU[i,i+1:n], x[i+1:n])) / LU[i,i]
    return x
```

Решение нашей конкретной системы линейных алгебраических уравнений дается следующей программой.

exer_4_1.py

```
import numpy as np
from lu import decLU, solveLU
n = 8
A = -np.ones((n, n), 'float')
for i in range(0,n):
    A[i,i] = 1.
    A[i,n-1] = 1.
    if i < n-1:
        A[i,i+1:n-1] = 0.
print 'A:\n', A
LU = decLU(A)
print 'LU:\n', LU
f = np.ones((n), 'float')
print 'b:\n', f
x = solveLU(A, f)
print 'x:\n', x
```

```
A
[[ 1  0  0  0  0  0  0  1 ]
 [-1  1. 0. 0. 0. 0  0  1. ]
 [-1 -1  1  0  0  0  0  1 ]
 [-1 -1. -1. 1  0  0  0  1 ]
 [-1 -1  -1 -1  1  0  0  1 ]
 [-1 -1  -1. -1. -1. 1. 0. 1. ]
 [-1 -1. -1  -1  -1. -1. 1  1. ]
 [-1 -1  -1. -1  -1  -1  -1  1 ]]
LU.
[[  1    0    0.   0.   0.   0.   0    1  ]
 [ -1.   1.   0.   0.   0.   0.   0.   2. ]
 [ -1   -1    1.   0    0.   0    0    4. ]
 [ -1.  -1   -1.   1    0    0    0    8  ]
 [ -1   -1   -1   -1   1    0    0   16  ]
 [ -1   -1   -1   -1  -1    1.   0.  32  ]
 [ -1   -1   -1.  -1  -1   -1    1   64  ]
 [ -1   -1.  -1   -1  -1.  -1   -1  128 ]]
b
[ 1   1   1   1.  1   1.  1   1 ]
x.
[ 0   0   0.  0   0.  0   0   1 ]
```

Рассматриваемая модельная задача имеет точное решение. В частности, треугольная матрица U есть

$$u_{ij} = \begin{cases} 1, & i = j, \\ 2^j, & j = n, \\ 0, & i \neq j,\ j \neq n, \end{cases} \qquad i = 1, 2, ..., n, \quad j = 1, 2, ..., n.$$

в LU-разложении матрицы A.

Упражнение 4.2 *Напишите программу, реализующую разложение Холецкого $A = LL^*$ для симметричной положительно определенной матрицы A и вычисляющей определитель матрицы на основе этого разложения. Найдите разложение Холецкого и определитель матрицы Гильберта, для которой*

$$a_{ij} = \frac{1}{i + j - 1}, \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., n$$

при $n = 4$.

Функция `decChol()` в модуле chol, реализует разложение Холецкого (вычисляется нижняя треугольная матрица L), функция `detChol()` вычисляет определитель матрицы A.

Модуль `chol`

```
import numpy as np
```

```
def decChol(A):
    """
    Returns the decompositon Choleski for matrix A.
    """
    n = len(A)
    L = np.copy(A)
    for j in range(n):
        try:
            L[j,j] = np.sqrt(L[j,j] - np.dot(L[j,0:j], L[j,0:j]))
        except ValueError:
            print 'Matrix is not positive definite'
        for i in range(j+1,n):
            L[i,j] = (L[i,j] - np.dot(L[i,0:j], L[j,0:j])) / L[j,j]
    for j in range(1,n):
        L[0:j,j] = 0.
    return L
def detChol(A):
    """
    Returns the determinant of the matrix A.
    """
    n = len(A)
    L = decChol(A)
    det = 1.
    for i in range(n):
        det *= L[i,i]**2
    return det
```

Разложение матрицы Гильберта и вычисление ее определителя проводится в следующей программе.

exer_4_2.py

```
import numpy as np
from chol import decChol, detChol
n = 4
A = np.zeros((n, n), 'float')
for i in range(0,n):
    for j in range(0,n):
        A[i,j] = 1./(i+j+1)
print 'A:\n', A
L = decChol(A)
print 'L:\n', L
det = detChol(A)
print 'det = ', det
```

```
[[ 0.5          0.33333333  0.25        0.2         ]
 [ 0.33333333   0.25        0.2         0.16666667]
 [ 0.25         0.2         0.16666667  0.14285714]]
L
[[ 1           0           0           0           ]
 [ 0.5         0.28867513  0           0           ]
 [ 0.33333333  0.28867513  0.0745356   0           ]
 [ 0.25        0.25980762  0.1118034   0.01889822]]
det =  1.65343915344e-07
```

Определитель матрицы Гильберта при возрастании n стремиться к нулю.

Упражнение 4.3 *Напишите программу, которая решает систему линейных уравнений для трехдиагональной ($a_{ij} = 0$ при $|i-j| > 1$) матрицы $n \times n$ на основе LU-разложения. Найдите решение уравнения с*

$$a_{ii} = 2, \quad a_{i,i-1} = a_{i,i+1} = -1$$

при постоянной правой части $f_i = 2h^2$, $h = (n+1)^{-1}$, $i = 1, 2, ..., n$ и сравните его с точным решением $y_i = ih(1-ih)$, $i = 1, 2, ..., n$. Рассчитайте определитель матрицы и сравните его значение с точным $(n+1)$.

Трехдиагональная матрица A задается тремя диагоналями:

$$a_i = a_{ii}, \quad b_i = a_{i,i+1}, \quad c_i = a_{i,i-1}.$$

В модуле lu3 функция `decLU3()` предназначена для LU-разложение трехдиагональной матрицы A. Результат записан в виде трех диагоналей, причем

$$d_i = l_{ii}, \quad u_i = u_{i,i+1}, \quad l_i = l_{i,i-1}.$$

Для решения системы уравнений используется функция `solveLU3()`.

Модуль lu3

```
import numpy as np
def decLU3(a, b, c):
    """
    Input of tridiagonal matrix A:
        a[i] = A[i,i],
        b[i] = A[i,i+1],
        c[i] = A[i,i-1]
    Returns the decompositon LU for tridiagonal matrix:
        d[i] = L[i,i]
        u[i] = U[i,i+1]
        l[i] = L[i,i-1]
    """
    n = len(a)
    d = np.copy(a)
    u = np.copy(b)
```

```
    l = np.copy(c)
    for i in range(1, n):
        al = l[i] / d[i-1]
        d[i] = d[i] - al*u[i-1]
        l[i] = al
    return d, u, l
def solveLU3(a, b, c, f):
    """
    Solve the linear system Ax = b with tridiagonal matrix:
        a[i] = A[i,i],
        b[i] = A[i,i+1],
        c[i] = A[i,i-1]
    """
    n = len(a)
    # LU decomposition
    d, u, l = decLU3(a, b, c)
    x = np.copy(f)
    # forward substitution process
    for i in range(1, n):
        x[i] = x[i] - l[i]*x[i-1]
    # back substitution process
    x[n-1] = x[n-1] / d[n-1]
    for i in range(n-2,-1,-1):
        x[i] = (x[i] - u[i]*x[i+1]) / d[i]
    return x
```

Решение системы уравнений, вычисление определителя и сравнение с точными данными проводится следующей программой.

exer_4_3.py

```
import numpy as np
from lu3 import solveLU3, decLU3
n = 50
h = 1. / (n + 1)
a = 2*np.ones((n), 'float')
b = - np.ones((n), 'float')
c = - np.ones((n), 'float')
f = h**2*2*np.ones((n), 'float')
# solution
x = solveLU3(a, b, c, f)
# test function
y = np.zeros((n), 'float')
# LU decomposition
d, u, l = decLU3(a, b, c)
er = 0.
```

```
det = 1.
for i in range(n):
    y[i] = (i+1)*h*(1-(i+1)*h)
    er = er + (y[i]-x[i])**2*h
    det = det*d[i]
er = np.sqrt(er)
print 'error =', er
print 'determinant calculated =', det, '\nexact =', n+1
```

```
error = 1.29192707206e-15
determinant calculated = 51.0
exact = 51
```

Аналогичные данные мы получим и при других значениях n.

4.4 Задачи

Задачи представляют собой задание для самостоятельного создания программ на языке Python. Как и в упражнениях, предусматривается написание модуля для решения поставленной задачи и программы, которая иллюстрирует работу этого модуля.

Задача 4.1 *Напишите программу для вычисления норм вещественных векторов* ($\|x\|_p$, $p = 1, 2, \infty$) *и норм вещественных матриц*

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|,$$

$$\|A\|_E = \left(\sum_{i,j=1}^{n} |a_{ij}|^2 \right)^{1/2},$$

$$\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|.$$

На основе экспериментов со случайными матрицами при различных n убедитесь в выполнении неравенств

$$\frac{1}{\sqrt{n}}\|A\|_\alpha \le \|A\|_E \le \sqrt{n}\|A\|_\alpha, \quad \alpha = 1, \infty$$

для евклидовой (сферической) нормы (нормы Фробениуса) $\|A\|_E$.

Задача 4.2 *Напишите программу, реализующую LU-разложение с выбором главного элемента по строке (схема частичного выбора) с выводом матрицы LU и вектора перенумерации переменных (перестановки столбцов),*

который определяет матрицу перестановок P. С ее помощью найдите LU-разложение матрицы Паскаля, для которой

$$a_{ij} = \frac{(i+j-2)!}{(i-1)!(j-1)!} \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., n.$$

Проверьте выполнение $\det(A) = 1$ и зависимость определителя от n. Повторите эксперименты с LU-разложением без выбора главного элемента (модуль lu).

Задача 4.3 *Разработайте алгоритм и реализуйте его программно для варианта декомпозиции Холецкого симметричной невырожденной матрицы в виде $A = LDL^*$, где L — нижняя треугольная матрица с единичной диагональю и D — диагональная матрица, без вычисления квадратного корня. С помощью этой программы найдите декомпозицию Холецкого KMS (Kac-Murdock-Szego) матрицы, для которой*

$$a_{ij} = \varrho^{|i-j|}, \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., n.$$

При различных n и ϱ сравните рассчитанную матрицу D с точным решением, для которого

$$d_{ii} = \begin{cases} 1, & i = 1, \\ 1 - \varrho^2, & i = 2, 3, ..., n. \end{cases}$$

Задача 4.4 *Рассмотрите алгоритм построения обратной матрицы на основе решения матричного уравнения*

$$AX = E,$$

где E — единичная, а X — искомая квадратная матрица. Напишите программу вычисления обратной матрицы на основе LU-разложение с выбором главного элемента по строке (схема частичного выбора) (задача 4.2). Найдите обратную к матрице Лемера (Lehmer), для которой

$$a_{ij} = \frac{\min(i,j)}{\max(i,j)}, \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., n.$$

При вычислениях с различными значениями n убедитесь в том, что A^{-1} является трехдиагональной матрицей.

Задача 4.5 *Напишите программу для вычисления числа обусловленности квадратной матрицы с использованием норм $\|\cdot\|_\alpha$, $\alpha = 1, E, \infty$ (см. задачу 4.1) и вычисления обратной матрицы на основе LU-разложение матрицы (упражнение 4.1). Найдите при различных n число обусловленности матрицы A, в которой*

$$a_{ij} = \min(i,j), \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., n.$$

Убедитесь, что обратная матрица является трехдиагональной, внедиагональные элементы которой $a^{-1}_{i,i\pm1} = -1$, *а диагональные*

$$a^{-1}_{i,i} = \begin{cases} 2, & i = 1,2,...,n-1, \\ 1, & i = n. \end{cases}$$

Задача 4.6 *Рассмотрите алгоритм решения системы уравнений с трехдиагональной циклической матрицей, элементы которой* $a_{ij} = 0$ *при* $1 < |i-j| < n-1$ ($a_{1n} \neq 0$, $a_{n1} \neq 0$), *на основе LU-разложения. Покажите, что в этом случае*

$$l_{ij} \neq 0, \quad i = 1,2,...,n-1, \; j = i, i-1, \quad i = n, \; j = 1,2,...,n,$$

$$u_{ij} \neq 0, \quad j = 1,2,...,n-1 \; i = j, j-1, \quad j = n, \; i = 1,2,...,n,$$

и напишите программу решения задачи $Ax = f$. *Найдите решение уравнения с циклической трехдиагональной матрицей, для которой*

$$a_{ii} = 2 + h^2, \quad a_{i,i-1} = a_{i,i+1} = -1, \quad a_{1n} = a_{n1} = -1$$

при правой части

$$f_i = \left(1 + \frac{4}{h^2}\sin^2(\pi h)\right)\sin(2\pi(i-1)h), \quad i = 1,2,...,n$$

где $h = n^{-1}$. *Покажите, что*

$$y_i = \sin(2\pi(i-1)h), \quad i = 1,2,...,n$$

есть точное решение рассматриваемой задачи.

Итерационные методы линейной алгебры

Для приближенного решения больших систем линейных алгебраических уравнений используются итерационные методы. Такие системы возникают при приближенном решении многомерных краевых задач математической физики. Рассмотрение начинается с классических итерационных методов Якоби и Зейделя. Приведены базовые понятия теории итерационных методов решения систем линейных уравнений, рассматриваемых в евклидовых пространствах. Обсуждаются проблемы выбора итерационных параметров, матрицы перехода (переобуславливателя).

Основные обозначения

$x = \{x_i\} = \{x_1, x_2, \ldots, x_n\}$	—	n-мерный вектор
$A = \{a_{ij}\}$	—	матрица с элементами a_{ij}
E	—	единичная матрица
$D = \operatorname{diag}\{d_1, d_2, \ldots, d_n\}$	—	диагональная матрица
$\|x\|$	—	норма вектора x
$\|A\|$	—	норма матрицы A
x^k	—	приближенное решение на k-й итерации
$z^k = x^k - x$	—	погрешность приближенного решения
$r^k = Ax^k - f$	—	невязка на k-й итерации
τ, τ_k	—	итерационные параметры

5.1 Итерационное решение систем линейных уравнений

Рассматриваются проблемы итерационного решения системы линейных уравнений

$$Ax = f \tag{5.1}$$

для нахождения вектора x. В теории итерационных методов матрица A обычно рассматривается как линейный оператор, действующий на евклидовом

пространстве $H = l_2$, в котором скалярное произведение есть $(x, y) = \sum_{i=1}^{n} x_i y_i$, а норма $\|x\| = (x, x)^{1/2}$.

Итерационный метод основан на том, что начиная с некоторого начального приближения $x^0 \in H$ последовательно определяются приближенные решения уравнения (5.1) $x^1, x^2, \ldots, x^k, \ldots$, где k — номер итерации. Значения x^{k+1} определяются по ранее найденным x^k, $x^{k-1}, \ldots$. Если при вычислении x^{k+1} используются только значения на предыдущей итерации x^k, то итерационный метод называется одношаговым (двухслойным). Соответственно, при использовании x^k и x^{k-1} итерационный метод называется двухшаговым (трехслойным).

Двухслойный итерационный метод записывается в следующей канонической форме

$$B\frac{x^{k+1} - x^k}{\tau_{k+1}} + Ax^k = f, \quad k = 0, 1, \ldots. \tag{5.2}$$

Для характеристики точности приближенного решения естественно ввести погрешность $z^k = x^k - x$. Будем рассматривать сходимость итерационного метода в энергетическом пространстве H_R, порожденном симметричной и положительно определенной матрицей R. В H_R скалярное произведение и норма есть

$$(y, w)_R = (Ry, w), \quad \|y\|_R = ((y, y)_R)^{1/2}.$$

Итерационный метод сходится в H_R, если $\|z^k\|_R \to 0$ при $k \to \infty$. В качестве меры сходимости итераций принимают относительную погрешность ε, так что на K-й итерации

$$\|x^K - x\|_R \le \varepsilon \|x^0 - x\|_R. \tag{5.3}$$

В силу того, что само точное решение x неизвестно, оценка точности приближенного решения проводится по невязке

$$r^k = Ax^k - f = Ax^k - Ax,$$

которая может быть вычислена непосредственно. Например, итерационный процесс проводится до выполнения оценки

$$\|r^K\| \le \varepsilon \|r^0\|. \tag{5.4}$$

Использование критерия сходимости (5.4) соответствует выбору $R = A^*A$ в (5.3). Минимальное число итераций, которое гарантирует точность ε (выполнение (5.3) или (5.4)), обозначим $K(\varepsilon)$.

При построении итерационного метода мы должны стремиться к минимизации вычислительной работы по нахождению приближенного решения задачи (5.1) с заданной точностью. Пусть Q_k — число арифметических действий для

нахождения приближения x^k и пусть делается $K \geq K(\varepsilon)$ итераций. Тогда общие затраты оцениваются величиной

$$Q(\varepsilon) = \sum_{k=1}^{K} Q_k.$$

Применительно к двухслойному итерационному методу (5.2) минимизация $Q(\varepsilon)$ может достигаться за счет выбора операторов B_k и итерационных параметров τ_{k+1}. Обычно матрицы B_k (перообуславливатели) задаются из каких-либо соображений близости к матрице A, а оптимизация итерационного метода (5.2) осуществляется за счет выбора итерационных параметров.

5.2 Итерационные алгоритмы решения систем линейных уравнений

Рассматриваются традиционные итерационные методы решения систем линейных уравнений — метод Якоби и метод Зейделя. Приведены основные результаты о скорости сходимости итерационных методов при решении задач с вещественной симметричной положительно определенной матрицей. Приводится оптимальный выбор постоянных и переменных итерационных параметров. Второй класс итерационных методов связан с определением итерационных параметров на каждом итерационном шаге из минимума функционалов для невязки — итерационные методы вариационного типа.

Классические итерационные методы

В итерационном методе Якоби новое приближение на $k+1$-й итерации определяется из условий

$$\sum_{j=1}^{i-1} a_{ij} x_j^k + a_{ii} x_i^{k+1} + \sum_{j=i+1}^{n} a_{ij} x_j^k = f, \quad i = 1, 2, \ldots, n. \tag{5.5}$$

Тем самым следующее приближение для отдельной компоненты вектора определяется из соответствующего уравнения системы, когда все другие компоненты берутся с предыдущей итерации.

Метод Зейделя основан на том, что найденное приближение для компонент вектора сразу же задействуются в вычислениях:

$$\sum_{j=1}^{i-1} a_{ij} x_j^{k+1} + a_{ii} x_i^{k+1} + \sum_{j=i+1}^{n} a_{ij} x_j^k = f, \quad i = 1, 2, \ldots, n. \tag{5.6}$$

Для записи итерационных методов (5.5), (5.6) используется следующее разложение матрицы A:

$$A = L + D + U. \tag{5.7}$$

Здесь $D = \operatorname{diag}\{a_{11}, a_{22}, \ldots, a_{nn}\}$ — диагональная часть матрицы A, а L и U — нижняя и верхняя треугольные матрицы с нулевыми элементами на главной диагонали, т.е.

$$L = \begin{bmatrix} 0 & 0 & 0 & \ldots & 0 \\ a_{21} & 0 & 0 & \ldots & 0 \\ a_{31} & a_{32} & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \ldots & 0 \end{bmatrix},$$

$$U = \begin{bmatrix} 0 & a_{12} & a_{13} & \ldots & a_{1n} \\ 0 & 0 & a_{23} & \ldots & a_{2n} \\ 0 & 0 & 0 & \ldots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 0 \end{bmatrix}.$$

С учетом (5.7) итерационный метод Якоби (5.5) записывается в каноническом виде (5.2) при

$$B = D, \quad \tau_{k+1} = 1.$$

Для итерационного метода Зейделя (5.6) имеем

$$B = D + L, \quad \tau = 1.$$

Наиболее естественным обобщением рассматриваемых итерационных методов является использование переменных итерационных параметров. В этом случае мы получим

$$D\frac{x^{k+1} - x^k}{\tau_{k+1}} + Ax^k = f, \quad k = 0, 1, \ldots, \tag{5.8}$$

$$(D + L)\frac{x^{k+1} - x^k}{\tau_{k+1}} + Ax^k = f, \quad k = 0, 1, \ldots. \tag{5.9}$$

Отметим также метод верхней релаксации

$$(D + \tau L)\frac{x^{k+1} - x^k}{\tau} + Ax^k = f, \quad k = 0, 1, \ldots. \tag{5.10}$$

который можно рассматривать как параметрическое обобщение итерационного метода Зейделя.

Запишем стационарный итерационный метод ($B_k = B$, $\tau_{k+1} = \tau$ в виде

$$x^{k+1} = Sx^k + B^{-1}f, \quad k = 0, 1, \ldots, \tag{5.11}$$

где $S = E - \tau B^{-1}A$ — матрица перехода. Необходимым и достаточным условием сходимости итерационного метода (5.11) является условие, чтобы спектральный радиус матрицы перехода S меньше единицы, т.е. когда все собственные значения матрицы S по модулю меньше единицы.

Двухслойные итерационные методы

Приведем некоторые факты теории итерационных методов при решении задачи (5.1) с симметричной вещественной положительно определенной матрицей A, т.е. когда

$$A = A^* > 0. \tag{5.12}$$

Метод простой итерации (стационарный итерационный метод) соответствует использованию в (5.2) постоянного итерационного параметра $\tau_{k+1} = \tau$, т.е.

$$B\frac{x^{k+1} - x^k}{\tau} + Ax^k = f, \quad k = 0, 1, \ldots. \tag{5.13}$$

Итерационный метод (5.13) для решения задачи (5.1), (5.12) сходится в H_A, т.е. $\|z\|_A \to 0$ при $k \to \infty$, если выполнено неравенство

$$B > \frac{\tau}{2}A. \tag{5.14}$$

Будем считать, что при

$$B = B^* > 0 \tag{5.15}$$

и задана априорная информация об операторах B и A в виде двухстороннего операторного неравенства

$$\gamma_1 B \le A \le \gamma_2 B, \quad \gamma_1 > 0, \tag{5.16}$$

т.е. операторы B и A энергетически эквивалентны с постоянными энергетической эквивалентности γ_α, $\alpha = 1, 2$. Тогда итерационный метод (5.13) сходится в H_R, $R = A, B$ при $0 < \tau < 2/\gamma_2$. Оптимальным значением итерационного параметра является

$$\tau = \tau_0 = \frac{2}{\gamma_1 + \gamma_2}, \tag{5.17}$$

при котором для числа итераций K, необходимых для достижения точности ε, справедлива оценка

$$K \ge K_0(\varepsilon) = \frac{\ln \varepsilon}{\ln \varrho_0}, \tag{5.18}$$

где

$$\varrho_0 = \frac{1 - \xi}{1 + \xi}, \quad \xi = \frac{\gamma_1}{\gamma_2}.$$

Заметим, что в (5.18) $K_0(\varepsilon)$, вообще говоря, нецелое и K -- минимальное целое, при котором выполнено $K \ge K_0(\varepsilon)$. Этот результат указывает путь оптимизации сходимости итерационного процесса (5.13) за счет выбора оператора B в соответствии с (5.16), т.е. оператор B должен быть близок оператору A по энергии.

Оптимальный набор итерационных параметров в нестационарном итерационном методе (5.2) для приближенного решения задачи (5.1) при (5.12), (5.15) связан с корнями полиномов Чебышева, поэтому такой итерационный метод называется чебышевским итерационным методом (методом Ричардсона). Определим множество $\mathcal{M}_K$ следующим образом:

$$\mathcal{M}_K = \left\{ -\cos\left(\frac{2i-1}{2K}\pi\right), \quad i = 1, 2, \ldots, K \right\}.$$

Для итерационных параметров τ_k используется формула

$$\tau_k = \frac{\tau_0}{1 + \varrho_0 \mu_k}, \quad \mu_k \in \mathcal{M}_K, \quad k = 1, 2, \ldots, K. \tag{5.19}$$

Чебышевский итерационный метод (5.2), (5.19) сходится в H_R, $R = A, B$ и для числа итераций K, необходимых для достижения точности ε, справедлива оценка

$$K \geq K_0(\varepsilon) = \frac{\ln(2\varepsilon^{-1})}{\ln \varrho_1^{-1}}, \tag{5.20}$$

где

$$\varrho_1 = \frac{1 - \xi^{1/2}}{1 + \xi^{1/2}}, \quad \xi = \frac{\gamma_1}{\gamma_2}.$$

Заметим, что в чебышевском методе (см., (5.19)) расчет итерационных параметров осуществляется по заданному общему числу итераций K. Естественно, что вырожденный случай $K = 1$ соответствует рассмотренному выше методу простой итерации. Практическая реализация чебышевского итерационного метода связана с проблемой вычислительной устойчивости, которая решается специальным упорядочиванием итерационных параметров (выбором μ_k из множества $\mathcal{M}_K$).

Итерационные методы вариационного типа

Выше рассматривались итерационные методы решения задачи в условиях, когда задана априорная информация об операторах B и A в виде констант (см. (5.16)) энергетической эквивалентности γ_1 и γ_2. Через эти постоянные определяются оптимальные значения итерационных параметров (см. (5.17), (5.19)). В итерационных методах вариационного типа, в которых итерационные параметры вычисляются без такой априорной информации.

Обозначая невязку $r^k = Ax^k - f$ и поправку $w^k = B^{-1}r^k$, для итерационных параметров при естественном предположении о минимизации погрешности в H_R получим формулу

$$\tau_{k+1} = \frac{(Rw^k, z^k)}{(Rw^k, w^k)}. \tag{5.21}$$

Итерационный процесс (5.2) запишется следующим образом

$$x^{k+1} = x^k - \tau_{k+1} w^k, \quad k = 0, 1, \ldots .$$

Конкретизация итерационного метода достигается за счет выбора оператора $R = R^* > 0$. Этот выбор должен быть подчинен, в частности, условию возможности вычисления итерационных параметров. В формулу (5.21) входит невычисляемая величина z^k и поэтому простейший выбор $R = B$ здесь не проходит. Вторая отмеченная выше возможность $R = A$ приводит нас к итерационному методу скорейшего спуска, когда

$$\tau_{k+1} = \frac{(w^k, r^k)}{(Aw^k, w^k)}. \tag{5.22}$$

Среди других возможностей выбора R отметим случай $R = AB^{-1}A$ — метод минимальных поправок, когда

$$\tau_{k+1} = \frac{(Aw^k, w^k)}{(B^{-1}Aw^k, Aw^k)}. \tag{5.23}$$

Двухслойный итерационный метод вариационного типа сходится не медленнее метода простой итерации, т.е. для числа итераций n, необходимых для достижения точности ε, справедлива оценка (5.18).

В вычислительной практике наибольшее распространение получили трехслойные итерационные методы вариационного типа. По скорости сходимости они не хуже итерационного метода с чебышевским набором итерационных параметров.

В трехслойном (двухшаговом) итерационном методе новое приближение находится по двум предыдущим. Для реализации метода требуются два начальных приближения x^0, x^1. Обычно x^0 задается произвольно, а x^1 находится по двухслойному итерационному методу. Трехслойный метод записывается в следующей канонической форме трехслойного итерационного метода:

$$\begin{gathered} Bx^{k+1} = \alpha_{k+1}(B - \tau_{k+1}A)x^k + (1 - \alpha_{k+1})Bx^{k-1} + \alpha_{k+1}\tau_{k+1}\varphi, \\ k = 1, 2, \ldots, \\ Bx^1 = (B - \tau_1 A)x^0 + \tau_1 \varphi, \end{gathered} \tag{5.24}$$

где α_{k+1} и τ_{k+1} — итерационные параметры.

В методе сопряженных градиентов итерационные параметры рассчитываются по формулам

$$\tau_{k+1} = \frac{(w^k, r^k)}{(Aw^k, w^k)}, \quad k = 0, 1, \ldots,$$

$$\alpha_{k+1} = \left(1 - \frac{\tau_{k+1}}{\tau_k} \frac{(w^k, r^k)}{(w^{k-1}, r^{k-1})} \frac{1}{\alpha_k}\right)^{-1}$$

$$k = 1, 2 \ldots, \quad \alpha_1 = 1.$$

Этот метод наиболее широко используется в вычислительной практике при решении задач с симметричной положительно определенной матрицей.

5.3 Упражнения

Упражнение 5.1 *Напишите программу, реализующую приближенное решение системы линейных алгебраических уравнений итерационным методом Зейделя. С ее помощью найдите решение системы уравнений с трехдиагональной матрицей*

$$Ax = f,$$

в которой

$$a_{ii} = 2, \quad a_{i,i+1} = -1 - \alpha, \quad a_{i,i-1} = -1 + \alpha,$$

а правая часть

$$f_0 = 1 - \alpha, \quad f_i = 0, \quad i = 2, 3, ..., n-1, \quad f_n = 1 + \alpha,$$

определяет точное решение $x_i = 1, \quad i = 1, 2, ..., n$. *Исследуйте зависимость числа итераций от* n *и параметра* α *при* $0 \leq \alpha \leq 1$.

В модуле seidel функция `seidel()` обеспечивает итерационное решение системы линейных уравнений методом Зейделя, результатом выступает приближенное решение и число итераций для достижения необходимой точности (нормы невязки).

Модуль `seidel`

```
import numpy as np
import math as mt
def seidel(A, f, tol = 1.0e-9):
    """
    Solve the linear system Ax = b by Seidel method
    """
    n = len(f)
    x = np.zeros((n), 'float')
    r = np.copy(f)
    # the maximum number of iterations is 10000
    for k in range(1,10001):
        for i in range(n):
            x[i] = x[i] + (f[i] - np.dot(A[i,0:n], x[0:n])) / A[i,i]
        for i in range(n):
            r[i] = f[i] - np.dot(A[i,0:n], x[0:n])
        if np.dot(r,r) < tol**2: return x, k
    print 'Seidel method failed to converge:'
```

```
    print 'after 10000 iteration residual =', mt.sqrt(np.dot(r,r))
    return x, k
```

Решение модельной задачи дается следующей программой.

exer_5_1.py

```
import numpy as np
from seidel import seidel
n = 10
al = 0.5
A = np.zeros((n, n), 'float')
for i in range(0,n):
    A[i,i] = 2
    if i > 0:
        A[i,i-1] = - 1 + al
    if i < n-1:
        A[i,i+1] = - 1 - al
print 'A:\n', A
f = np.zeros((n), 'float')
f[0] = 1 - al
f[n-1] = 1 + al
print 'f:\n', f
x, iter = seidel(A, f)
print 'iter =', iter
print 'x:\n', x
```

```
A
[[ 2.   -1.5  0.   0.   0.   0.   0.   0.   0.   0. ]
 [-0.5  2.   -1.5  0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   -0.5  2.   -1.5  0.   0.   0.   0.   0.   0. ]
 [ 0.    0.  -0.5  2.   -1.5  0.   0.   0.   0.   0. ]
 [ 0.    0.   0.  -0.5  2.   -1.5  0.   0.   0.   0. ]
 [ 0.    0.   0.   0.  -0.5  2.   -1.5  0.   0.   0. ]
 [ 0.    0.   0.   0.   0.  -0.5  2.   -1.5  0.   0. ]
 [ 0.    0.   0.   0.   0.   0.  -0.5  2.   -1.5  0. ]
 [ 0.    0.   0.   0.   0.   0.   0.  -0.5  2.   -1.5]
 [ 0.    0.   0.   0.   0.   0.   0.   0.  -0.5  2. ]]
f
[ 0.5  0.   0.   0.   0.   0.   0.   0.   0.   1.5]
iter = 67
x
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

Данные по числу итераций для рассматриваемой задачи с диагональным преобладанием суммированы в табл. 5.1

Таблица 5.1 *Метод Зейделя*

α	$n = 25$	$n = 50$	$n = 100$
0	1233	4482	> 1000
0.25	289	410	599
0.5	107	159	254
0.75	58	93	160
1	25	50	100

При $\alpha \neq 0$ матрица системы уравнений несимметрична. Наблюдается вполне ожидаемое увеличение числа итераций при увеличении размерности задачи (параметр n) и сильная зависимость от параметра несимметричности α.

Упражнение 5.2 *Напишите программу, реализующую приближенное решение системы линейных алгебраических уравнений с симметричной положительно определенной матрицей методом сопряженных градиентов. С ее помощью найдите решение системы*

$$Ax = f$$

с матрицей Гильберта

$$a_{ij} = \frac{1}{i + j - 1}, \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., n$$

и правой частью

$$f_i = \sum_{j=1}^{n} a_{ij}, \quad i = 1, 2, ..., n,$$

для которой точное решение есть $x_i = 1, \quad i = 1, 2, ..., n$. *Исследуйте зависимость числа итераций от* n.

Расчетные формулы метода сопряженных градиентов возьмем (сравни с (5.24) при $B = E$) в виде:

$$x^{k+1} = x^{k+1} - \alpha_{k+1} s^k, \quad s^{k+1} = r^{k+1} + \beta_{k+1} s^k, \quad k = 0, 1, ...$$

при задании итерационных параметров $\alpha_{k+1}, \beta_{k+1}$ в виде

$$\alpha_{k+1} = \frac{(r^k, r^k)}{(s^k, As^k)}, \quad \beta_{k+1} = \frac{(r^{k+1}, r^{k+1})}{(r^k, r^k)} = -\frac{(r^{k+1}, As^k)}{(s^k, As^k)}.$$

В модуле cg функция cg() реализует описанный вариант метода сопряженных градиентов для системы с самосопряженной положительно определенной матрицей A. На выходе мы имеем приближенное решение и число итераций для достижения необходимой точности по невязке.

Модуль cg

```
import numpy as np
def cg(A, f, tol = 1.0e-9):
```

```
    """
    Solve the linear system Ax = b by conjugate gradient method
    """
    n = len(f)
    x = np.zeros((n), 'float')
    r = np.copy(f)
    for i in range(n):
        r[i] = np.dot(A[i,0:n], x[0:n]) - f[i]
    s = np.copy(r)
    As = np.zeros((n), 'float')
    for k in range(n):
        for i in range(n):
            As[i] = np.dot(A[i,0:n], s[0:n])
        alpha = np.dot(r, r) / np.dot(s, As)
        x = x - alpha*s
        for i in range(n):
            r[i] = np.dot(A[i,0:n], x[0:n]) - f[i]
        if np.dot(r, r) < tol**2:
            break
        else:
            beta = - np.dot(r, As) / np.dot(s, As)
            s = r + beta*s
    return x, k
```

Решение тестовой задачи дается следующей программой.

exer-5 2 py

```
import numpy as np
from cg import cg
n = 4
A = np.zeros((n, n), 'float')
for i in range(0,n):
    for j in range(0,n):
        A[i,j] = 1./(i+j+1)
print 'A:\n', A
f = np.ones((n), 'float')
for i in range(0,n):
    f[i] = 0.
    for j in range(0,n):
        f[i] = f[i] + A[i,j]
print 'f:\n', f
x, iter = cg(A, f)
print 'iter =', iter
print 'x:\n', x
```

A

```
[| 1           0 5          0.33333333  0 25        |
 | 0.5         0 33333333   0 25        0 2         |
 | 0.33333333  0.25         0.2         0 16666667 |
 | 0 25        0 2          0 16666667  0 14285714 ||
f
[ 2 08333333  1 28333333  0 95        0 75952381]
iter = 3
x
[ 1   1   1   1 ]
```

Зависимость числа итераций от размерности матрицы прослеживается по следующим данным: при $n = 25$ необходимо 13 итераций, при $n = 50 - 17$, а при $n = 100 - 20$.

5.4 Задачи

Задача 5.1 *Напишите программу, реализующую приближенное решение системы линейных алгебраических уравнений итерационным методом Якоби. С ее помощью найдите решение системы уравнений, рассмотренной в упражнении 5.1. Сравните скорости сходимости итерационных методов Якоби и Зейделя при различных параметрах n и α.*

Задача 5.2 *Напишите программу, реализующую приближенное решение системы линейных алгебраических уравнений методов релаксации (5.9). Исследуйте зависимость скорости сходимости этого итерационного метода от итерационного параметра $\tau_{k+1} = \tau$ при численном решении системы уравнений из упражнения 5.1 при различных n и α.*

Задача 5.3 *Напишите программу для решения системы линейных алгебраических уравнений явным ($B = E$) итерационным методом минимальных поправок (итерационные параметры определяются согласно (5.23)). Сравните скорость сходимости этого метода со скоростью сходимости метода сопряженных градиентов на примере задачи из упражнения 5.2.*

Задача 5.4 *Пусть*

$$A = A_1 + A_2 = A^* > 0, \quad A_2^* = A_1 = \frac{1}{2}D + L.$$

В попеременно-треугольном методе переобуславливатель B зададим в виде

$$B = (E + \omega A_1)(E + \omega A_2).$$

Напишите программу для решения системы линейных алгебраических уравнений с симметричной положительно определенной матрицей попеременно-треугольным методом с выбором итерационных параметров по (5.23). Исследуйте зависимость скорости сходимости этого итерационного метода от параметра ω в задаче из упражнения 5.2.

Спектральные задачи линейной алгебры

Важной и трудной задачей линейной алгебры является нахождение собственных значений и собственных векторов матриц. Рассматривается проблема устойчивости собственных значений по отношению к малым возмущениям элементов матрицы. Для приближенного нахождения отдельных собственных значений широко используется степенной метод в различных модификациях. Для решения полной проблемы для симметричных матриц применяется итерационный метод Якоби и QR-алгоритм.

Основные обозначения

$x = \{x_i\} = \{x_1, x_2, \ldots, x_n\}$	—	n-мерный вектор
$A = \{a_{ij}\}$	—	матрица с вещественными элементами a_{ij}
E	—	единичная матрица
$D = \operatorname{diag}\{d_1, d_2, \ldots, d_n\}$	—	диагональная матрица
$\lambda_i,\ i = 1, 2, \ldots, n$	—	собственные значения
$\varphi^i,\ i = 1, 2, \ldots, n$	—	собственные вектора
$(x, y) = \sum_{i=1}^{n} x_i y_i$	—	скалярное произведение

6.1 Собственные значения и собственные вектора матриц

Рассматриваются проблемы нахождения собственных значений и собственных векторов квадратной вещественной матрицы A. Собственным числом называется число λ такое, что для некоторого ненулевого вектора (собственного вектора) φ имеет место равенство

$$A\varphi = \lambda\varphi. \tag{6.1}$$

Собственные вектора определены с точностью до числового множителя. Множество всех собственных значений матрицы A называется спектром матрицы A.

С учетом того, что ищется нетривиальное решение уравнения (6.1), то

$$\det(A - \lambda E) = 0. \tag{6.2}$$

Тем самым собственные значения λ матрицы A являются корнями характеристического многочлена n-й степени (6.2). Задача отыскания собственных значений и собственных векторов матрицы сводится к построению характеристического многочлена, отысканию его корней и последующему нахождению нетривиальных решений уравнения (6.1) для найденных собственных значений.

В вычислительной практике рассматривается как полная проблема собственных значений, когда необходимо найти все собственные значения матрицы A, так и частичная проблема собственных значений, когда ищутся только некоторые собственные значения, например, максимальные (минимальные) по модулю.

6.2 Численные методы решения задач на собственные значения

Начнем с приведением некоторых полезных фактов о свойствах собственных значений и собственных векторов квадратной матрицы. Далее рассматриваются методы решения частичной и полной проблемы собственных значений.

Свойства собственных значений и собственных векторов

Квадратная вещественная матрица порядка n имеет n собственных значений, при этом каждое собственное значение считается столько раз, какова его кратность как корня характеристического многочлена. Для симметричной матрицы A собственные значения вещественны, а собственные вектора, соответствующие различным собственным значениям, ортогональны, т.е. $(\varphi^i, \varphi^j) = 0$, если $i \neq j$.

Отметим также некоторые свойства собственных значений и собственных векторов для сопряженной матрицы A^*:

$$A^*\psi = \mu\psi. \tag{6.3}$$

Для спектральных задач (6.1), (6.3) имеем

$$\lambda_i = \mu_i, \quad i = 1, 2, \ldots, n,$$

$$(\varphi^i, \psi^j) = 0, \quad i \neq j.$$

Две квадратные матрицы A и B одинаковых размеров называются по ными, если существует такая невырожденная матрица P, что $A = P^{-1}$ Подобные матрицы имеют одни и те же собственные значения, так ка (6.1) непосредственно следует

$$B\phi = \lambda\phi, \quad \phi = P\varphi.$$

Поэтому вычислительные алгоритмы решения спектральных задач баз ются на подобном преобразовании матрицы к матрице B, для которой с тральная задача решается проще. В качестве B естественно выбирать ди нальную матрицу, причем в данном случае это будет

$$\Lambda = \mathrm{diag}\{\lambda_1, \lambda_2, \ldots, \lambda_n\}.$$

Упорядочим собственные значения симметричной матрицы A по возра нию, т.е. $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$. Свойства собственных значений и собствен функций связаны с отношением Релея $\dfrac{(Ax,x)}{(x,x)}$. Отметим, например, что любого ненулевого вектора x справедливы неравенства

$$\lambda_1 \le \frac{(Ax,x)}{(x,x)} \le \lambda_n.$$

Важны также экстремальные свойства Релея

$$\lambda_1 = \min_{x\neq 0} \frac{(Ax,x)}{(x,x)}, \quad \lambda_n = \max_{x\neq 0} \frac{(Ax,x)}{(x,x)}.$$

Для локализации собственных значений произвольной матрицы A приме ются круги Гершгорина: любое собственное значение матрицы A лежи крайней мере в одном из кругов

$$|\lambda - a_{ii}| \le \sum_{i\neq j=1}^{n} |a_{ij}|, \quad i = 1, 2 \ldots, n.$$

Приведем теперь некоторые о возмущении собственных значений при воз щении элементов матрицы. Помимо (6.1) рассмотрим задачу

$$\tilde{A}\tilde{\varphi} = \tilde{\lambda}\tilde{\varphi}. \qquad (6.$$

Ограничимся случаем, когда все собственные значения простые. С то стью до членов второго порядка возмущение собственных значений за возмущения матрицы дается оценкой

$$|\tilde{\lambda} - \lambda| \le c_i \|\tilde{A} - A\|, \qquad (6.$$

где $\|x\| = (x,x)^{1/2}$. Мерой устойчивости собственного значения λ_i слу величина

$$c_i = \frac{\|\varphi^i\|\|\psi^i\|}{|(\varphi_i, \psi_i)|}, \qquad (6.$$

которая называется коэффициентом перекоса матрицы A, соответствующим данному собственному значению. Здесь ψ_i — собственное значение матрицы A^*. Для нормированных собственных векторов задач (6.1) и (6.5) соответствующая оценка устойчивости имеет вид

$$\|\tilde{\varphi}^i - \varphi^i\| \le \sum_{i \ne j=1}^{n} \frac{c_j}{|\lambda_i - \lambda_j|} \|\tilde{A} - A\|.$$

В частности, для симметричной матрицы все коэффициенты перекоса равны единице и оценки устойчивости вычисления собственных значений оптимальны.

Итерационные методы решения частичной проблемы собственных значений

Для нахождения минимального по модулю (максимального) собственного значения используется прямые и обратные итерации. Пусть матрица A является симметричной, все ее собственные значения простые и упорядочены следующим образом

$$|\lambda_1| < |\lambda_2| < \cdots < |\lambda_n|.$$

Определим последовательность векторов

$$x^{k+1} = Ax^k, \quad k = 0, 1, \ldots \tag{6.7}$$

при некотором заданном x^0 (прямые итерации). Рассматривая последовательности скалярных произведений (x^k, x^k), (x^{k+1}, x^k), при ограничении $(x^0, \varphi^n) \ne 0$, получим

$$\frac{(x^{k+1}, x^k)}{(x^k, x^k)} = \lambda_n + O\left(\left|\frac{\lambda_{n-1}}{\lambda_n}\right|^{2k}\right). \tag{6.8}$$

Тем самым при использовании итерационного процесса (6.7) находится максимальное по модулю собственное значение матрицы A.

Принимая во внимание, что собственные значения матрицы A^{-1} есть $1/\lambda_i$, $i = 1, 2, \ldots, n$, при использовании последовательности (обратные итерации)

$$y^{k+1} = A^{-1}y^k, \quad (y^0, \varphi^1) \ne 0, \quad k = 0, 1, \ldots \tag{6.9}$$

имеем

$$\lim_{k\to\infty} \frac{(y^{k+1}, y^k)}{(y^k, y^k)} = \frac{1}{\lambda_1}.$$

Тем самым при обратных итерациях находится минимальное по модулю собственное значение матрицы.

Заметим, что в прямых и обратных итерациях нет необходимости в специальном вычислении соответствующих собственных векторов, так как

$$\lim_{k\to\infty} x^k = \varphi^n, \quad \lim_{k\to\infty} y^k = \varphi^1. \tag{6.10}$$

Вычислительная реализация обратных итераций (6.9) может быть осно на однократном LU разложении матрицы A. После этого каждая обрат итерация по вычислительным затратам эквивалентна прямой итерации.

Отметим процедуру ускорения сходимости обратных итераций при извест хорошем приближении к собственному значению $\widetilde{\lambda}_1$ к собственному значе λ_1. В этом случае рассматриваются обратные итерации со сдвигом, когд

$$z^{k+1} = (A - \widetilde{\lambda}_1 E)^{-1} z^k, \quad (z^0, \varphi^1) \neq 0, \quad k = 0, 1, \ldots.$$

Скорость сходимости обратных итераций со сдвигом и без сдвига опред ется отношениями

$$\left|\frac{\lambda_1 - \widetilde{\lambda}_1}{\lambda_2 - \lambda_1}\right|, \quad \left|\frac{\lambda_1}{\lambda_2}\right|$$

соответственно. В более общей ситуации обратные итерации со сдвигом пользуются для нахождения ближайшего к заданному числу собствен значения соответствующего собственного вектора.

Решение полной проблемы собственных значений

Прямые и обратные итерации хорошо приспособлены для определения дельных собственных значений и собственных векторов. Для решения с тральной задачи в целом используется QR алгоритм. Он основан на предс ении матрицы A в произведение $A = QR$, где Q — ортогональная $Q^*Q =$ а R — верхняя треугольная матрицы.

Строится последовательность ортогональных матриц Q_k и верхних треуг ных матриц R_K по рекуррентным формулам

$$A = Q_1 R_1, \quad A_1 = R_1 Q_1,$$

$$A_1 = Q_2 R_2, \quad A_2 = R_2 Q_2,$$

$$A_{k-1} = Q_k R_k, \quad A_k = R_k Q_k,$$

.....

Процесс построения по матрице A матриц Q_k, R_k, A_k называется QR-а ритмом.

Пусть для невырожденной матрицы A собственные значения различны модулю и

$$|\lambda_1| > |\lambda_1| > \cdots > |\lambda_n|$$

и существует представление

$$A = T\Lambda T^{-1}, \quad T^{-1} = LU, \quad \Lambda = \operatorname{diag}\{\lambda_1, \lambda_2, \ldots, \lambda_n\}.$$

Тогда последовательность матриц A_k QR-алгоритма сходится к верхней треугольной матрице, а диагональные элементы — к собственным значениям матрицы A.

При решении полной спектральной задачи на основе QR для минимизации вычислительной работы проводится предварительное преобразование исходной матрицы к верхней почти треугольной матрице, в которой $a_{ij} = 0$, $i > j + 1$. При рекуррентных преобразованиях QR-алгоритма матрицы A_k остаются почти верхнетреугольными.

Решение спектральной проблемы для симметричной вещественной матрицы может осуществляться методом вращений (методом Якоби). Вещественная матрица, отличающаяся от единичной матрицы четырьмя элементами, расположенными на пересечении строк и столбцов с номерами i, j, и имеющая вид

$$T(ij) = \begin{bmatrix} & & i & & j & & \\ 1 & & \vdots & & \vdots & & 0 \\ & & \vdots & & \vdots & & \\ & & \vdots & & \vdots & & \\ & & c & \cdots & s & \cdots & \cdots & \cdots & i \\ & & \vdots & \ddots & \vdots & & & & \\ & & -s & \cdots & c & \cdots & \cdots & \cdots & j \\ & & & & & & & & \\ & & & & & & & & \\ 0 & & & & & & & 1 & \end{bmatrix},$$

где $c^2 + s^2 = 1$ называется матрицей вращения. Заметим, что матрица вращения является ортогональной и при умножении вектора на матрицу вращения $T(ij)$ меняются только i и j координаты вектора.

Для любой матрицы A и любой пары индексов i, j, $i \neq j$ всегда можно найти такую матрицу вращения $T(ij)$, что элемент b_{ij} матрицы $B = T^*(ij)AT(ij)$ равен нулю. Определим последовательность матриц

$$A_0 = A,\ A_1,\ A_2, \ldots,$$

каждая из которых получается из предыдущей с помощью преобразования подобия, определяемой матрицей вращения. На каждом шаге этого процесса обнуляется отдельный внедиагональный элемент. При таком преобразовании сумма квадратов внедиагональных элементов убывает. Последовательность матриц A_k сходится к диагональной матрице и диагональные элементы матрицы A_k в рассматриваемом методе вращений (метод Якоби) являются соответствующими приближениями для собственных значений матрицы A.

Оптимизация метода вращений достигается за счет выбора элемента для уничтожения на каждом шаге преобразований. Это может быть максимальный по модулю внедиагональный элемент всей матрицы A_k или на выбранном столбце.

6.3 Упражнения

Упражнение 6.1 *Напишите программу для нахождения минимального по модулю собственного значения и соответствующего собственного вектора симметричной матрицы при использовании обратных итераций. С ее помощью найдите минимальное по модулю собственное значение матрицы Гильберта A, когда*

$$a_{ij} = \frac{1}{i+j-1}, \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., n,$$

при значениях n от 2 до 10.

В модуле invIter функция `invIter()` обеспечивает нахождение минимального по модулю собственного значения и соответствующего собственного вектора с помощью обратных итераций (см. (6.9)). Для решения системы линейных уравнений используется функция solveLU из модуля lu.

Модуль `invIter`

```
import numpy as np
import math as mt
from lu import decLU, solveLU
def invIter(A, tol = 1.0e-6):
    '''
    The inverse power method.
    Returns the smallest eigenvalue and
    the corresponding eigenvector.
    '''
    n = len(A)
    iterMax = 100
    x = np.random.rand(n)
    xNorm = mt.sqrt(np.dot(x, x))
    x = x / xNorm
    for i in range(0, iterMax):
        x1 = x.copy()
        x = solveLU(A, x)
        xNorm = mt.sqrt(np.dot(x, x))
        x = x / xNorm
        if mt.sqrt(np.dot(x1-x, x1-x)) < tol:
            return 1 / xNorm, x
    print 'Method did not converge (100 iterations).'
```

Решение частичной проблемы собственных значений для матрицы Гильберта дается следующей программой.

exer_6_1.py

```
import numpy as np
from invIter import invIter
print 'n eigenvalue'
for n in range(2, 11):
    A = np.zeros((n, n), 'float')
    for i in range(0, n):
        for j in range(0, n):
            A[i,j] = 1./(i+j+1)
    lam, x = invIter(A)
    print n, lam
```

```
n eigenvalue
2 0.0657414540893
3 0.00268734035577
4 9.67023040226e-05
5 3.28792877217e-06
6 1.08279948448e-07
7 3.49389859222e-09
8 1.11153895367e-10
9 3.49968924865e-12
10 1.0933068413e-13
```

Приведенные расчетные данные демонстрируют быстрое приближение минимального собственного значения к нулю при увеличении размерности матрицы n.

Упражнение 6.2 *Напишите программу для нахождения собственных значений и собственных векторов симметричной вещественной матрицы методом вращений (методом Якоби). С ее помощью найдите собственные значение матрицы Лемера (Lehmer) A, для которой*

$$a_{ij} = \frac{\min(i,j)}{\max(i,j)}, \quad i = 1,2,...,n, \quad j = 1,2,...,n,$$

при $n = 8$.

Приведем необходимые расчетные формулы, которые связаны с применением матрицы вращения. Пусть

$$B = T^*(kl)AT(kl).$$

Для измененных элементов матрицы B имеем

$$b_{kk} = c^2 a_{kk} + s^2 a_{ll} - 2csa_{kl},$$

$$b_{ll} = s^2 a_{kk} + c^2 a_{ll} + 2csa_{kl},$$

$$b_{kl} = (c^2 - s^2)a_{kl} + cs(a_{kk} - a_{ll}),$$

$$b_{ki} = ca_{ki} - sa_{li}, \quad i \neq k, \quad i \neq l,$$

$$b_{li} = sa_{ki} + ca_{li}, \quad i \neq k, \quad i \neq l.$$

В силу этого, для обнуления элемента b_{kl} необходимо

$$(c^2 - s^2)a_{kl} + cs(a_{kk} - a_{ll}) = 0.$$

Пусть $c = \cos\theta, s = \sin\theta$ (θ — угол поворота) тогда достаточно положить

$$\phi = \frac{1}{\tan(2\theta)} = -\frac{a_{kk} - a_{ll}}{2a_{kl}}.$$

Полагая $t = s/c = \tan\theta$, для определения t получим квадратное уравнение

$$t^2 + 2\phi t - 1 = 0,$$

решение которого есть

$$t = -\phi \pm \sqrt{\phi^2 + 1}.$$

Меньший по модулю корень $|t| < 1$ соответствует повороту на угол меньший $\pi/4$ и порождает более устойчивый алгоритм. Поэтому

$$t = \operatorname{sign}(\phi)(-|\phi| + \sqrt{\phi^2 + 1}).$$

Для более точных вычислений при больших $|\phi|$ положим

$$t = \frac{\operatorname{sign}(\phi)}{|\phi| + \sqrt{\phi^2 + 1}},$$

причем для очень больших $|\phi|$ имеем $t \approx 1/(2\phi)$.

По значению t находим

$$c = \frac{1}{\sqrt{1 + t^2}}, \quad s = tc.$$

Расчетные формулы для элементов матрицы B принимают вид

$$b_{kk} = a_{kk} - ta_{kl},$$

$$b_{ll} = a_{ll} + ta_{kl},$$

$$b_{kl} = 0,$$

$$b_{ki} = a_{ki} - s(a_{li} + \tau a_{ki}), \quad i \neq k, \quad i \neq l,$$

$$b_{li} = a_{li} + s(a_{ki} - \tau a_{li}), \quad i \neq k, \quad i \neq l,$$

где $\tau = s/(1 + c)$.

В модуле jacobi функция `elemMax()` находит максимальный по модулю элемент матрицы, для обнуления отдельного элемента матрицы используется функция `rotate()`.

Модуль jacobi

```
import numpy as np
import math as mt
def elemMax(A):
    """
    Find the largest (absolute value) off-diagonal element
    A[k,l] in the upper half of A.
    """
    n = len(A)
    aMax = 0.
    for i in range(n-1):
        for j in range(i+1, n):
            if abs(A[i,j]) >= aMax:
                aMax = abs(A[i,j])
                k = i
                l = j
    return aMax, k, l
def rotate(A, P, k, l):
    """
    Rotate of A to make A[k,l] = 0.
    """
    n = len(A)
    d = A[l,l] - A[k,k]
    if abs(A[k,l]) < abs(d)*1.0e-36:
        t = A[k,l] / d
    else:
        phi = d / (2*A[k,l])
        t = 1 / (abs(phi) + mt.sqrt(1 + phi**2))
        if phi < 0.:
            t = - t
    c = 1 / mt.sqrt(1 + t**2)
    s = t*c
    tau = s / (1 + c)
    #  Modify the matrix elements
    tt = A[k,l]
    A[k,l] = 0.0
    A[k,k] = A[k,k] - t*tt
    A[l,l] = A[l,l] + t*tt
    for i in range(k):
        tt = A[i,k]
        A[i,k] = tt - s*(A[i,l] + tau*tt)
        A[i,l] = A[i,l] + s*(tt - tau*A[i,l])
    for i in range(k+1,l):
        tt = A[k,i]
```

```
            A[k,i] = tt - s*(A[i,l] + tau*A[k,i])
            A[i,l] = A[i,l] + s*(tt - tau*A[i,l])
        for i in range(l+1,n):
            tt = A[k,i]
            A[k,i] = tt - s*(A[l,i] + tau*tt)
            A[l,i] = A[l,i] + s*(tt - tau*A[l,i])
        #  Update transformation matrix
        for i in range(n):
            tt = P[i,k]
            P[i,k] = tt - s*(P[i,l] + tau*P[i,k])
            P[i,l] = P[i,l] + s*(tt - tau*P[i,l])
    def jacobi(A, tol = 1.0e-12):
        """
        Solution of eigenvalue problem by Jacobi's method.
        Returns eigenvalues in vector lam
        and the eigenvectors as columns of matrix .
        """
        n = len(A)
        # Number of rotations limit
        rotMax = 5*(n**2)
        P = np.identity(n)
        # Jacobi rotation loop
        for i in range(rotMax):
            aMax, k, l = elemMax(A)
            if aMax < tol:
                return np.diagonal(A), P
            rotate(A, P, k, l)
        print 'Jacobi method did not converge'
```

Решение полной проблемы собственных значений для матрицы Гильбе при использовании метода Якоби дается следующей программой.

exer −6.2 py

```
import numpy as np
from jacobi import jacobi
n = 8
A = np.zeros((n, n), 'float')
for i in range(0, n):
    for j in range(0, n):
        if i < j:
            A[i,j] = (i+1.)/(j+1)
        else:
            A[i,j] = (j+1.)/(i+1)
lam, x = jacobi(A)
print 'All eigenvalue:\n', lam
```

```
All eigenvalue
| 0 74602786   1 46486865   0 2760521   0 439372
  0 12603956   0 18325435   0.08707296  4.67731251|
```

Сравнение с результатами вычисления минимального собственного значения методом Якоби и обратными итерациями (см. упражнение 6.1) показывает хорошее согласие результатов.

6.4 Задачи

Задача 6.1 *Напишите программу для нахождения ближайшего к заданному числу собственного значения и соответствующего собственного вектора симметричной вещественной матрицы при использовании обратных итераций со сдвигом. С ее помощью найдите первые три минимальных по модулю собственных значение матрицы Паскаля A, для которой*

$$a_{ij} = \frac{(i+j-2)!}{(i-1)!(j-1)!} \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., n.$$

при $n = 8$.

Задача 6.2 *Напишите программу для нахождения минимального по модулю собственного значения и соответствующего собственного вектора симметричной трехдиагональной матрицы. С ее помощью найдите первое минимальное по модулю собственное значение матрицы* $n \times n$, *для которой*

$$a_{ii} = 2, \quad a_{i,i-1} = a_{i,i+1} = -1$$

при различных n. *Сравните численное решение с точным.*

Задача 6.3 *Напишите программу для преобразования симметричной вещественной матрицы A к симметричной трехдиагональной матрице PAP с помощью ортогонального преобразования Хаусхольдера* ($P^{-1} = P$). *С ее помощью проведите преобразование матрицы A, элементы которой есть*

$$a_{ij} = \min(i, j), \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., n,$$

при различных n.

Задача 6.4 *Напишите программу для вычисления собственных значений трехдиагональной вещественной матрицы A с использованием QR алгоритма. Найдите собственные числа трехдиагональной матрицы A, для которой*

$$a_{ii} = 2, \quad a_{i,i+1} = -1 - \alpha, \quad a_{i,i-1} = -1 + \alpha,$$

при различных значениях n *и параметра* α ($0 \le \alpha \le 1$). *Сравните найденные собственные значения с точными.*

Нелинейные уравнения и системы

Многие прикладные задачи приводят к необходимости нахождения приближенного решения нелинейных уравнений и систем нелинейных уравнений. С этой целью используются итерационные методы. Приведены алгоритмы решения нелинейных уравнений с одним неизвестным и систем нелинейных уравнений. Применяются итерационные метод последовательных приближений (простой итерации) и метод Ньютона в различных модификациях.

Основные обозначения

$f(x)$	—	функция одной переменной
$f_i(x),\ i = 1, 2, \ldots, n$	—	функции n переменных $(x = \{x_i\} = \{x_1, x_2, \ldots, x_n\})$
$F(x) = \{f_1, f_2, \ldots, f_n\}$	—	вектор-функция с компонентами $f_1, f_2, \ldots, f_n$
$F'(x)$	—	матрица Якоби
x^k	—	приближенное решение на k-й итерации

7.1 Решение нелинейных уравнений и систем

Ищется решение нелинейного уравнения

$$f(x) = 0, \tag{7.1}$$

где $f(x)$ — заданная функция. Корни уравнения (7.1) могут быть комплексными и кратными. Выделяют как самостоятельную проблему разделения корней, когда проводится выделение области в комплексной плоскости, содержащей один корень. После этого на основе тех или иных итерационных процессов при выбранном начальном приближении находится решение нелинейного уравнения (7.1).

В более общем случае мы имеем не одно уравнение (7.1), а систему нелинейных уравнений

$$f_i(x_1, x_2, \ldots, x_n) = 0, \quad i = 1, 2, \ldots, n. \tag{7.2}$$

Обозначим через $x = \{x_1, x_2, \ldots, x_n\}$ вектор неизвестных и определим вектор-функцию $F(x) = \{f_1, f_2, \ldots, f_n\}$. Тогда система (7.2) записывается в виде уравнения

$$F(x) = 0. \tag{7.3}$$

Частным случаем (7.3) является уравнение (7.1) ($n = 1$). Второй пример (7.3) — система линейных алгебраических уравнений, когда $F(x) = Ax - f$.

7.2 Итерационные методы решения нелинейных уравнений

Для приближенного решения нелинейных уравнений и систем нелинейных уравнений используются итерационные методы. Среди основных подходов можно выделить метод последовательных приближений (простой итерации) и метод Ньютона.

Алгоритмы для решения нелинейного уравнения

При итерационном решении уравнений (7.1), (7.3) задается некоторое начальное приближение, достаточно близкое к искомому решению x^*. В одношаговых итерационных методах новое приближение x^{k+1} определяется по предыдущему приближению x^k. Говорят, что итерационный метод сходится с линейной скоростью, если $x^{k+1} - x^* = O(x^k - x^*)$ и итерационный метод имеет квадратичную сходимость, если $x^{k+1} - x^* = O(x^k - x^*)^2$.

Заменим уравнение (7.1) эквивалентным уравнением

$$x = \varphi(x), \tag{7.4}$$

полагая, например,

$$\varphi(x) = x + g(x)f(x),$$

где функция $g(x)$ не меняет знака на отрезке, на котором ищется решение уравнения (7.1). Для приближенного решения уравнения (7.4) используется метод простой итерации, когда

$$x^{k+1} = \varphi(x^k), \quad k = 0, 1, \ldots \tag{7.5}$$

при некотором заданном начальном приближении x^0.

Пусть в некоторой окрестности $R = \{x \mid |x - x^*| \le r\}$ корня $x = x^*$ уравнения (7.4) функция $\varphi(x)$ удовлетворяет условию Липшица

$$|\varphi(x) - \varphi(y)| \le q|x - y| \quad x, y \in R \tag{7.6}$$

с постоянной $q < 1$. Тогда метод простой итерации (7.5) сходится и для погрешности верна оценка

$$|x^k - x^*| \le q^k |x^0 - x^*|. \tag{7.7}$$

Можно сформулировать условия, гарантирующие, что имеется единственный корень в окрестности начального приближения x^0. Пусть теперь $R = \{x \mid |x - x^0| \le r\}$ и

$$|x^0 - \varphi(x^0)| \le (1 - q)r, \tag{7.8}$$

тогда при выполнении (7.6) с $q < 1$ уравнение (7.4) имеет единственное решение в R.

В итерационном методе Ньютона (методе касательных) для нового приближения имеем

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}, \quad k = 0, 1, \ldots, \quad f'(x) \equiv \frac{df}{dx}(x). \tag{7.9}$$

Пусть x^* — простой вещественный корень уравнения (7.1) и определим $R = \{x \mid |x - x^*| \le r\}$ — окрестность этого корня. Предположим также, что

$$\inf_{x \in R} |f'(x)| = m > 0, \quad \sup_{x \in R} |f''(x)| = M,$$

причем

$$|x^0 - x^*| < \frac{2m}{M}.$$

Тогда при $x^0 \in R$ метод Ньютона (7.9) сходится и для погрешности справедлива оценка

$$|x^k - x^*| \le q^{2^k - 1} |x^0 - x^*|.$$

Тем самым метод Ньютона имеет квадратичную сходимость.

Модификации метода Ньютона направлены на минимизацию вычислительной работы, на увеличение окрестности корня, в которой можно задавать начальное приближение. Примером выступает метод секущих, который получается из метода Ньютона заменой производной в знаменателе на соответствующую разделенную разность:

$$x^{k+1} = x^k - \frac{x^k - x^{k-1}}{f(x^k) - f(x^{k-1})} f(x^k), \quad k = 0, 1, \ldots. \tag{7.10}$$

Этот метод является простейшим двухшаговым итерационным методом, когда новое приближение x^{k+1} находится по двум предыдущим x^k и x^{k-1}.

Методы решения систем нелинейных уравнений

При приближенном решении систем нелинейных уравнений (7.3) можно ориентироваться на аналоги метода многомерные простой итерации и метода

Ньютона. Многие одношаговые методы для приближенного решения (7.3) по аналогии с двухслойными итерационными методами для решения систем линейных алгебраических уравнений можно записать в виде

$$B_{k+1}\frac{x^{k+1}-x^k}{\tau_{k+1}}+F(x^k)=0,\quad k=0,1,\ldots, \tag{7.11}$$

где τ_{k+1} — итерационные параметры, а B_{k+1} — квадратная матрица $n\times n$, имеющая обратную.

Для стационарного итерационного метода (7.11) (B и τ не зависят от k) имеем

$$x^{k+1}=S(x^k), \tag{7.12}$$

где $S(x)=x-\tau B^{-1}F(x)$. Тем самым (7.12) соответствует применению метода простой итерации для преобразованного уравнения

$$x=S(x). \tag{7.13}$$

Пусть в окрестности $R=\{x\mid \|x-x^0\|\le r\}$ заданного начального приближения x^0 выполнены условия

$$\|S(x)-S(y)\|\le q\|x-y\|,\quad x,y\in R,$$

$$\|x^0-S(x^0)\|\le(1-1)r,\quad q<1.$$

Тогда уравнение (7.13) имеет в R единственное решение x^*, которое дается итерационным процессом (7.12), причем для погрешности справедлива оценка

$$\|x^{k+1}-x^*\|\le q^k\|x^0-x^*\|.$$

В методе Ньютона новое приближение для решения системы уравнений (7.2) определяется из решения системы линейных уравнений

$$\sum_{j=1}^{n}(x_j^{k+1}-x_j^k)\frac{\partial f_i(x^k)}{\partial x_j}+f_i(x^k)=0, \tag{7.14}$$

$$i=1,2,\ldots,n,\quad k=0,1,\ldots.$$

Определим матрицу Якоби

$$F'(x)=\begin{bmatrix}\dfrac{\partial f_1(x)}{\partial x_1} & \dfrac{\partial f_1(x)}{\partial x_2} & \dfrac{\partial f_1(x)}{\partial x_n}\\ \dfrac{\partial f_2(x)}{\partial x_1} & \dfrac{\partial f_2(x)}{\partial x_2} & \dfrac{\partial f_2(x)}{\partial x_n}\\ \\ \dfrac{\partial f_n(x)}{\partial x_1} & \dfrac{\partial f_n(x)}{\partial x_2} & \dfrac{\partial f_n(x)}{\partial x_n}\end{bmatrix}$$

и запишем (7.14) в виде

$$F'(x)(x^{k+1} - x^k) + F(x^k) = 0, \quad k = 0, 1, \ldots. \tag{7.15}$$

При использовании записи (7.11) метод Ньютона (7.15) соответствует выбору

$$B_{k+1} = F'(x^k), \quad \tau_{k+1} = 1.$$

Система линейных уравнений (7.15) для нахождения нового приближения x^{k+1} может решаться итерационно. В этом случае мы имеет двухступенчатый итерационный процесс со внешними и внутренними итерациями. Например, внешний итерационный процесс может осуществляться по методу Ньютона, а внутренние итерации — на основе итерационного метода Зейделя.

При решении систем нелинейных уравнений можно использовать прямые аналоги стандартных итерационных методов, которые применяются для решения систем линейных уравнений. Нелинейный метод Зейделя применительно к решению (7.2) дает

$$f_i(x_1^{k+1}, x_2^{k+1}, \ldots, x_i^{k+1}, x_{i+1}^k, \ldots, x_n) = 0, \quad i = 1, 2, \ldots, n.$$

В этом случае каждая компонента нового приближения из решения нелинейного уравнения, что можно сделать на основе итерационных метода простой итерации и метода Ньютона в различных модификациях. Тем самым снова приходим к двухступенчатому итерационному методу, в котором внешние итерации проводятся в соответствии с методом Зейделя, а внутренние — методом Ньютона.

Основные вычислительные сложности применения метода Ньютона для приближенного решения систем нелинейных уравнений связаны с необходимостью решения линейной системы уравнений с матрицей Якоби на каждой итерации, причем от итерации к итерации эта матрица меняется. В модифицированном методе Ньютона

$$F'(x^0)(x^{k+1} - x^k) + F(x^k) = 0, \quad k = 0, 1, \ldots$$

матрица Якоби обращается только один раз.

7.3 Упражнения

Упражнение 7.1 *Напишите программу для нахождения решения нелинейного уравнения $f(x) = 0$ методом бисекций. С ее помощью найдите корни уравнения $(1 + x^2)e^{-x} + \sin(x) = 0$ на интервале $[0, 10]$.*

В модуле bisection функция `bisection()` обеспечивает нахождение корня уравнения $f(x) = 0$ на интервале $[x_1, x_2]$ при условии, что $f(x_1)f(x_2) < 0$.

Модуль `bisection`

```
import numpy as np
import math as mt
def bisection(f, x1, x2, tol=1.0e-10):
    """
    Finds a root of f(x) = 0
    between the arguments x1 and x2 by bisection.
    f(x1) and f(x2) can not have the same signs.
    """
    f1 = f(x1)
    f2 = f(x2)
    if f1*f2 > 0.:
        print 'f(x1) and f(x2) can not have the same signs'
    n = int(mt.ceil(mt.log(abs(x2 - x1)/tol)/mt.log(2.)))
    for i in range(n):
        x3 = 0.5*(x1 + x2)
        f3 = f(x3)
        if f2*f3 < 0.:
            x1 = x3
            f1 = f3
        else:
            x2 =x3
            f2 = f3
    return (x1 + x2)/2.
```

Для определения интервалов, которые содержат корни уравнения $f(x) = 0$ будем использовать график $y = f(x)$. С учетом этого для нахождения корней уравнения $(1 + x^2)e^{-x} + \sin(x) = 0$ используется следующая программа.

exer_7_1.py

```
import numpy as np
import matplotlib.pyplot as plt
from bisection import bisection
def f(x):
    return (1. + x**2) * np.exp(-x) + np.sin(x)
x = np.linspace(0., 10., 200)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.grid(True)
plt.show()
xRoot1 = bisection(f, 2, 4)
print 'root(1) = ', xRoot1
xRoot2 = bisection(f, 6, 8)
print 'root(2) = ', xRoot2
xRoot3 = bisection(f, 8, 10)
```

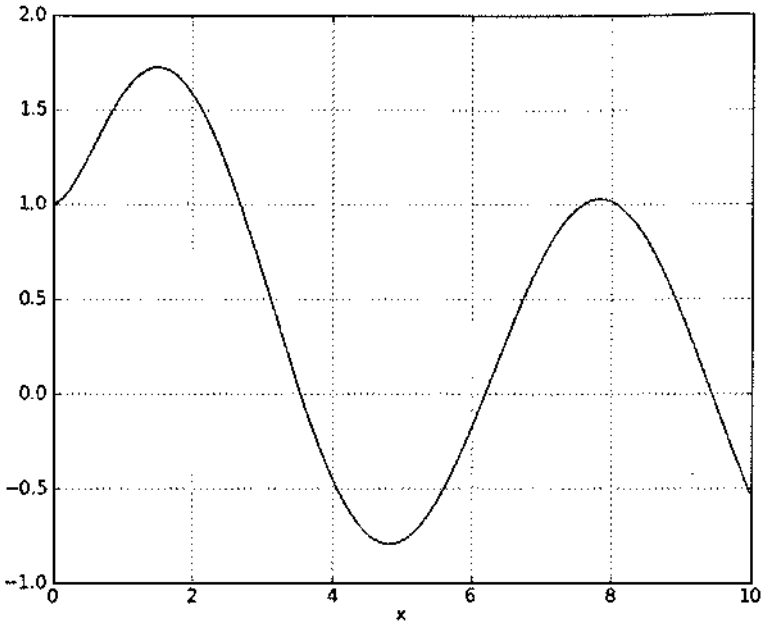

Рис. 7.1 График функции $y = (1 + x^2)e^{-x} + \sin(x)$

```
print 'root(3) = ', xRoot3
```

```
root(1) =  3.54419311814
root(2) =  6.20323687073
root(3) =  9.4319858017
```

На интервале [0,10] найдены три корня (см. рис. 7.1).

Упражнение 7.2 *Напишите программу для нахождения решения системы нелинейных уравнений $F(x) = 0$ методом Ньютона при численном вычислении матрицы Якоби. С ее помощью найдите приближенное решение системы*

$$(3 + 2x_1)x_1 - 2x_2 = 3,$$

$$(3 + 2x_i)x_i - x_{i-1} - 2x_{i+1} = 2, \quad i = 2, 3, ..., n - 1,$$

$$(3 + 2x_n)x_n - x_{n-1} = 4$$

при $n = 10$ и сравните его с точным решением $x_i = 1$, $i = 1, 2, ..., n$.

В модуле newton функция `jacobian()` предназначена для вычисления элементов матрицы Якоби с использованием разностной производной. Функция `newton()` реализует метод Ньютона при решении соответствующей системы

линейных уравнений с помощью LU-разложения (функция solveLU() из модуля lu).

Модуль newton

```
import numpy as np
import math as mt
from lu import solveLU
def jacobian(f, x):
    """
    Calculation of the Jacobian using finite differences.
    """
    h = 1.0e-4
    n = len(x)
    Jac = np.zeros((n,n), 'float')
    f0 = f(x)
    for i in range(n):
        tt = x[i]
        x[i] = tt + h
        f1 = f(x)
        x[i] = tt
        Jac[:,i] = (f1 - f0)/h
    return Jac, f0
def newton(f, x, tol=1.0e-9):
    """
    Solves the system equations f(x) = 0 by
    the Newton method using {x} as the initial guess.
    Solve the linear system Ax = b by lu module.
    """
    iterMax = 50
    for i in range(iterMax):
        Jac, f0 = jacobian(f, x)
        if mt.sqrt(np.dot(f0, f0) / len(x)) < tol:
            return x, i
        dx = solveLU(Jac, f0)
        x = x - dx
    print 'Too many iterations for the Newton method'
```

В тестовой задаче метод Ньютона применяется при начальном приближении $x_i^0 = 0,\ i = 1, 2, ..., n$ и с использованием следующей программы.

exer − 7.2 py

```
import numpy as np
from newton import newton
n = 10
def f(x):
```

```
    f = np.zeros((n), 'float')
    for i in range(1,n-1):
        f[i] = (3 + 2*x[i])*x[i] - x[i-1] - 2*x[i+1] - 2
    f[0] = (3 + 2*x[0])*x[0] - 2*x[1] - 3
    f[n-1] = (3 + 2*x[n-1])*x[n-1] - x[n-2] - 4
    return f
x0 = np.zeros((n), 'float')
x, iter = newton(f, x0)
print 'Newton iteration = ', iter
print 'Solution:\n', x
```

```
Newton iteration =  9
Solution:
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

При используемых параметрах метода точное решение задачи совпадает с численным.

7.4 Задачи

Задача 7.1 *Напишите программу для решения нелинейного уравнения $f(x) = 0$ методом секущих. Используйте ее для решения уравнения $4\sin(x) + 1 - x = 0$ на интервале $[-10, 10]$.*

Задача 7.2 *Напишите программу для решения нелинейного уравнения $f(x) = 0$ методом Ньютона. С ее помощью найдите положительные корни уравнения $x^2 - 10\sin(x) = 0$.*

Задача 7.3 *Напишите программу для нахождения решения системы нелинейных уравнений $F(x) = 0$ методом Ньютона при аналитическом задании матрицы Якоби. С использованием этой программы найдите приближенное решение системы уравнений*

$$n - \sum_{j=1}^{n} (\cos x_j + i(1 - \cos x_i) - \sin x_i) = 0 \quad i = 1, 2, ..., n,$$

при $n = 20$ и начальном приближении $x_i^0 = 1/n$, $i = 1, 2, ..., n$.

Задача 7.4 *Напишите программу для нахождения решения системы нелинейных уравнений $F(x) = 0$ методом Зейделя при решении нелинейного уравнения методом Ньютона. Рассмотрите возможности использования этой программы при приближенном решении системы уравнений из упражнения 7.2.*

Задачи минимизации функций

Среди основных проблем вычислительной математики можно отметить задачи минимизации функций многих переменных (задачи оптимизации). Поиск минимума часто проводится при некоторых дополнительных ограничениях — условная оптимизация. Для численного решения таких задач используются итерационные методы. В задачах с ограничениями применяются методы штрафных функций. Простейшей задачей рассматриваемого класса является поиск минимума одномерной функции.

Основные обозначения

$x = \{x_i\} = \{x_1, x_2, \ldots, x_n\}$	—	n-мерный вектор
$f(x)$	—	функция одной или n переменных
$f(x) \to \min, \quad x \in \mathbf{R}^n$	—	задача безусловной оптимизации
$f(x) \to \min, \quad x \in X$	—	задача условной оптимизации
X	—	допустимое множество
x^k	—	приближенное решение на k-й итерации
$(x, y) = \sum_{i=1}^{n} x_i y_i$	—	скалярное произведение

8.1 Поиск минимума функции многих переменных

Для заданной функции $f(x)$, определенной на допустимом множестве X из евклидова пространства $\mathbf{R}^n$, ищется точки минимума (максимума) функции $f(x)$, т.е.

$$f(x) \to \min, \quad x \in X. \tag{8.1}$$

Точка $x^* \in X$ есть точка глобального минимума функции $f(x)$ на множестве X, если

$$f(x^*) \le f(x), \quad \forall x \in X, \tag{8.2}$$

и точка локального минимума, если $f(x^*) \le f(x)$ в окрестности точки $x^* \in X$.

Задача (8.1) называется задачей безусловной оптимизации, если $X = \mathbf{R}^n$, т.е.

$$f(x) \to \min, \quad x \in \mathbf{R}^n. \tag{8.3}$$

Если X некоторое подмножество пространства $\mathbf{R}^n$, то мы имеем задачу условной оптимизации. Такие задачи существенно сложнее для численного решения, чем задачи безусловной минимизации. Ограничения могут формулироваться в виде равенств (например, $g_i(x) = 0,\ i = 1, 2, \ldots, m$) или неравенств ($g_i(x) \le 0,\ i = 1, 2, \ldots, m$).

8.2 Методы решения задач оптимизации

Вычислительные алгоритмы для приближенного решения задачи оптимизации чаще всего строятся на использовании необходимых и достаточных условиях оптимальности, т.е. на условиях, которые имеют место в точке минимума. Реализация такого подхода связана с решением соответствующих нелинейных уравнений итерационными методами.

Поиск минимума функции одной переменной

Пусть $X = [a, b]$ и кусочно-непрерывная функция $f(x)$ имеет в некоторой точке $x^* \in X$ один минимум. Мы отметим прежде всего простейшие итерационные методы решения задачи минимизации, наиболее полно учитывающие специфику одномерных задач.

Вычислим функцию на концах отрезка и в двух внутренних точках x^1 и $x^2 < x^1$. Будем считать, что эти точки симметричны относительно середины отрезка $[a, b]$. В методе золотого сечения точки x^1 и x^2 выбираются так, чтобы отношение длины всего отрезка $[a, b]$ к длине большей из его частей $[a, x^1]$ равно отношению длины большей части $[a, x^1]$ к длине меньшей части $[x^1, b]$:

$$\frac{b-a}{x^1-a} = \frac{x^1-a}{x^2-a}. \tag{8.4}$$

Далее проводится сравнение значений функции в четырех точках a, x^2, x^1, b и выберем точку, в которой значение функции наименьшее. Пусть это будет точка x^2, тогда минимум функции достигается в одном из прилегающих к этой точке отрезков: $[a, x^2]$ или $x^2, x^1]$ и поэтому в дальнейшем можно рассматривать проблему минимизации на отрезке $[a, x^1]$. После этого процесс повторяется — в соответствии с правилом золотого сечения делится точкой x^3 отрезок $[a, x^1]$.

Для минимизации функции одной переменной широко используются методы полиномиальной интерполяции. В этом случае с использованием ранее найденных точек строится интерполяционный полином, точка минимума которого принимается за очередное приближение. В методе парабол используется интерполяционный многочлен второго порядка.

Пусть, например, известны три приближения $x^{k-2} < x^{k-1}$ и $x^{k-2} < x^k < x^{k-1}$, причем $f(x^k) < f(x^{k-2})$ и $f(x^k) < f(x^{k-1})$. Новое приближение ищется как решение задачи минимизации

$$L_2(x) \to \min, \tag{8.5}$$

где $L_2(x)$ — интерполяционный многочлен второго порядка, построенный по узлам x^{k-2}, x^{k-1} и x^k.

Интерполяционный многочлен Ньютона имеет вид

$$L_2(x) = f(x^k) + (x - x^k)f(x^k, x^{k-1}) +$$

$$+(x - x^k)(x - x^{k-1})f(x^k, x^{k-1}, x^{k-2}),$$

где

$$f(x^k, x^{k-1}) = \frac{f(x^k) - f(x^{k-1})}{x^k - x^{k-1}},$$

$$f(x^k, x^{k-1}, x^{k-2}) = \frac{f(x^{k-1}, x^{k-2}) - f(x^k, x^{k-1})}{x^{k-2} - x^k}$$

разделенные разности первого и второго порядка соответственно. Решение задачи (8.5) приводит нас к следующей формуле для нового приближения для точки минимума

$$2x^{k+1} = x^k + x^{k-1} - \frac{f(x^k, x^{k-1})}{f(x^k, x^{k-1}, x^{k-2})}. \tag{8.6}$$

Для дифференцируемой функции $f(x)$ строятся итерационные методы, основанные на решении уравнения (необходимое условие минимума)

$$f'(x) = 0. \tag{8.7}$$

Корень этого уравнения $x^* \in X$ является точкой минимума, если $f''(x) > 0$ (достаточные условия минимума). Для приближенного решения нелинейного уравнения используются итерационные методы.

В итерационном методе Ньютона новое приближение для точки минимума определяется в соответствии с формулой

$$x^{k+1} = x^k - \frac{f'(x^k)}{f''(x^k)}, \quad k = 0, 1, \ldots \tag{8.8}$$

Различные модификации метода Ньютона рассматривались в главе 8.

Минимизация функций многих переменных

Для функции многих переменных $f(x)$, $x = \{x_1, x_2, \ldots, x_n\}$ определим вектор первых частных производных (градиент)

$$f'(x) = \left\{\frac{\partial f}{\partial x_i}(x)\right\} \equiv \left\{\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \ldots, \frac{\partial f}{\partial x_n}(x)\right\}.$$

Матрица вторых частных производных (гессиан) в точке x есть

$$f''(x) = \left\{ \frac{\partial^2 f}{\partial x_i \partial x_j}(x) \right\}.$$

Будем рассматривать задачу безусловной оптимизации (8.3). Пусть функция $f(x)$ дифференцируема в точке локального минимума $x = x^*$, тогда (необходимые условия оптимальности)

$$f'(x^*) = 0. \tag{8.9}$$

Если функция $f(x)$ дважды дифференцируема в точке $x = x^*$ и выполнено (8.9). Если матрица $f''(x)$ положительно определена, т.е.

$$(f''(x^*)y, y) > 0, \quad \forall y \neq 0, \tag{8.10}$$

тогда x^* — точка локального минимума. Условия (8.9), (8.10) есть достаточные условия оптимальности.

Для итерационных методов минимизации будем использовать обозначения

$$x^{k+1} = x^k + \alpha_k h^k, \quad k = 0, 1, \ldots, \tag{8.11}$$

где h^k — вектор, который определяет направление $(k+1)$-го шага минимизации, а коэффициент α_k — длину этого шага.

Вектор h задает направление убывания функции $f(x)$ в точке x, если $f(x + \alpha h) < f(x)$ при достаточно малых $\alpha > 0$. Если вектор h^k задает направление убывания функции $f(x)$ в точке x^k, а $\alpha_k > 0$ такое, что

$$f(x^{k+1}) < f(x^k),$$

то итерационный метод (8.11) называется методом спуска. В градиентном методе $h^k = -f'(x^k)$, т.е.

$$x^{k+1} = x^k - \alpha_k f'(x^k), \quad k = 0, 1, \ldots \tag{8.12}$$

Особое внимание уделяется выбору итерационных параметров α_k, $k = 0, 1, \ldots$ в методе (8.11). Их можно определять из условия

$$f(x^k + \alpha_k h^k) = \min_{\alpha \geq 0} f(x^k + \alpha h^k),$$

т.е. из решения дополнительной одномерной задачи минимизации.

В вычислительной практике широко используется процедура дробления шага, когда параметр α уменьшается, например, в два раза, до тех пор пока не будет выполнено неравенство

$$f(x^k + \alpha h^k) < f(x^k).$$

При применении метода Ньютона для решения системы нелинейных уравнений (8.9) получим

$$f''(x^k)(x^{k+1} - x^k) + f'(x^k) = 0, \quad k = 0, 1, \ldots \tag{8.13}$$

Он записывается в виде (8.11) при

$$\alpha_k = 1, \quad h^k = -(f''(x^k))^{-1} f'(x^k). \tag{8.14}$$

Среди модификаций метода Ньютона отметим метод Ньютона с регулировкой шага, когда вместо (8.14) используется

$$\alpha_k > 0, \quad h^k = -(f''(x^k))^{-1} f'(x^k).$$

В квазиньютоновских методах

$$h^k = -H_k f'(x^k),$$

где H_k — матрица, которая аппроксимирует матрицу $(f''(x^k))^{-1}$.

Задачи условной минимизации

При минимизации функций с ограничениями широко используются подходы, которые аналогичны тем, которые разработаны для задач безусловной минимизации. Наиболее просто это реализуется при переходе от задачи условной минимизации к задаче минимизации без ограничений.

Рассмотрим задачу минимизации с ограничениями типа равенств:

$$f(x) \to \min, \quad g_i(x) = 0, \quad i = 1, 2, \ldots, m. \tag{8.15}$$

Эту задачу можно записать в общем виде (8.1), задав допустимое множество

$$X = \{x \in \mathbf{R}^n \mid g_i(x) = 0,\ i = 1, 2, \ldots, m\}.$$

При некоторых ограничениях задача условной минимизации (8.15) эквивалентна задаче безусловной минимизации функции Лагранжа

$$\Phi(x, y) = f(x) + \sum_{i=1}^{m} y_i g_i(x),$$

где y_i — неизвестные множители Лагранжа. Тем самым мы приходим к задаче минимизации функции $n + m$ переменных.

Более сложно перейти к задаче безусловной оптимизации при учете ограничений в виде неравенств. Рассмотрим, например, задачу

$$f(x) \to \min, \quad g_i(x) \le 0, \quad i = 1, 2, \ldots, m, \tag{8.16}$$

т.е. в (8.1)

$$X = \{x \in \mathbf{R}^n \mid g_i(x) \le 0,\ i = 1, 2, \ldots, m\}.$$

Вместо функции $f(x)$ в методе штрафов минимизируется функция

$$\Phi(x, \varepsilon) = f(x) + \psi(x, \varepsilon), \tag{8.17}$$

где $\psi(x,\varepsilon)$ -- штрафная функция, $\varepsilon > 0$ -- параметр штрафа. Выбор штрафной функции на допустимом множестве подчинен условиям

$$\psi(x,\varepsilon) \geq 0, \quad \psi(x,\varepsilon) \to 0, \text{ если } \varepsilon \to 0, \quad x \in X,$$

а вне допустимого множества —

$$\psi(x,\varepsilon) \to \infty, \text{ если } \varepsilon \to 0, \quad x \notin X.$$

В качестве характерного примера приведем штрафную функцию для задачи условной минимизации (8.16):

$$\psi(x,\varepsilon) = \frac{1}{\varepsilon}\sum_{i=1}^{m}[\max\{0, g_i(x)\}]^2.$$

После этого рассматривается задача безусловной минимизации

$$\Phi(x,\varepsilon) \to \min, \quad x \in \mathbf{R}^n.$$

Помимо выбора штрафной функции в методах этого класс очень важен выбор величины параметра штрафа ε.

8.3 Упражнения

Упражнение 8.1 *Напишите программу для нахождения минимума функции одной переменной $f(x)$ на интервале $[a,b]$ методом золотого сечения. С ее помощью найдите минимум функции $(x^2-6x+12)(x^2+6x+12)^{-1}$ на интервале $[0,20]$.*

Точка x^1 является точкой золотого сечения, если

$$\frac{b-a}{x^1-a} = \frac{x^1-a}{b-x^1}.$$

Исходя из этого определения имеем

$$x^1 = a + \frac{\sqrt{5}-1}{2}(b-a).$$

Аналогично для x^2 имеем

$$\frac{x^1-a}{x^2-a} = \frac{x^2-a}{x^1-x^2}$$

и поэтому

$$x^2 = a + \frac{3-\sqrt{5}}{2}(b-a).$$

На основе сравнения значений функции $f(x)$ в этих точках проводится итерационное уточнение интервала, на котором функция имеет минимум. Число

итераций n для достижения необходимой точности ϵ в определении точки минимума определяется равенством

$$|b-a|c^n = \epsilon, \quad c = \frac{\sqrt{5}-1}{2}.$$

В модуле golden функция golden() обеспечивает нахождение минимума функции одной переменной $f(x)$ на интервале $[a,b]$.

Модуль golden

```
import math as mt
def golden(f, a, b, tol=1.0e-10):
    """
    Golden section method for determining x
    that minimizes the scalar function f(x).
    The minimum must be bracketed in (a,b).
    """
    c1 = (mt.sqrt(5.) - 1.) / 2.
    c2 = 1. - c1
    nIt = int(mt.ceil(mt.log(tol / abs(b-a)) / mt.log(c1)))
    # First step
    x1 = c1*a + c2*b
    x2 = c2*a + c1*b
    f1 = f(x1)
    f2 = f(x2)
    # Iteration
    for i in range(nIt):
        if f1 > f2:
            a = x1
            x1 = x2
            f1 = f2
            x2 = c2*a + c1*b
            f2 = f(x2)
        else:
            b = x2
            x2 = x1
            f2 = f1
            x1 = c1*a + c2*b
            f1 = f(x1)
    if f1 < f2:
        return x1,f1
    else:
        return x2,f2
```

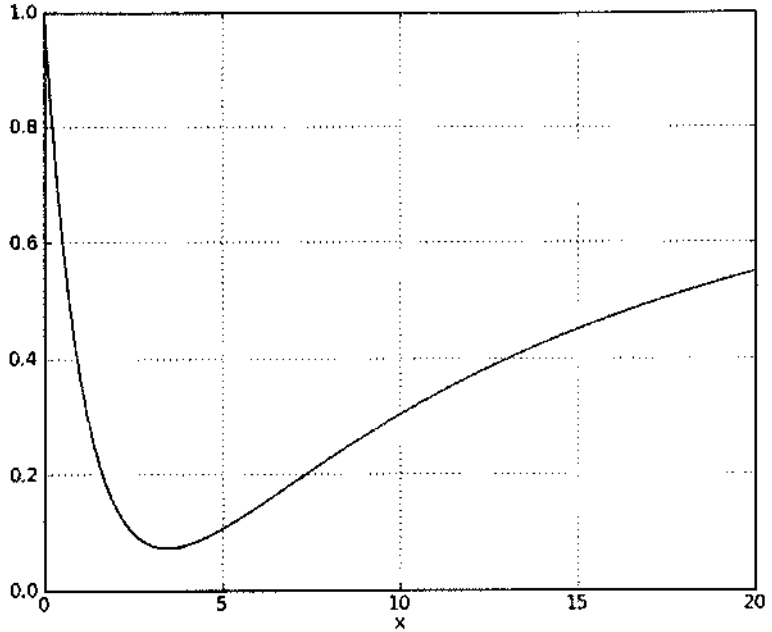

Рис. 8.1 График функции $y = (x^2 - 6x + 12)(x^2 + 6x + 12)^{-1}$

Для оценки интервала, на котором функция $f(x)$ достигает минимума, будем использовать график $y = f(x)$. С учетом этого для нахождения минимума функции $(x^2 - 6x + 12)(x^2 + 6x + 12)^{-1}$ используется следующая программа.

exer_8_1.py

```
import numpy as np
import matplotlib.pyplot as plt
from golden import golden
def f(x):
    return (x**2 - 6.*x + 12.) / (x**2 + 6.*x + 12.)
a = 0.
b = 20.
x = np.linspace(a, b, 200)
y = f(x)
plt.plot(x, y)
plt.xlabel('x')
plt.grid(True)
plt.show()
xMin, fMin = golden(f, a, b)
```

```
print 'xMin =', xMin
print 'fMin =', fMin
```

```
xMin = 3.46410163303
fMin = 0.0717967697245
```

На интервале |0,20| найдены одна точка минимума (см. рис. 8.1).

Упражнение 8.2 *Напишите программу для нахождения минимума функции нескольких переменной $f(x)$ градиентным методом при выборе итерационных параметров из минимума функции одной переменной, который находиться методом золотого сечения (модуль golden). Проиллюстрируйте работу программы при минимизации функции $10(x_2 - x_1^2)^2 + (1 - x_1)^2$.*

В модуле grad функция grad() находиться минимум функции многих переменных при аналитическом задании ее градиента.

Модуль grad

```
import numpy as np
import math as mt
from golden import golden
def grad(F, GradF, x, d=0.5, tol=1.e-10):
    """
    Gradient method for determining vector x
    that minimizes the function F(x),
    GradF(x) is function for grad(F),
    x is starting point.
    """
    # Line function along h
    def f(al):
        return F(x + al*h)
    gr0 = - GradF(x)
    h = gr0.copy()
    F0 = F(x)
    itMax = 500
    for i in range(itMax):
      # Minimization 1D function
      al, fMin = golden(f, 0, d)
      x = x + al*h
      F1 = F(x)
      gr1 = - GradF(x)
      if (mt.sqrt(np.dot(gr1,gr1)) <= tol) or (abs(F0 - F1) < tol):
          return x, i+1
      h = gr1
      gr0 = gr1.copy()
       F0 = F1
    print "Gradient method did not converge (500 iterations)"
```

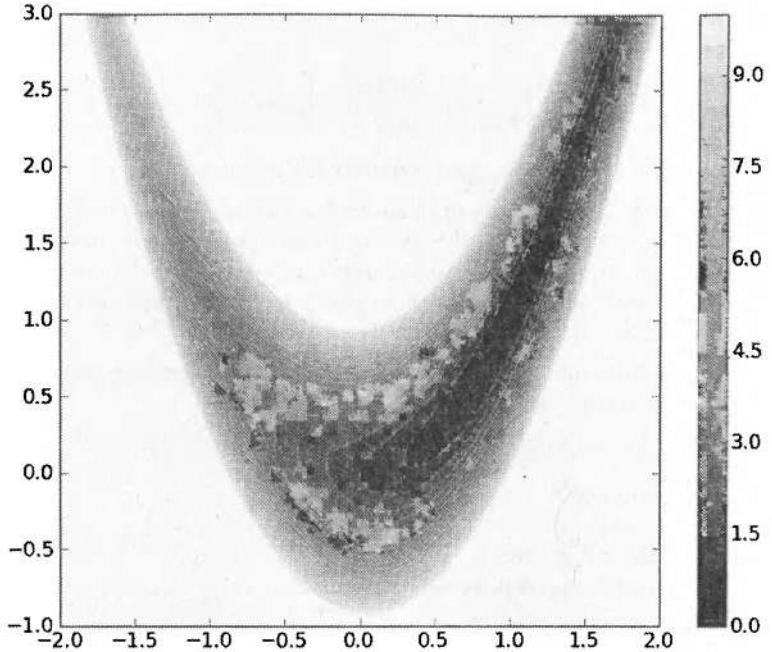

Рис. 8.2 Линии уровня функции $10(x_2 - x_1^2)^2 + (1 - x_1)^2$

Для оценки начального приближения используется график исследуемой функции $10(x_2 - x_1^2)^2 + (1 - x_1)^2$ с использованием следующей программы.

exer − 8 2.py

```
import numpy as np
import matplotlib.pyplot as plt
from grad import grad
def F(x):
    return 10.*(x[1] - x[0]**2)**2 + (1 - x[0])**2
def GradF(x):
    gr = np.zeros((2),'float')
    gr[0] = -40.*(x[1] - x[0]**2)*x[0] - 2.*(1 - x[0])
    gr[1] = 20.*(x[1] - x[0]**2)
    return gr
# graph of function
x = np.linspace(-2., 2., 101)
y = np.linspace(-1., 3., 101)
X, Y = np.meshgrid(x, y)
z = F([X, Y])
```

```
v = np.linspace(0., 10., 21)
plt.contourf(x, y, z, v, cmap=plt.cm.gray)
plt.colorbar()
plt.show()
# minimum function
x0 = np.array([0., 0.1])
xMin, nIt = grad(F, GradF, x0)
print 'xMin:', xMin
print 'Number of iterations = ', nIt
```

```
xMin   [ 0.99995383  0.99990305]
Number of iterations =  372
```

Функция имеет выраженный овражный характер (рис. 8.2), что обуславливает достаточно медленную сходимость градиентного метода.

8.4 Задачи

Задача 8.1 *Напишите программу для нахождения минимума функции одной переменной $f(x)$ на интервале $[a, b]$ с использованием интерполяционного полинома второго порядка. С помощью этой программы найдите два первых локальных минимума функции $x^2 - 50\sin(x)$ при $x \geq 0$.*

Задача 8.2 *Напишите программу для нахождения локального минимума функции одной переменной $f(x)$ методом Ньютона. С ее помощью найдите минимум функции $2x - 1 + 2\cos(\pi x)$ на интервале $[0, 2]$.*

Задача 8.3 *Напишите программу для нахождения локального минимума функции многих переменных $f(x)$ методом Ньютона. Продемонстрируйте работоспособность программы на задаче нахождения минимума функции $x_1^3 + x_2^3 - 3x_1x_2$, $x_\alpha > 0$, $\alpha = 1, 2$ при различных начальных приближениях.*

Упражнение 8.3 *Напишите программу для нахождения минимума функции одной переменной $f(x)$ на интервале $[a, b]$ при ограничениях $g(x) \geq 0$ методом штрафа с использованием метода золотого сечения для решения задачи безусловной минимизации. С ее помощью найдите минимум функции $e^{-x}x$ при $x \geq 2$.*

Интерполирование и приближение функций

Рассматриваются задачи приближенного восстановления значений функции одной переменной по ее значениям в некоторых точках. Традиционный подход для одномерной интерполяции связан с построением алгебраических многочленов, принимающих заданные значения в точках интерполяции. Более перспективными являются подходы с использованием кусочно-гладких полиномов. Отдельно выделены задачи приближения функций в нормированных пространствах.

Основные обозначения

$f(x)$	—	интерполируемая функция
$x_0 < x_1 < \ldots < x_n$	—	узлы интерполирования
$\{\varphi_i(x)\}_{i=0}^{n}$	—	система Чебышева
$\varphi(x)$	—	обобщенный интерполяционный многочлен
$L_n(x)$	—	интерполяционный многочлен n-го порядка
$f(x_i, x_{i+1})$	—	разделенная разность первого порядка
$f(x_i, x_{i+1}, \ldots, x_{i+k})$	—	разделенная разность k-го порядка
$S_m(x)$	—	интерполяционный сплайн m-го порядка

9.1 Задачи интерполяции и приближения функций

Задача интерполяции ставится следующим образом. Пусть на отрезке $[a, b]$ в узлах интерполирования $x_0 < x_1 < \ldots < x_n$ известны значения функции $y_i = f(x_i)$, $i = 0, 1, \ldots, n$. Необходимо найти значения функции $f(x)$ в точках $x \neq x_i$, $i = 0, 1, \ldots, n$.

Пусть на отрезке $[a,b]$ задана система функций $\{\varphi_i(x)\}_{i=0}^n$ и определим

$$\varphi(x) = \sum_{i=0}^{n} c_i \varphi_i(x) \tag{9.1}$$

с действительными коэффициентами $c_i, i = 0,1,\ldots,n$. При интерполировании функции $f(x)$ для нахождения коэффициентов используются условия

$$\varphi(x_i) = f(x_i), \quad i = 0,1,\ldots,n. \tag{9.2}$$

В частном случае алгебраической интерполяции $\varphi_i(x) = x^i$, $i = 0,1,\ldots,n$.

При интерполяции сплайнами функция $f(x)$ приближается многочленами невысокой степени на частичных отрезках $[x_i, x_{i+1}]$, где $i = 0,1,\ldots,n-1$.

Рассматривается также задача построения обобщенного многочлена $\varphi(x)$ приближающего заданную функцию $f(x)$. В линейном нормированном пространстве коэффициенты обобщенного многочлена $\varphi(x)$ определяются из условия минимальности нормы погрешности интерполирования:

$$\|f(x) - \sum_{i=0}^{n} c_i \varphi_i(x)\|. \tag{9.3}$$

Аналогично ставится задачи интерполяции и приближения многомерных функций.

9.2 Алгоритмы интерполяции и приближения функций

Для одномерных функций задачи интерполяции решаются с использованием алгебраических многочленов Лагранжа и Ньютона, параболических и кубических сплайнов, рассмотрена задача наилучшего приближения в гильбертовом пространстве.

Полиномиальная интерполяция

При аппроксимации полиномами используются функции

$$\varphi_i(x) = x^i, \quad i = 0,1,\ldots,n$$

и интерполяционный многочлен (см. (9.1)) имеет вид

$$\varphi(x) = L_n(x) = \sum_{i=0}^{n} c_i x^i.$$

Интерполяционный многочлен Лагранжа записывается в виде

$$L_n(x) = \sum_{i=0}^{n} \frac{\omega(x)}{(x - x_i)\omega'(x_i)} f(x_i), \tag{9.4}$$

где $\omega(x)$ — многочлен степени $n+1$:

$$\omega(x) = \prod_{i=0}^{n}(x - x_i),$$

а

$$\omega'(x) \equiv \frac{d\omega}{dx}(x).$$

Можно использовать другую запись интерполяционного многочлена в виде интерполяционного многочлена Ньютона, которая строится с помощью разделенных разностей. Разделенной разностью первого порядка называется отношение

$$f(x_i, x_{i+1}) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}.$$

Разделенная разность k-го порядка определяется по рекуррентной формуле

$$f(x_i, x_{i+1}, \ldots, x_{i+k}) =$$

$$= \frac{f(x_{i+1}, x_{i+2}, \ldots, x_{i+k}) - f(x_i, x_{i+1}, \ldots, x_{i+k-1})}{x_{i+k} - x_i}.$$

С использованием таких обозначений получим

$$L_n(x) = f(x_0) + (x - x_0)f(x_0, x_1) +$$

$$+(x - x_0)(x - x_1)f(x_0, x_1, x_2) + \ldots$$

$$\ldots + (x - x_0)(x - x_1)\cdots(x - x_{n-1})f(x_0, x_1, \ldots, x_n). \qquad (9.5)$$

Интерполяционные сплайны

Пусть функция $f(x)$ задана в узлах $a = x_0 < x_1 < \ldots < x_n = b$. Интерполяционный сплайн $S_m(x)$ порядка m определяется из условий:

1. на каждом отрезке $[x_i, x_{i+1}], i = 0, 1, \ldots, n-1$ $S_m(x)$ является полиномом степени m;
2. на всем отрезке $[a, b]$ $S_m(x)$ имеет непрерывные производные до порядка $m-1$;
3. в узлах интерполяции

$$S_m(x_i) = f(x_i), \quad i = 0, 1, \ldots, n.$$

При $m \geq 2$ единственность $S_m(x)$ обеспечивается $m-1$ дополнительными условиями. Обычно эти условия формулируются на концах отрезка интерполяции $[a, b]$.

Интерполяционный кубический сплайн $S_3(x)$ на отрезке $[x_i, x_{i+1}]$ задается полиномом третьей степени:

$$S_3^{(i)} = a_i + b_i(x - x_i) + \frac{c_i}{2}(x - x_i)^2 + \frac{d_i}{6}(x - x_i)^3,$$

$$x_i \le x \le x_{i+1}, \quad i = 0, 1, \ldots, n-1, \tag{9.6}$$

причем

$$a_i = S_3^{(i)}(x_i), \quad b_i = \frac{dS_3^{(i)}}{dx}(x_i),$$

$$c_i = \frac{d^2 S_3^{(i)}}{dx^2}(x_i), \quad d_i = \frac{d^3 S_3^{(i)}}{dx^3}(x_i).$$

По определению для $S_3(x)$ выполнены условия:

$$S_3^{(i)}(x_i) = f(x_i), \quad i = 0, 1, \ldots, n-1,$$

$$S_3^{(i)}(x_{i+1}) = f(x_{i+1}), \quad i = 0, 1, \ldots, n-1,$$

$$\frac{dS_3^{(i)}}{dx}(x_{i+1}) = \frac{dS_3^{(i+1)}}{dx}(x_{i+1}), \quad i = 0, 1, \ldots, n-2,$$

$$\frac{d^2 S_3^{(i)}}{dx^2}(x_{i+1}) = \frac{d^2 S_3^{(i+1)}}{dx^2}(x_{i+1}), \quad i = 0, 1, \ldots, n-2.$$

Два дополнительных условия можно взять в виде (естественные кубические сплайны)

$$\frac{d^2 S_3^{(0)}}{dx^2}(x_0) = 0, \quad \frac{d^2 S_3^{(n-1)}}{dx^2}(x_n) = 0.$$

Приближение функций в нормированном пространстве

Пусть H — вещественное гильбертово пространство со скалярным произведением (f, g) и нормой $\|f\| = (f, f)^{1/2}$. В случае $H = L_2(a, b)$ имеем

$$(f, g) = \int_a^b f(x)g(x)dx, \quad \|f\| = \left(\int_a^b |f(x)|^2 dx\right)^{1/2}$$

В задаче о наилучшем приближении по системе функций

$$\varphi_i(x) \in H, \quad i = 0, 1, \ldots, n$$

строится обобщенный многочлен (9.1) (элемент наилучшего приближения), который для заданной приближаемой функции $f(x) \in H$ минимизирует норму отклонения (9.3).

Коэффициенты элемента наилучшего приближения находятся из решения следующей системы линейных уравнений:

$$\sum_{j=0}^{n} c_j(\varphi_i, \varphi_j) = (f, \varphi_i), \quad i = 0, 1, \ldots, n. \tag{9.7}$$

9.3 Упражнения

Упражнение 9.1 *Напишите программу для интерполирования данных на основе интерполяционного многочлена Ньютона. С помощью этой программы интерполируйте данные в равноотстоящих узлах для функции Рунге* $f(x) = (1 + 25x^2)^{-1}$ *на интервале [-1,1] при* $n = 4, 6, 10$.

В модуле interpolation функция `coef()` предназначена для вычисления коэффициентов интерполяционного многочлена Ньютона, а функция `interpolation()` — для вычисления значения интерполяционного полинома в заданной точке.

Модуль `interpolation`

```
def interpolation(c, x, x0):
    """
    Evaluates Newton's polynomial at x0.
    """
    # Degree of polynomial
    n = len(x) - 1
    y0 = c[n]
    for k in range(1,n+1):
        y0 = c[n-k] + (x0 - x[n-k]) * y0
    return y0
def coef(x, y):
    """
    Computes the coefficients of Newton's polynomial.
    """
    # Number of data points
    m = len(x)
    c = y.copy()
    for k in range(1,m):
        c[k:m] = (c[k:m] - c[k-1])/(x[k:m] - x[k-1])
    return c
```

Решение задачи полиномиальной интерполяция для функции Рунге дается следующей программой.

`exer-9.1.py`

```
import numpy as np
```

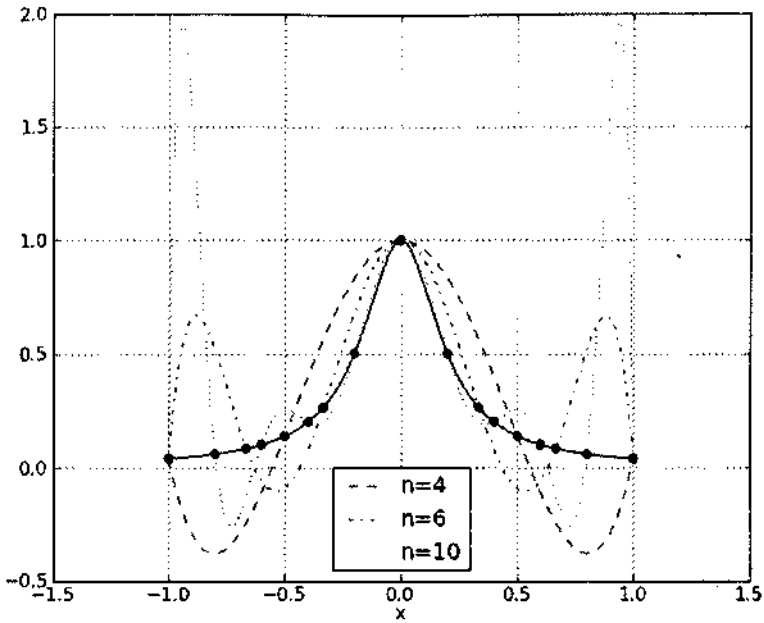

Рис. 9.1 Интерполяция функции Рунге $f(x) = (1 + 25x^2)^{-1}$

```
import matplotlib.pyplot as plt
from interpolation import interpolation, coef
def f(x):
    return 1. / (1 + 25*x**2)
a = -1.
b = 1.
x1 = np.linspace(a, b, 200)
y1 = f(x1)
plt.plot(x1, y1)
nList = [4, 6, 10]
sglist = ['--', '-.', ':']
for k in range(len(nList)):
    n = nList[k]
    x = np.linspace(a, b, n+1)
    y = f(x)
    plt.scatter(x, y, marker='o')
    c = coef(x, y)
    y1 = interpolation(c, x, x1)
    sl = 'n=' + str(n)
```

```
    sg = sglist[k]
    plt.plot(x1, y1, sg, label=sl)
plt.xlabel('x')
plt.grid(True)
plt.legend(loc=0)
plt.show()
```

Расчетные данные, приведенные на рис. 9.1, демонстрируют недостатки полиномиальной аппроксимации при больших n.

Упражнение 9.2 *Напишите программу для интерполирования данных на основе естественного интерполяционного кубического сплайна. С помощью этой программы интерполируйте данные в равноотстоящих узлах для функции Рунге* $f(x) = (1 + 25x^2)^{-1}$ *на интервале* $[-1, 1]$ *при* $n = 4, 6, 10$.

Сначала получим расчетные формулы для коэффициентов естественного кубического сплайна.

Введем обозначения

$$h_i = x_i - x_{i-1}, \quad i = 1, 2, \ldots, n.$$

Для кубического сплайна $S_3(x)$ с учетом представления (9.6) получим следующую систему уравнений:

$$a_i = f(x_i), \quad i = 0, 1, \ldots, n-1, \tag{9.8}$$

$$a_i + b_i h_{i+1} + \frac{c_i}{2} h_{i+1}^2 + \frac{d_i}{6} h_{i+1}^3 = f(x_{i+1}), \quad i = 0, 1, \ldots, n-1, \tag{9.9}$$

$$b_i + c_i h_{i+1} + \frac{d_i}{2} h_{i+1}^2 = b_{i+1}, \quad i = 0, 1, \ldots, n-2, \tag{9.10}$$

$$c_i + d_i h_{i+1} = c_{i+1}, \quad i = 0, 1, \ldots, n-2, \tag{9.11}$$

$$c_0 = 0, \quad c_{n-1} + d_{n-1} h_n = 0. \tag{9.12}$$

Формально доопределим $c_n = 0$, тогда из (9.11) и второго условия (9.12) получим

$$d_i = \frac{c_{i+1} - c_i}{h_{i+1}}, \quad i = 0, 1, \ldots, n-1, \tag{9.13}$$

а вместо (9.12) будем иметь

$$c_0 = 0, \quad c_n = 0. \tag{9.14}$$

Подстановка (9.8), (9.13) в (9.9) даст следующее представление для коэффициентов b_i:

$$b_i = \frac{f(x_{i+1}) - f(x_i)}{h_{i+1}} - \frac{h_{i+1}}{6}(c_{i+1} + 2c_i), \tag{9.15}$$

$$i = 1, 2, \ldots, n-1.$$

С учетом (9.13), (9.15) соотношения (9.10) приводят к уравнению

$$c_{i-1}h_i + 2c_i(h_i + h_{i+1}) + c_{i+1}h_{i+1} =$$

$$= 6\left(\frac{f(x_{i+1}) - f(x_i)}{h_{i+1}} - \frac{f(x_i) - f(x_{i-1})}{h_i}\right), \qquad (9.16)$$

$$i = 1, 2, \ldots, n-1.$$

Тем самым приходим к линейной системе уравнений (9.14), (9.16) с трехдиагональной матрицей с диагональным преобладанием. Решение этой системы всегда существует и единственно. Другие коэффициенты сплайна определяются в соответствии с (9.8), (9.13), (9.15).

В модуле spline функция `coefspline()` предназначена для вычисления коэффициентов интерполяционного кубического сплайна, а функция `spline()` – для вычисления значения кубического сплайна в заданной точке. Для нахождения коэффициентов сплайна используется функция `solveLU3()` (решение системы уравнений с трехдиагональной матрицей) из модуля lu3.

Модуль spline

```
import numpy as np
from lu3 import solveLU3
def spline(x, y, c, x0):
    """
    Evaluates cubic spline at x0.
    """
    def find(x, x0):
        """
        Find the segment spanning x0
        """
        iL = 0
        iR = len(x) - 1
        while iR - iL > 1:
            i = (iL + iR) / 2
            if x0 < x[i]:
                iR = i
            else:
                iL = i
        return iL
    i = find(x, x0)
    h = x[i+1] - x[i]
    y0 = ((x[i+1] - x0)**3/h - (x[i+1] - x0)*h) * c[i] / 6. \
       + ((x0 - x[i])**3/h - (x0 - x[i])*h) * c[i+1] / 6.   \
       + (y[i]*(x[i+1] - x0) + y[i+1]*(x0 - x[i])) / h
    return y0
def coefspline(x, y):
```

```
    """
    Computes the coefficients of cubic spline.
    """
    n = len(x) - 1
    a = np.ones((n+1), 'float')
    b = np.zeros((n+1), 'float')
    c = np.zeros((n+1), 'float')
    f = np.zeros((n+1), 'float')
    for i in range(1, n):
        a[i] = 2.*(x[i+1] - x[i-1])
        b[i] = x[i+1] - x[i]
        c[i] = x[i] - x[i-1]
        f[i] = 6.*(y[i+1] - y[i]) / (x[i+1] - x[i]) \
             - 6.*(y[i] - y[i-1]) / (x[i] - x[i-1])
    return solveLU3(a, b, c, f)
```

Решение задачи интерполяции сплайнами для функции Рунге дается следующей программой.

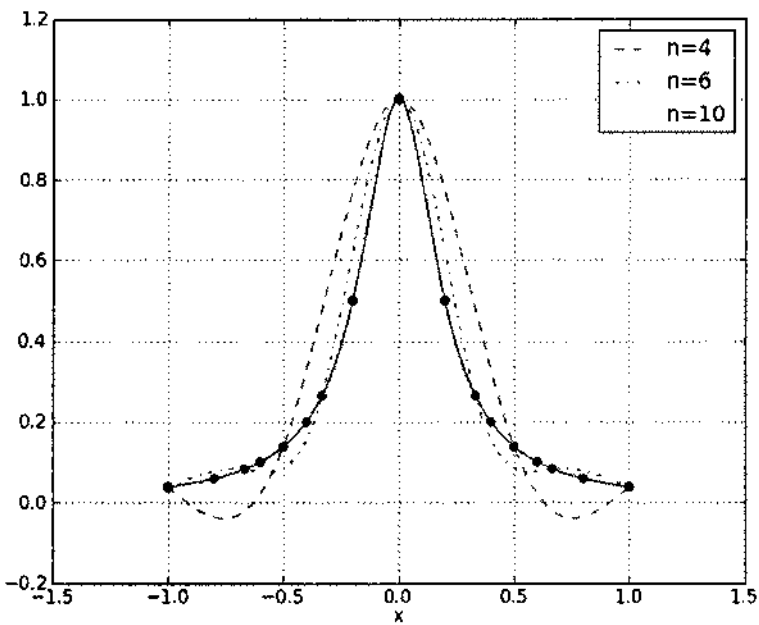

Рис. 9.2 Сплайн-интерполяция функции Рунге $f(x) = (1 + 25x^2)^{-1}$

exer −9 2 py

```
import numpy as np
import matplotlib.pyplot as plt
from spline import spline, coefspline
def f(x):
    return 1. / (1 + 25*x**2)
a = -1.
b = 1.
m = 200
x1 = np.linspace(a, b, m)
y1 = f(x1)
plt.plot(x1, y1)
nList = [4, 6, 10]
sglist = ['--', '-.', ':']
for k in range(len(nList)):
    n = nList[k]
    x = np.linspace(a, b, n+1)
    y = f(x)
    plt.scatter(x, y, marker='o')
    c = coefspline(x, y)
    for i in range(m):
        y1[i] = spline(x, y, c, x1[i])
    sl = 'n=' + str(n)
    sg = sglist[k]
    plt.plot(x1, y1, sg, label=sl)
plt.xlabel('x')
plt.grid(True)
plt.legend(loc=0)
plt.show()
```

Преимущества интерполяции сплайнами иллюстрируются рис. 9.2.

9.4 Задачи

Задача 9.1 *Напишите программу для нахождения значения интерполирующего полинома в точке x на основе рекуррентных соотношений (алгоритм Невилля):*

$$p_{ii}(x) = y_i, \quad 0 \le i \le n,$$

$$p_{ij}(x) = \frac{(x - x_j)p_{i,j-1}(x) + (x_i - x)p_{i-1,j}(x)}{x_i - x_j}, \quad 0 \le i < j \le n.$$

С помощью этой программы найдите значения в точках $x = 1.5, 2.5$ при интерполировании функции $f(x) = (\arctan(1+x^2))^{-1}$ при использовании равноотстоящих узлов на интервале $[-3, 3]$ при $n = 4, 6, 10$.

Задача 9.2 *Напишите программу для интерполирования данных на основе кусочно-линейного восполнения (интерполяционного линейного сплайна). С помощью этой программы интерполируйте данные в равноотстоящих узлах для функции* $f(x) = (\arctan(1 + x^2))^{-1}$ *на интервале* $[-3, 3]$ *при* $n = 4, 6, 10$.

Задача 9.3 *Напишите программу для приближения сеточной функции* y_i, $i = 0, 1, ..., n$ *полиномом* $p_m(x) = c_0 + c_1 x + ... + c_m x^m$, $m \leq n$ *методом наименьших квадратов. Работоспособность программы проиллюстрируйте на примере аппроксимации*

$$y_i = 1 - \cos(x_i),$$

$$x_i = ih, \quad h = 1/n, \quad i = 0, 1, ..., n$$

для $n = 10$ *и* $m = 1, 2, 3$.

Задача 9.4 *Напишите программу для приближения сеточной функции* y_i, $i = 0, 1, ..., n$ *функцией* $\xi(x) = ae^{bx}$ *методом наименьших квадратов. С помощью этой программы аппроксимируйте сеточную функцию*

$$y_i = 1 - \cos(x_i),$$

$$x_i = ih, \quad h = 1/n, \quad i = 0, 1, ..., n$$

для $n = 10$.

10

Численное интегрирование

Задача приближенного интегрирования состоит в вычислении определенного интеграла по значениям подынтегральной функции в отдельных точках. Рассматриваются классические квадратурные формулы прямоугольников, трапеций и Симпсона. Проводится рассмотрение формул интегрирования при заданных узлах квадратурной формулы. В более общем случае проводится оптимизация квадратурных формул за счет выбора узлов.

Основные обозначения

$f(x)$	—	подынтегральная функция
$\varrho(x)$	—	весовая функция
$x_0 < x_1 < \ldots < x_n$	—	узлы квадратурной формулы
$c_i,\ i = 0, 1, \ldots, n$	—	коэффициенты квадратурной формулы
$L_n(x)$	—	интерполяционный многочлен n-го порядка

10.1 Задачи приближенного вычисления интегралов

Рассматривается задача приближенного вычисления определенных интегралов

$$\int_a^b \varrho(x) f(x) dx \tag{10.1}$$

на некотором классе функций $f(x)$ с заданной весовой функцией $\varrho(x)$.

С этой целью подынтегральная функция задается в отдельных точках x_i отрезка $[a, b]$, $i = 0, 1, \ldots, n$. Под квадратурной формулой понимается приближенное равенство

$$\int_a^b \varrho(x) f(x) dx \approx \sum_{i=0}^{n} c_i f(x_i), \tag{10.2}$$

в котором c_i, $i = 0, \ldots, n$ — коэффициенты квадратурной формулы. Через

$$\psi_n = \int_a^b \varrho(x) f(x) dx - \sum_{i=0}^{n} c_i f(x_i),$$

определим погрешность квадратурной формулы.

Минимизация погрешности (увеличение точности) квадратурной формулы на выбранном классе функций может достигается прежде всего за счет выбора коэффициентов квадратурной формулы а также за счет выбора узлов интегрирования.

10.2 Алгоритмы приближенного вычисления интегралов

Рассматриваются простейшие квадратурные формулы прямоугольников, трапеций и Симпсона для приближенного вычисления определенных интегралов. Строятся квадратурные формулы интерполяционного типа с заданными узлами квадратурной формулы, обсуждаются вопросы оптимизации квадратурные формулы за счет выбора узлов.

Классические квадратурные формулы составного типа

Будем рассматривать задачу вычисления интеграла

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

т.е. весовая функция $\varrho(x) = 1$. Представим интеграл в виде суммы по частичным отрезкам:

$$\int_a^b f(x) dx = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x) dx.$$

Приведем некоторые простейшие квадратурные формулы для приближенного вычисления интеграла на частичном отрезке $[x_{i-1}, x_i]$.

Формула

$$\int_{x_{i-1}}^{x_i} f(x) dx \approx f(x_{i-1/2})(x_i - x_{i-1}) \tag{10.3}$$

называется формулой прямоугольников на частичном отрезке $[x_{i-1}, x_i]$, где $x_{i-1/2} = (x_{i-1} + x_i)/2$.

Для случая равномерной сетки

$$\omega = \{x \mid x = x_i = a + ih,\ i = 0, 1, \ldots, n,\ nh = b - a\}$$

суммирование (10.1) по i от 1 до N даст составную формулу прямоугольников

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_{i-1/2})h.$$

При использовании значений интегрируемой функции в узлах простейшей является квадратурная формула трапеций. В этом случае

$$\int_{x_{i-1}}^{x_i} f(x)dx \approx \frac{f(x_{i-1}) + f(x_i)}{2}(x_i - x_{i-1}), \tag{10.4}$$

а для всего отрезка при использовании равномерной сетки получим

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \frac{f(x_{i-1}) + f(x_i)}{2}h.$$

К формуле трапеций мы приходим при замене подынтегральной функции $f(x)$ кусочно-линейной функцией, которая проходит через точки $(x_{i-1}, f(x_{i-1}))$, $(x_i, f(x_i))$. При интерполировании подынтегральной функции на частичном отрезке $[x_{i-1}, x_i]$ с использование трех точек $(x_{i-1}, f(x_{i-1}))$, $(x_{i-\frac{1}{2}}, f(x_{i-1/2}))$ и $(x_i, f(x_i))$ получим квадратурную формулу Симпсона частичного отрезка:

$$\int_{x_{i-1}}^{x_i} f(x)dx \approx \frac{f(x_{i-1}) + 4f(x_{i-1/2}) + f(x_i)}{6}(x_i - x_{i-1}). \tag{10.5}$$

На всем отрезке соответствующая квадратурная формула составного типа имеет вид

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \frac{f(x_{i-1}) + 4f(x_{i-1/2}) + f(x_i)}{6}h.$$

Квадратурные формулы интерполяционного типа

Приведенные выше квадратурные формулы построены на основе разбиения отрезка интегрирования $[a, b]$ на частичные отрезки $[x_{i-1}, x_i]$, $i = 1, 2, \ldots, n$ и использованию некоторых простейших интерполяций для подынтегральной функции на этих отрезках. Можно ориентироваться на применение квадратурные формул интерполяционного типа, когда подынтегральная функция заменяется интерполяционным многочленом сразу на всем отрезке интегрирования $[a, b]$.

Подынтегральная функция $f(x)$ в (10.1) заменяется интерполяционным многочленом Лагранжа

$$L_n(x) = \sum_{i=0}^{n} \frac{\omega(x)}{(x - x_i)\omega'(x_i)} f(x_i),$$

где

$$\omega(x) = \prod_{i=0}^{n}(x - x_i), \quad \omega'(x) \equiv \frac{d\omega}{dx}(x).$$

Для коэффициентов квадратурной формулы (10.2) получим представление

$$c_i = \int_a^b \varrho(x) \frac{\omega(x)}{(x - x_i)\omega'(x_i)} dx, \quad i = 0, 1, \ldots, n. \tag{10.6}$$

Квадратурные формулы Гаусса

Для повышения точности квадратурной формулы можно оптимизировать выбор не только коэффициентов квадратурной формулы c_i, $i = 0, 1, \ldots, n$, но и узлов интерполяции x_i, $i = 0, 1, \ldots, n$. Квадратурные формулы интерполяционного типа (10.2), (10.6) являются точными для алгебраических полиномов степени n. За счет выбора узлов интерполирования строятся квадратурные формулы наивысшей алгебраической степени точности (квадратурные формулы Гаусса), которые точны для любого алгебраического многочлена степени $2n + 1$.

Потребуем, чтобы квадратурная формула (10.1) была точна для любого алгебраического многочлена степени m. Это означает, что формула точна для функций $f(x) = x^\alpha$, $\alpha = 0, 1, \ldots, m$:

$$\int_a^b \varrho(x) x^\alpha dx = \sum_{i=0}^{n} c_i x_i^\alpha, \quad i = 0, 1, \ldots, m. \tag{10.7}$$

Для определения $2n + 2$ неизвестных c_i, x_i, $i = 0, 1, \ldots, n$ имеем нелинейную систему (10.7) $m + 1$ уравнений.

Для знакопостоянной весовой функции $\varrho(x)$ система уравнений (10.7) при $m = 2n+1$ имеет единственное решение. При этом квадратурная формула является интерполяционной, т.е. коэффициенты вычисляются согласно (10.6), а узлы должны быть такими, чтобы многочлен

$$\omega(x) = \prod_{i=0}^{n} (x - x_i)$$

был ортогонален с весом $\varrho(x)$ любому многочлену $q(x)$ степени меньше $n+1$:

$$\int_a^b \varrho(x)\omega(x)q(x)dx = 0.$$

10.3 Упражнения

Упражнение 10.1 *Напишите программу для приближенного вычисления интеграла от функции $f(x)$ на интервале $[a, b]$ с использованием квадратурной формулы трапеций на основе рекуррентного уточнения при увеличении числа частичных отрезков в два раза. С помощью этой программы*

рассчитайте значение интеграла ошибок

$$I(x) = \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

при $x = 1$ *с различной точностью.*

Пусть имеется 2^n частичных отрезков длиной $h = (b-a)/2^n$. При $n = 0$ имеем

$$I_0 = (f(a) + f(b))\frac{h}{2},$$

при $n = 1$ —

$$I_1 = (f(a) + f(b))\frac{h}{2} + f(a+h))h = \frac{1}{2}I_0 + f(a+h))h.$$

В общем случае имеет место рекуррентное соотношение

$$I_{n+1} = \frac{1}{2}I_n + \sum_{i=1}^{2^{n-1}} f(a + (2i-1)h)h.$$

Тем самым при увеличении числа частичных отрезков вычисления проводятся только в новых точках.

В модуле trap функция `trap()` вычисляет приближенное значение интеграла по рекуррентной квадратурной формуле трапеций.

Модуль `trap`

```
def trap(f, a, b, tol=1.e-6):
    """
    Integral of f(x) from a to b computed by trapezoidal rule
    """
    h = b - a
    Iold = h * (f(a) + f(b)) / 2.
    m = 1
    kMax = 25
    for k in range(1, kMax):
        x = a + h / 2.
        sum = 0.
        for i in range(m):
            sum = sum + f(x)
            x = x + h
        Inew = (Iold + h*sum) /2
        if (k > 1) and (abs(Inew - Iold) < tol): break
        Iold = Inew
        m = m * 2
        h = h / 2.
    return Inew
```

Решение задачи приближенного вычисления интеграла ошибок дается следующей программой.

exer_10_1.py

```
import math as mt
from trap import trap
def f(x):
    return 2.*mt.exp(-x**2) / mt.sqrt(mt.pi)
a = 0.
b = 1.
for n in range(5, 11):
    er = 10.**(-n)
    Inew = trap(f, a, b, tol=er)
    print 'tol =', er, 'Integral =', Inew
```

```
tol = 1e-05 Integral = 0.842699737276
tol = 1e-06 Integral = 0.842700529031
tol = 1e-07 Integral = 0.842700776455
tol = 1e-08 Integral = 0.842700791919
tol = 1e-09 Integral = 0.842700792692
tol = 1e-10 Integral = 0.842700792934
```

Точность вычислений контролируется параметром *tol*, при его уменьшении точность вычислений повышается ($\mathrm{erf}(x) = 0.8427007929497...$).

Упражнение 10.2 *Напишите программу для приближенного вычисления интеграла от функции $f(x)$ на интервале $[a, b]$ с использованием квадратурной формулы Гаусса с весом $\rho(x)$ на основе табличного задания узлов и коэффициентов. Используйте эту программу для вычисления интеграла*

$$I = \int_0^1 \frac{\ln(x)}{1-x} dx$$

при различном числе узлов и сравните приближенное значение интеграла с точным.

Узлы и коэффициенты для рассматриваемой квадратурной формулы Гаусса–Лежандра даны в табл. 10.1.

В модуле gauss функция gauss() вычисляет приближенное значение интеграла по этим табличным данным.

Модуль gauss

```
import numpy as np
def gauss(f, a, b, n):
    """
    Integral of f(x) from a to b computed by
    Gauss-Legendre quadrature using n nodes.
```

Таблица 10.1 *Квадратурная формула Гаусса—Лежандра*

n	узлы x_i	коэффициенты c_i
2	± 0.57735027	1.
3	± 0.77459667	0.55555556
	0.	0.88888889
4	± 0.86113631	0.34785485
	± 0.33998104	0.65214515
5	± 0.90617985	0.23692689
	± 0.53846931	0.47862867
	0.	0.56888889
6	± 0.93246951	0.17132449
	± 0.66120939	0.36076157
	± 0.23861919	0.46791393
7	± 0.94910791	0.12948497
	± 0.74153119	0.27970539
	± 0.40584515	0.38183005
	0.	0.41795918
8	± 0.96028986	0.10122854
	± 0.79666648	0.22238103
	± 0.52553241	0.31370665
	± 0.18343464	0.36268378

```
    """
    if n > 8 or n < 2:
        print 'The number of nodes must be greater than 2
               and less than 8'
        return 0
    x = np.zeros(n, 'float')
    c = np.zeros(n, 'float')
    if n == 2:
        x[0] = 0.57735027
        x[1] = - x[0]
        c[0] = 1.
        c[1] = c[0]
    if n == 3:
        x[0] = 0.77459667
        x[1] = - x[0]
        x[2] = 0.
        c[0] = 0.55555556
        c[1] = c[0]
        c[2] = 0.88888889
    if n == 4:
        x[0] = 0.86113631
        x[1] = - x[0]
        x[2] = 0.33998104
        x[3] = - x[2]
```

```
        c[0] = 0.34785485
        c[1] = c[0]
        c[2] = 0.65214515
        c[3] = c[2]
    if n == 5:
        x[0] = 0.90617985
        x[1] = - x[0]
        x[2] = 0.53846931
        x[3] = - x[2]
        x[4] = 0.
        c[0] = 0.23692689
        c[1] = c[0]
        c[2] = 0.47862867
        c[3] = c[2]
        c[4] = 0.56888889
    if n == 6:
        x[0] = 0.93246951
        x[1] = - x[0]
        x[2] = 0.66120939
        x[3] = - x[2]
        x[4] = 0.23861919
        x[5] = - x[4]
        c[0] = 0.17132449
        c[1] = c[0]
        c[2] = 0.36076157
        c[3] = c[2]
        c[4] = 0.46791393
        c[5] = c[4]
    if n == 7:
        x[0] = 0.94910791
        x[1] = - x[0]
        x[2] = 0.74153119
        x[3] = - x[2]
        x[4] = 0.40584515
        x[5] = - x[4]
        x[6] = 0.
        c[0] = 0.12948497
        c[1] = c[0]
        c[2] = 0.27970539
        c[3] = c[2]
        c[4] = 0.38183005
        c[5] = c[4]
        c[6] = 0.41795918
    if n == 8:
        x[0] = 0.96028986
```

```
        x[1] = - x[0]
        x[2] = 0.79666648
        x[3] = - x[2]
        x[4] = 0.52553241
        x[5] = - x[4]
        x[6] = 0.18343464
        x[7] = - x[6]
        c[0] = 0.10122854
        c[1] = c[0]
        c[2] = 0.22238103
        c[3] = c[2]
        c[4] = 0.31370665
        c[5] = c[4]
        c[6] = 0.36268378
        c[7] = c[6]
    c1 = (b + a)/2.
    c2 = (b - a)/2.
    sum = 0.
    for i in range(n):
        sum = sum + c[i]*f(c1 + c2*x[i])
    return c2*sum
```

Решение задачи приближенного вычисления рассматриваемого интеграла дается следующей программой.

exer-10.2.py

```
import math as mt
from gauss import gauss
def f(x):
    return - mt.log(x) / (1. - x)
a = 0.
b = 1.
for n in range(2, 9):
    I = gauss(f, a, b, n)
    print 'n =', n, 'Integral =', I
Iexact = mt.pi**2 / 6.
print 'Exact value =', Iexact
```

```
n = 2 Integral = 1.54712020996
n = 3 Integral = 1.59426085346
n = 4 Integral = 1.61407794115
n = 5 Integral = 1.62420811093
n = 6 Integral = 1.63006541632
n = 7 Integral = 1.6337521364
n = 8 Integral = 1.63622112467
Exact value = 1.64493406685
```

Относительно невысокая точность вычислений связана с особенностью подынтегральной функции при $x \to 0$.

10.4 Задачи

Задача 10.1 *Напишите программу для приближенного вычисления интеграла от функции $f(x)$ на интервале $[a, b]$ с использованием составной квадратурной формулы Симпсона. Продемонстрируйте работоспособность программы при вычислении интеграла*

$$I = \int_0^{\pi/2} \ln(\sin(x))dx$$

при различном числе узлов и сравните найденное приближенное значение с точным.

Задача 10.2 *Рекуррентную квадратурную формулу трапеций (см. упражнение 10.1) запишем в виде*

$$R(n+1,1) = \frac{1}{2}R(n,1) + \sum_{i=1}^{2^{n-1}} f(a+(2i-1)h)h.$$

В методе Ромберга эти данные используются для уточнения приближенного значения интеграла согласно рекуррентным соотношениям

$$R(n+1,m+1) = R(n+1,m) + \frac{R(n+1,m+1) - R(n+1,m)}{4^m - 1}, \quad m \le n.$$

Напишите программу для приближенного вычисления интеграла от функции $f(x)$ на интервале $[a, b]$ методом Ромберга на основе рекуррентной квадратурной формулы трапеций. С использованием этой программы вычислите интеграл

$$I = \int_0^2 (4 - x^2)^{1/2}dx$$

при различных n и сравните найденное приближенное значение с точным.

Задача 10.3 *Напишите программу для приближенного вычисления интеграла от функции $(1 - x^2)^{-1/2}f(x)$ на интервале $[-1, 1]$ с использованием квадратурная формула Гаусса—Чебышева, в которой для узлов и коэффициентов имеют место соотношения*

$$x_i = \cos\frac{(2i+1)\pi}{2(n+1)}, \quad c_i = \frac{\pi}{n+1}, \quad i = 0, 1, ..., n.$$

С помощью этой программы вычислите интеграл

$$I = \int_0^{\pi/2} (\sin(x))^{-1/2}dx$$

при различном числе узлов.

Задача 10.4 *Узлы квадратурной формулы Гаусса для вычисления интеграла от функции $f(x)$ на интервале $[-1, 1]$ есть нули полиномов Лежандра*

$$p_n(x) = \frac{(-1)^n}{2^n n!} \frac{d^n}{dx^n} \left((1 - x^2)^n\right),$$

а для коэффициентов квадратурной формулы имеет место выражение

$$c_i = \frac{2}{(1 - x_i^2)} \left(\frac{dp_{n+1}}{dx}(x_i)\right)^2.$$

С учетом этих соотношений напишите программу для вычисления узлов и коэффициентов квадратурной формулы Гаусса—Лежандра. Для нахождения корней полиномов Лежандра используйте метод Ньютона и начальные приближения

$$x_i \approx \cos \frac{(i + 3/4)\pi}{n + 3/2}, \quad i = 0, 1, ..., n.$$

С помощью этой программы рассчитайте параметры квадратурной формулы Гаусса—Лежандра для $n = 2, 3, ..., 8$ и сравните эти результаты с данными в табл. 10.1.

11

Интегральные уравнения

Среди типичных интегральных уравнений можно выделить интегральные уравнения Фредгольма второго рода. Для их приближенного решения применяется метод квадратур. Второй широко используемый класс методов решения интегральных уравнений — различные варианты проекционных методов. Отдельно необходимо выделить интегральные уравнения с переменными пределами интегрирования — интегральные уравнения Вольтерра. Интегральные уравнения Фредгольма первого рода являются характерным примером некорректных задач, для численного решения которых используются методы регуляризации.

Основные обозначения

$K(x,s)$	—	ядро интегрального уравнения
λ	—	числовой параметр
$x_i,\ i=1,2,\ldots,n$	—	узлы квадратурной формулы
α	—	параметр регуляризации
y^k	—	приближенное решение на k-й итерации
$\varphi^i(x),\ i=1,2,\ldots,n$	—	линейно независимые координатные функции

11.1 Задачи для интегральных уравнений

Будем рассматривать одномерные интегральные уравнения, решение которых есть $u(x)$, $x \in [a,b]$. Линейное интегральное уравнение с постоянными пределами интегрирования (уравнение Фредгольма) записывается в виде

$$g(x)u(x) - \lambda \int_a^b K(x,s)u(s) = f(x), \quad x \in [a,b], \tag{11.1}$$

где $k(x,s)$ — ядро интегрального уравнения, $g(x), f(x)$ — заданные функции, а λ — заданный или неизвестный числовой параметр.

Наибольшее внимание уделяется нахождению приближенного решения интегрального уравнения Фредгольма второго рода:

$$u(x) - \lambda \int_a^b K(x,s)u(s)ds = f(x), \quad x \in [a,b] \tag{11.2}$$

при заданном λ.

При $f(x)$ уравнение (11.2) есть однородное уравнение Фредгольма

$$u(x) - \lambda \int_a^b K(x,s)u(s)ds = 0, \quad x \in [a,b], \tag{11.3}$$

которое всегда имеет тривиальное решение $u(x) \equiv 0$. Те значения параметра λ, при которых уравнение (11.3) имеет ненулевое решение, называются характеристическими числами, а соответствующие ненулевые решения уравнения — собственными функциями ($1/\lambda$ — собственные значения).

Линейное интегральное уравнение Фредгольма первого рода имеет вид

$$\int_a^b K(x,s)u(s)ds = f(x), \quad x \in [a,b], \tag{11.4}$$

т.е. в общей записи (11.1) $g(x) = 0$ и $\lambda = -1$. Принципиальные трудности приближенного решения этого уравнения порождены тем, что задача нахождения решения интегрального уравнения первого рода является некорректно поставленной. Некорректность обусловлена, прежде всего, отсутствием устойчивости решения по отношению к малым возмущениям правой части уравнения (11.4).

Отдельного рассмотрения заслуживают интегральные уравнения с переменными пределами интегрирования. Интегральное уравнение Вольтерра второго рода записывается в виде

$$u(x) - \lambda \int_a^x K(x,s)u(s)ds = f(x), \quad x \in [a,b]. \tag{11.5}$$

По аналогии с (11.4), (11.5) для уравнения Вольтерра первого рода имеем

$$\int_a^x K(x,s)u(s)ds = f(x), \quad x \in [a,b]. \tag{11.6}$$

В вычислительной практике рассматриваются и более общие задачи для интегральных уравнений, среди которых отметим прежде всего многомерные интегральные уравнения. Большое внимание уделяется разработке численных методов для специальных классов интегральных уравнений. Отметим, в частности, интегральные уравнения с разностным ядром $K(x-s)$.

11.2 Методы решения интегральных уравнений

Выделены основные классы методов приближенного решения интегральных уравнений. Метод квадратур (механических квадратур) основан на замене интегралов конечными суммами с использованием квадратурных формул. В проекционных методах приближенное решение ищется в виде разложения по системе известных линейно независимых функций. Отмечаются особенности решения интегральных уравнений Вольтерра, кратко обсуждаются методы решения интегральных уравнений первого рода.

Интегральные уравнения Фредгольма второго рода

Будем рассматривать алгоритмы численного решения интегральных уравнений (11.2), считая заданным параметр λ. В основе метода квадратур лежит та или иная квадратурная формула. Пусть $x_1 < x_2 < \cdots < x_n$ — узлы, а c_i, $i = 1, 2, \ldots, n$ — коэффициенты квадратурной формулы на отрезке интегрирования $[a, b]$. При использовании квадратурной формулы

$$\int_a^b \theta(x)dx \approx \sum_{i=1}^{n} c_i \theta(x_i)$$

приближенное решение интегрального уравнения (11.2) определим из системы линейных алгебраических уравнений

$$y_i - \lambda \sum_{j=1}^{n} c_j K(x_i, s_j) y_j = f(x_i), \quad i = 1, 2, \ldots, n, \tag{11.7}$$

где y_i — приближенное решение в узле x_i, $i = 1, 2, \ldots, n$.

В проекционных методах приближенное решение интегрального уравнения (11.2) ищется в виде

$$y(x) = \sum_{i=1}^{n} c_i \varphi_i(x), \tag{11.8}$$

где $\varphi_i(x)$, $i = 1, 2, \ldots, n$ — заданные линейно независимые функции, которые называются координатными. Часто удобнее ориентироваться на несколько отличное от (11.8) представление приближенного решения:

$$y(x) = f(x) + \sum_{i=1}^{n} c_i \varphi_i(x), \tag{11.9}$$

Метод проекционного типа характеризуется выбором координатных функций $\varphi_i(x)$, $i = 1, 2, \ldots, n$ и способом определения вектора неизвестных коэффициентов $c = \{c_1, c_2, \ldots, c_n\}$. Отметим некоторые возможности по нахождению коэффициентов в представлении (11.8), (11.9).

При использовании представления (11.8) определим невязку

$$r(x,c) = \sum_{i=1}^{n} c_i\varphi_i(x) - \lambda \int_a^b K(x,s) \sum_{j=1}^{n} c_j\varphi_j(s)ds - f(x).$$

В методе наименьших квадратов постоянные c_i, $i = 1,2,\ldots,n$ находятся из минимума квадрата нормы невязки в $L_2(a,b)$, т.е.

$$J(c) \equiv \int_a^b r^2(x,c)dx \to \min, \quad c \in \mathbf{R}^n.$$

Для определения c_i, $i = 1,2,\ldots,n$ получим систему линейных алгебраических уравнений

$$\sum_{j=1}^{n} a_{ij}c_j = b_i, \quad i = 1,2,\ldots,n, \tag{11.10}$$

где

$$a_{ij} = \int_a^b (\varphi_i(x) - \lambda \int_a^b K(x,s)\varphi_i(s)ds) \times$$

$$\times (\varphi_j(x) - \lambda \int_a^b K(x,s)\varphi_j(s)ds)dx,$$

$$b_i = \int_a^b f(x)(\varphi_i(x) - \lambda \int_a^b K(x,s)\varphi_i(s)ds)dx, \quad i = 1,2,\ldots,n.$$

Тем самым матрица системы (11.10) симметрична.

В методе Галеркина коэффициенты c_i, $i = 1,2,\ldots,n$ определяются из условия ортогональности в $L_2(a,b)$ невязки $r(x,c)$ функциям $\varphi_i(x)$, $i = 1,2,\ldots,n$:

$$\int_a^n r(x,c)\varphi_i(x)dx, \quad i = 1,2,\ldots,n.$$

В этом случае имеем систему линейных уравнений (11.10), в которой

$$a_{ij} = \int_a^b (\varphi_j(x) - \lambda \int_a^b K(x,s)\varphi_j(s)ds)\varphi_i(x)dx,$$

$$b_i = \int_a^b f(x)\varphi_i(x)dx, \quad i = 1,2,\ldots,n.$$

Отметим среди проекционных методов и метод коллокации. В этом случае на отрезке $[a,b]$ выбирается n точек коллокации x_i, $i = 1,2,\ldots,n$ и коэффициенты c_i, $i = 1,2,\ldots,n$ в представлении (11.8) (или (11.9)) выбираются так, что невязка обращалась в нуль в точках коллокации, т.е.

$$r(x_i,c) = 0, \quad i = 1,2,\ldots,n.$$

Для коэффициентов матрицы и правой части системы (11.10) при использовании представления (11.8) получим

$$a_{ij} = \varphi_j(x_i) - \lambda \int_a^b K(x_i, s)\varphi_j(s)ds,$$

$$b_i = f(x_i), \quad i = 1, 2, \ldots, n.$$

Для решения системы линейных алгебраических уравнений (11.10) применяются прямые или итерационные методы.

Интегральные уравнения с переменными пределами интегрирования

При приближенном решении интегрального уравнения Вольтерра второго рода (11.5) используется как метод квадратур, так и проекционные методы. Для определенности, будем считать, что $x_1 = a$, $x_n = b$. Для точек x_i, $i = 1, 2, \ldots, n$ из (11.5) получим

$$u(x_i) - \lambda \int_a^{x_i} K(x_i, s)u(s)ds = f(x_i), \quad i = 1, 2, \ldots, n. \tag{11.11}$$

Принимая во внимание то, что интегрировать необходимо по отрезку переменной длины, запишем используемую квадратурную формулу в виде

$$\int_a^{x_i} \theta(x)dx \approx \sum_{j=1}^{i} c_j^{(i)}\theta(x_j), \quad i = 2, 3, \ldots, n.$$

Применение к (11.11) дает систему линейных уравнений

$$y_i - \lambda \sum_{j=1}^{i} c_j^{(i)} K(x_i, s_j)y_j = f(x_i), \quad i = 1, 2, \ldots, n. \tag{11.12}$$

Отличительная особенность системы уравнений (11.12) состоит в том, что матрица ее коэффициентов треугольная. Это позволяет найти приближенное решение интегрального уравнения $y_1, y_2, \ldots, y_n$ последовательно друг за другом по рекуррентным формулам в предположении, что все диагональные элементы матрицы ненулевые. Наиболее простые расчетные формулы при решении интегрального уравнения Вольтерра второго рода мы получим при использовании квадратурной формулы трапеций.

При численном решении интегрального уравнения первого рода (11.6) можно ориентироваться на использование метода квадратур. Подобно (11.11) из (11.6) будем иметь

$$\int_a^{x_i} K(x_i, s)u(s)ds = f(x_i), \quad i = 1, 2, \ldots, n,$$

что дает систему линейных алгебраических уравнений

$$\sum_{j=1}^{i} c_j^{(i)} K(x_i, s_j) y_j = f(x_i), \quad i = 1, 2, \ldots, n.$$

Для того чтобы решение этой системы существовало необходимо потребовать выполнение условия $K(x, x) \neq 0$.

При численном решении интегральных уравнений часто полезно провести предварительное преобразование исходной задачи. Типичным примером является приведение интегрального уравнения Вольтерра первого рода к интегральному уравнению второго рода. Будем считать, что ядро и правая часть дифференцируемы $K(x, x) \neq 0$. Тогда от уравнения (11.6) можно перейти к уравнению

$$u(x) + \int_a^x \frac{1}{K(x,x)} \frac{\partial K(x,s)}{\partial x} u(s) ds = \frac{1}{K(x,x)} \frac{df}{dx}(x),$$

которое представляет собой интегральное уравнения Вольтерра второго рода.

Интегральное уравнение Фредгольма первого рода

Интегральное уравнение (11.4) есть наиболее характерный пример некорректно поставленной задачи. Некорректность обусловлена тем, что при малых возмущениях правой части $f(x)$ не гарантируется малого возмущения решения.

Помимо (11.4) рассмотрим уравнение с возмущенной правой частью

$$\int_a^b K(x, s) \tilde{u}(s) ds = \tilde{f}(x), \quad x \in [a, b]. \tag{11.13}$$

Ядро $K(x, s)$ есть вещественная непрерывная функция двух аргументов, а $\tilde{f}(x), f(x) \in L_2(a, b)$, причем

$$\|\tilde{f}(x) - f(x)\| \leq \delta,$$

при использовании обозначений

$$\|u(x)\| = ((u, u))^{1/2}, \quad (u, v) = \int_a^b u(x) v(x) dx.$$

При $\delta \to 0$ норма погрешности решения $\|\tilde{u}(x) - u(x)\|$ не стремиться к нулю.

Определим линейный интегральный оператор

$$Ay = \int_a^b K(x, s) y(s) ds, \quad x \in [a, b]. \tag{11.14}$$

Задачу с неточной правой частью (11.13) запишем в виде операторного уравнения первого рода

$$A\tilde{u} = \tilde{f}. \tag{11.15}$$

В методе регуляризации Тихонова приближенное решение задачи (11.15) находится из минимума сглаживающего функционала:

$$J_\alpha(y) \to \min, \quad y \in L_2(a,b), \tag{11.16}$$

где

$$J_\alpha(y) = \|Ay - \tilde{f}\|^2 + \alpha\|y\|^2,$$

а $\alpha > 0$ — параметр регуляризации.

Обозначим решение задачи (11.16) через y_α. Оно может быть найдено как решение уравнения Эйлера для вариационной задачи (11.16)

$$\alpha y_\alpha + A^* A y_\alpha = A^* \tilde{f},$$

где

$$A^* y = \int_a^b K(s,x) y(s) ds, \quad x \in [a,b].$$

Тем самым приходим к интегральному уравнению Фредгольма

$$\alpha y_\alpha + \int_a^b G(x,s) y_\alpha(s) ds = \psi(x), \quad x \in [a,b]$$

с симметричным ядром

$$G(x,s) = \int_a^b K(t,x) K(t,s) dt$$

и правой частью

$$\psi(x) = \int_a^b K(s,x) f(s) ds.$$

Принципиальный момент в методе регуляризации связан с выбором параметра регуляризации α, его согласованием с погрешностью входных данных. При использовании принципа невязки параметр регуляризации выбирается из условия

$$\|Ay_\alpha - \tilde{f}\| = \delta.$$

При таком выборе $\alpha = \alpha(\delta)$ норма погрешности $\|y_\alpha - u\| \to 0$ при $\delta \to 0$, т.е. приближенное решение стремится к точному решению задачи.

11.3 Упражнения

Упражнение 11.1 *Напишите программу для численного решения интегрального уравнения Фредгольма второго рода методом квадратур с использованием квадратурной формулы трапеций при равномерном разбиении ин-*

тервала интегрирования. С помощью этой программы найдите приближенное решение интегрального уравнения

$$y(x) - \int_{-1}^{1} \frac{1}{\pi(1+(x-s)^2)} y(s)ds = 1$$

при различном числе частичных отрезков.

В модуле fredholm функция `fredholm()` обеспечивает приближенное решение интегрального уравнения

$$u(x) - \int_a^b K(x,s)u(s) = f(x), \quad x \in [a,b]$$

при использовании квадратурной формулы трапеций в методе квадратур. Для решения системы линейных уравнений используется функция `solveLU()` из модуля lu.

Модуль fredholm

```
import numpy as np
from lu import solveLU
def fredholm(k, f, a, b, n):
    """
    Solution Fredholm integral equation of the second kind.
    k(x,s) is the kernel of the integral equation,
    f(x) is the right part, 0 < x,s < b.
    Method with trapezoidal quadrature formula.
    """
    h = (b - a) / n
    A = np.identity(n+1, 'float')
    r = np.zeros((n+1), 'float')
    for i in range(n+1):
        x = a + i*h
        A[i,0] = A[i,0] - k(x,a)*h/2
        for j in range(1,n):
            s = a + j*h
            A[i,j] = A[i,j] - k(x,s)*h
        A[i,n] = A[i,n] - k(x,b)*h/2
        r[i] = f(x)
    y = solveLU(A, r)
    return y
```

Для приближенного решения модельного интегрального уравнения при использовании различного числа узлов $y_i, i = 1, 2, ..., n$ используется следующая программа.

exer_11_1.py

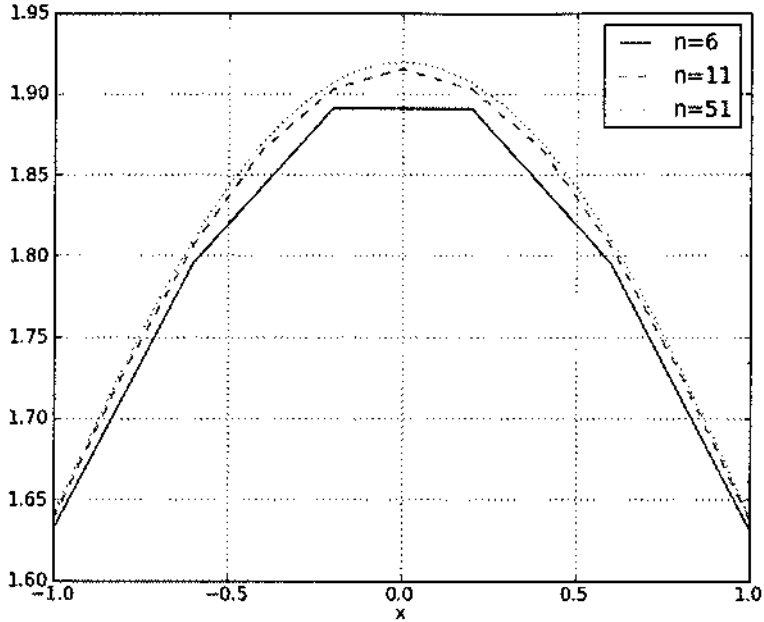

Рис. 11.1 Приближенное решение при числе узлов $n = 6, 11, 51$

```
import numpy as np
import matplotlib.pyplot as plt
from fredholm import fredholm
def k(x,s):
    return 1./(np.pi*(1. + (x-s)**2))
def f(x):
    return 1.
a = -1.
b = 1.
nList = [5, 10, 50]
sglist = ['-', '--', ':']
for kk in range(len(nList)):
    n = nList[kk]
    x = np.linspace(a, b, n+1)
    y = fredholm(k, f, a, b, n)
    sl = 'n=' + str(n+1)
    sg = sglist[kk]
    plt.plot(x, y, sg, label=sl)
plt.xlabel('x')
```

```
plt.grid(True)
plt.legend(loc=0)
plt.show()
```

Сходимость приближенного решения при увеличении числа разбиений n демонстрируется рис. 11.1.

Упражнение 11.2 *Напишите программу для численного решения интегрального уравнения Фредгольма первого рода методом квадратур с использованием квадратурной формулы прямоугольников при равномерном разбиении интервала интегрирования с симметризацией матрицы системы уравнений и ее итерационном решении методом сопряженных градиентов. С помощью этой программы найдите приближенное решение интегрального уравнения*

$$\int_{-1}^{1} |(x-s)|y(s)ds = x^2$$

при различной точности приближенного решения системы линейных уравнений и сравните его с точным решением интегрального уравнения.

После дискретизации приходим к системе уравнений

$$Ay = f.$$

Итерационный метод сопряженных градиентов применяется для приближенного решения системы уравнений

$$A^*Ay = A^*f.$$

В модуле fredholm1 функция `fredholm1()` обеспечивает приближенное решение интегрального уравнения

$$\int_a^b K(x,s)u(s) = f(x), \quad x \in [a,b]$$

при использовании квадратурной формулы прямоугольников в методе квадратур. Для решения системы линейных уравнений используется функция `cg()` из модуля cg.

Модуль fredholm1

```
import numpy as np
from cg import cg
def fredholm1(k, f, a, b, n, tol = 1.0e-9):
    """
    Solution Fredholm integral equation of the first  kind.
    k(x,s) is the kernel of the integral equation,
    f(x) is the right part, 0 < x, s < b.
    CG iterative method with
    rectangle quadrature formula.
    """
```

```
    h = (b - a) / n
    A = np.zeros((n,n), 'float')
    r = np.zeros((n), 'float')
    for i in range(n):
        x = a + h / 2. + i*h
        r[i] = f(x)
        for j in range(n):
            s = a + h / 2. + j*h
            A[i,j] = k(x,s)*h
    # Symmetrization
    B = np.copy(A)
    rr = np.copy(r)
    for i in range(n):
        r[i] = np.dot(B[i,0:n], rr[0:n])
        for j in range(n):
            A[i,j] = np.dot(B[i,0:n], B[0:n,j])
    y, iter = cg(A, r, tol = tol)
    return y, iter
```

Точное решение интегрального уравнения

$$\int_{-1}^{1} |(x-s)| y(s) ds = f(x)$$

есть

$$y(x) = \frac{1}{2} \frac{d^2 f}{dx^2}(x).$$

Для его приближенного решения используется следующая программа.

exer_11_2.py

```
import numpy as np
import matplotlib.pyplot as plt
from fredholm1 import fredholm1
def k(x,s):
    return abs(x-s)
def f(x):
    return x**2
a = -1.
b = 1.
n = 100
h = (b - a) / n
x = np.linspace(a+h/2., b-h/2., n)
erList = [1.0e-3, 1.0e-5, 1.0e-10]
sglist = ['-', '--', ':']
for kk in range(len(erList)):
    er = erList[kk]
```

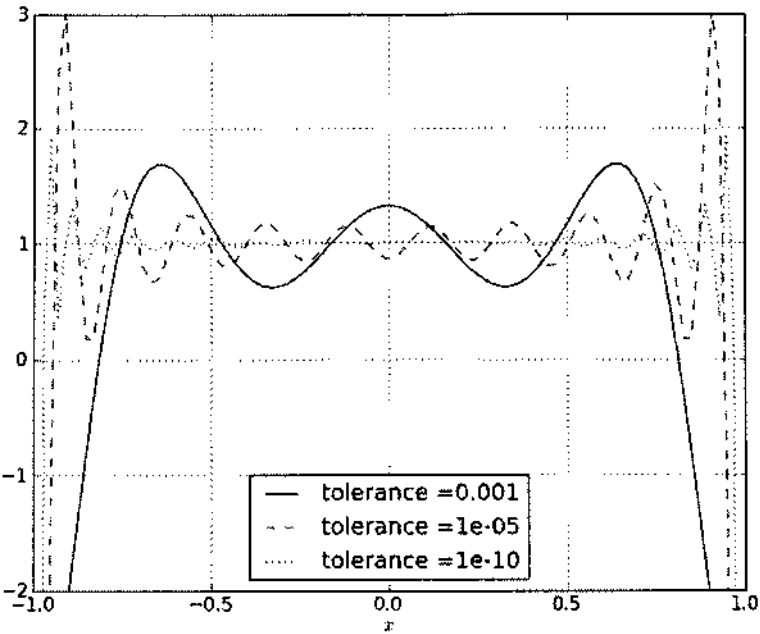

Рис. 11.2 Приближенное решение при различной точности решения системы уравнений

```
    y, iter = fredholm1(k, f, a, b, n, tol = er)
    print 'iteration =', iter, 'tolerance =', er
    sl = 'tolerance =' + str(er)
    sg = sglist[kk]
    plt.plot(x, y, sg, label=sl)
plt.xlabel('$x$')
plt.ylim(-2, 3)
plt.grid(True)
plt.legend(loc=8)
plt.show()
```

```
iteration = 3 tolerance = 0.001
iteration = 21 tolerance = 1e-05
iteration = 99 tolerance = 1e-10
```

Приближенное решение при различном уровне точности итерационного решения системы линейных уравнений представлено на рис. 11.2. Некорректность задачи проявляется в нарастании погрешности приближенного решения вблизи концов интервала при увеличении точности решения системы уравнений.

11.4 Задачи

Задача 11.1 *Напишите программу для численного решения интеграл го уравнения Фредгольма второго рода методом квадратур с использова ем квадратурной формулы Симпсона при равномерном разбиении интерв интегрирования. Используйте эту программу для приближенного реше интегрального уравнения*

$$y(x) - \int_0^1 (x-s)y(s)ds = 3 - 2x.$$

Исследуйте зависимость точности численного решения от числа част ных отрезков.

Задача 11.2 *Напишите программу для численного решения интегральн уравнения Фредгольма второго рода методом Галеркина при использова кусочно линейных координатных функций*

$$\varphi_i(x) \neq 0, \quad x_{i-1} < x < x_{i+1}, \quad i = 1, 2, ..., n$$

и равномерного разбиения интервала интегрирования. С помощью этой п граммы найдите приближенное решение интегрального уравнения

$$y(x) - \int_0^1 |x-s|y(s)ds = x^3$$

при различных n.

Задача 11.3 *Напишите программу для численного решения интегральн уравнения Вольтерра второго рода методом квадратур при использова квадратурной формулы трапеций с равномерным разбиением интервала тегрирования. Примените эту программу для приближенного решение тегрального уравнения*

$$y(x) - \int_0^x e^{x-s}y(s)ds = 1$$

при различном числе частичных отрезков.

Задача 11.4 *Напишите программу для численного решения интегральн уравнения Фредгольма первого рода методом Тихонова при использова квадратурной формулы прямоугольников с равномерным разбиением инт вала интегрирования. С помощью этой программы найдите приближен решение интегрального уравнения*

$$\int_{-1}^1 \frac{\mu}{\mu^2 + (x-s)^2} y(s)ds = \cos(\pi x)$$

при $\mu = 0.05, 0.1, 0.2$.

Задача Коши для обыкновенных дифференциальных уравнений

В вычислительной практике часто приходится иметь дело с задачами с начальными данными для системы дифференциальных уравнений. Для приближенного решения таких задач традиционно широко используются методы Рунге—Кутта, связанные с вычислением правой части системы уравнений в некоторых промежуточных точках. Второй большой класс методов составляют многошаговые методы, когда в вычислениях участвуют три и более расчетных слоев. Отдельно выделяются задачи, для которых решение имеет разномасштабные гармоники (жесткие системы обыкновенных дифференциальных уравнений).

Основные обозначения

$u = \{u_1, u_2, \ldots, u_m\}$ — вектор неизвестных
$f = \{f_1, f_2, \ldots, f_m\}$ — вектор правых частей
$\omega_\tau = \{t_n = n\tau,\ n = 0, 1, \ldots\}$ — равномерная сетка по t
$\tau > 0$ — шаг сетки
$y^n = y(t_n)$ — приближенное решение при $t = t_n$

12.1 Задачи с начальными условиями для систем обыкновенных дифференциальных уравнений

Рассматривается задача Коши для системы обыкновенных дифференциальных уравнений

$$\frac{du_i(t)}{dt} = f_i(t, u_1, u_2, \ldots, u_m), \quad t > 0, \tag{12.1}$$

$$u_i(0) = u_i^0, \quad i = 1, 2, \ldots, m. \tag{12.2}$$

С использованием векторных обозначений задачу (12.1), (12.2) можем записать как задачу Коши для одного уравнения:

$$\frac{du(t)}{dt} = f(t,u), \quad t > 0, \tag{12.3}$$

$$u(0) = u^0. \tag{12.4}$$

В задаче Коши по известному решению в точке $t = 0$ необходимо найти из уравнения (12.4) решение при других t.

12.2 Численные методы решения задачи Коши

Отмечаются классические методы Рунге—Кутта и многошаговые методы решения задачи Коши для систем обыкновенных дифференциальных уравнений, обсуждается специфика численного решения жестких систем.

Методы Рунге—Кутта

При построении численных алгоритмов будем считать, что решение этой дифференциальной задачи существует, оно единственно и обладает необходимыми свойствами гладкости.

При численном решении задачи (12.3), (12.4) будем использовать равномерную, для простоты, сетку по переменной t с шагом $\tau > 0$:

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

Приближенное решение задачи (12.3), (12.4) в точке $t = t_n$ обозначим y^n. Метод сходится в точке t_n, если $|y^n - u(t_n)| \to 0$ при $\tau \to 0$. Метод имеет p-й порядок точности, если $|y^n - u(t_n)| = O(\tau^p)$, $p > 0$ при $\tau \to 0$.

Простейшая разностная схема для приближенного решения задачи (12.3), (12.4) есть

$$\frac{y^{n+1} - y^n}{\tau} = \sigma f(t_{n+1}, y^{n+1}) + (1-\sigma) f(t_n, y^n), \quad n = 0,1,\ldots \tag{12.5}$$

При $\sigma = 0$ имеем явный метод Эйлера и в этом случае разностная схема аппроксимирует уравнение (12.4) с первым порядком.

Симметричная схема ($\sigma = 0,5$ в (12.5)) имеет второй порядок аппроксимации. Эта схема относится к классу неявных — для определения приближенного решения на новом слое необходимо решать нелинейную задачу. Явные схемы второго и более высокого порядка аппроксимации удобно строить ориентируясь на метод предиктор-корректор. На этапе предиктора (предсказания) используется явная схема

$$\frac{\overline{y}_{n+1} - y_n}{\tau} = f(t_n, y_n),$$

а на этапе корректора (уточнения) — схема

$$\frac{y^{n+1}-y^n}{\tau}=\frac{1}{2}(f(t_{n+1},\bar{y}^{n+1})+f(t_n,y^n)),\quad n=0,1,\ldots$$

В одношаговых методах Рунге—Кутта идеи предиктора-корректора реализуются наиболее полно. Этот метод записывается в общем виде

$$\frac{y^{n+1}-y^n}{\tau}=\sum_{i=1}^{s}b_ik_i, \tag{12.6}$$

где

$$k_i=f(t_n+c_i\tau,y^n+\tau\sum_{j=1}^{s}a_{ij}k_j),\quad i=1,2,\ldots,s. \tag{12.7}$$

Формула (12.6) основана на s вычислениях функции f и называется s-стадийной. Если $a_{ij}=0$ при $j\geq i$ имеем явный метод Рунге—Кутта. Если $a_{ij}=0$ при $j>i$ и $a_{ii}\neq 0$, то k_i определяется неявно из уравнения

$$k_i=f(t_n+c_i\tau,y^n+\tau\sum_{j=1}^{i-1}a_{ij}k_j+\tau a_{ii}k_i).$$

О таком методе Рунге—Кутта говорят как о диагонально-неявном.

Параметры b_i,c_i,a_{ij} определяют вариант метода Рунге—Кутта. Используется следующее представление метода (таблица Бутчера):

$$\begin{array}{c|c} c & A \\ \hline & b^* \end{array}=\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & & a_{1s} \\ c_2 & a_{21} & a_{22} & & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \tag{12.8}$$

Одним из наиболее распространенных является явный метод Рунге—Кутта четвертого порядка:

$$k_1=f(t_n,y^n),\quad k_2=f\Big(t_n+\frac{\tau}{2},y^n+\tau\frac{k_1}{2}\Big),$$

$$k_3=f\Big(t_n+\frac{\tau}{2},y^n+\tau\frac{k_2}{2}\Big),\quad k_4=f(t_n+\tau,y^n+\tau k_3),$$

$$\frac{y^{n+1}-y^n}{\tau}=\frac{1}{6}(k_1+2k_2+2k_3+k_4).$$

В компактном представлении (12.8) этого метода имеем

$$\frac{c \quad A}{b^*} = \begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}.$$

Применяя метод Рунге—Кутта (12.6), (12.7) к решению задачи Коши для уравнения

$$\frac{du(t)}{dt} = f(t), \quad t > 0,$$

получим

$$y^{n+1} - y^n = \sum_{i=1}^{s} \tau b_i f(t_n + c_i \tau).$$

Правую часть можно рассматривать как квадратурную формулу для правой части равенства

$$u(t_{n+1}) - u(t_n) = \int_{t_n}^{t_{n+1}} f(t)dt.$$

Исследование устойчивости используемых разностных схем при решении задачи Коши для систем обыкновенных дифференциальных уравнений проводится чаще всего на модельном одномерном уравнении

$$\frac{du(t)}{dt} = \lambda u, \quad t > 0, \tag{12.9}$$

где λ — комплексное число. Для конкретного численного метода рассматривается множество всех точек комплексной плоскости $\mu = \tau\lambda$, для которых имеет место устойчивость. Для явного метода Эйлера область устойчивости представляет круг единичного радиуса с центром в точке $(-1, 0)$. Метод называется A-устойчивым, если область его устойчивости содержит полуплоскость $\mathrm{Re}\ \mu < 0$. При $\mathrm{Re}\ \lambda < 0$ устойчиво решение уравнения (12.9) и поэтому для этой задачи условие A-устойчивости означает абсолютную устойчивость (устойчивость при всех $\tau > 0$).

Многошаговые методы

В методах Рунге—Кутта в вычислениях участвуют значения приближенного решения только в двух соседних узлах y^k и y^{n+1} — один шаг по переменной t.

Линейный m-шаговый разностный метод записывается в виде

$$\frac{1}{\tau}\sum_{i=0}^{m} a_i y^{n+1-i} = \sum_{i=0}^{m} b_i f(t_{n+1-i}, y^{n+1-i}), \quad n = m-1, m \ldots \tag{12.10}$$

Вариант численного метода определяется заданием коэффициентов a_i, b_i, $i = 0, 1, \ldots, m$, причем $a_0 \neq 0$. Для начала расчетов по рекуррентной формуле (12.10) необходимо задать m начальных значений $y^0, y^1, \ldots, y^{m-1}$.

Различные варианты многошаговых методов (методы Адамса) решения задачи с начальными условиями для систем обыкновенных дифференциальных уравнений могут быть получены на основе использования квадратурных формул для правой части равенства

$$u(t_{n+1}) - u(t_n) = \int_{t_n}^{t_{n+1}} f(t, u)dt. \tag{12.11}$$

Для получения неявного многошагового метода используем для подынтегральной функции интерполяционную формулу по значениям функции

$$f^{n+1} = f(t_{n+1}, y^{n+1}), f^n, \ldots, f^{n+1-m},$$

т.е.

$$\frac{y^{n+1} - y^n}{\tau} = \sum_{i=0}^{m} b_i f(t_{n+1-i}, y^{n+1-i}). \tag{12.12}$$

Для интерполяционного метода Адамса (12.12) наивысший порядок аппроксимации равен $m + 1$.

Для построения явных многошаговых методов можно использовать процедуру экстраполяции подынтегральной функции в правой части (12.11). В этом случае приближение осуществляется по значениям $f^n, f^{n-1}, \ldots, f^{n+1-m}$ и поэтому

$$\frac{y^{n+1} - y^n}{\tau} = \sum_{i=1}^{m} b_i f(t_{n+1-i}, y^{n+1-i}). \tag{12.13}$$

Для экстраполяционного метода Адамса (12.13) погрешность аппроксимации имеет m-й порядок.

Примерами методов Адамса (12.12), (12.13) при $m = 3$ являются

$$\frac{y^{n+1} - y^n}{\tau} = \frac{1}{24}(9f^{n+1} + 19f^n - 5f^{n-1} + f^{n-2}), \tag{12.14}$$

$$\frac{y^{n+1} - y^n}{\tau} = \frac{1}{12}(23f^n - 16f^{n-1} + 5f^{n-2}) \tag{12.15}$$

соответственно.

На основе методов Адамса строятся и схемы предиктор-корректор. На этапе предиктор используется явный метод Адамса, на этапе корректора – аналог неявного метода Адамса. Например, при использовании методов третьего

порядка аппроксимации в соответствии с (12.15) для предсказания решени положим

$$\frac{\overline{y}^{n+1} - y^n}{\tau} = \frac{1}{12}(23f^n - 16f^{n-1} + 5f^{n-2}).$$

Для уточнения решения (см. (12.14)) используется схема

$$\frac{y^{n+1} - y^n}{\tau} = \frac{1}{24}(9f(t_{n+1}, \overline{y}^{n+1}) + 19f^n - 5f^{n-1} + f^{n-2}).$$

Аналогично строятся и другие классы многошаговых методов.

Жесткие системы обыкновенных дифференциальных уравнений

При численном решении задачи Коши для систем обыкновенных дифферен циальных уравнений (12.1), (12.2) могут возникнуть дополнительные тру, ности, порожденные жесткостью системы. Локальные особенности поведени решения в точке $u = w$ передаются линейной системой

$$\frac{dv_i(t)}{dt} = \sum_{j=1}^{m} \frac{\partial f_i}{\partial u_j}(t, w)v + \overline{f}_i(t), \quad t > 0.$$

Пусть $\lambda_i(t)$, $i = 1, 2, \ldots, m$ — собственные числа матрицы

$$A(t) = \{a_{ij}(t)\}, \quad a_{ij}(t) = \frac{\partial f_i}{\partial u_j}(t, w).$$

Система уравнений (12.1) является жесткой, если число

$$S(t) = \frac{\max\limits_{1 \le i \le m} |\mathrm{Re}\lambda_i(t)}{\min\limits_{1 \le i \le m} |\mathrm{Re}\lambda_i(t)}$$

велико. Это означает, что в решении присутствуют составляющие с силы различающимися масштабами изменения по переменной t.

Для численное решения жестких задач используются вычислительные алг ритмы, которые имеют повышенный запас устойчивости. Необходимо ориен тироваться на использование A-устойчивых или $A(\alpha)$-устойчивых методов.

Метод называется A-устойчивым, если при решении задачи Коши для ура нения (12.9) область его устойчивости содержит угол

$$|\arg(-\mu)| < \alpha, \quad \mu = \lambda\tau.$$

Среди A-устойчивых методов можно выделить чисто неявные многошаговы методы (методы Гира), когда

$$\frac{1}{\tau}\sum_{i=0}^{m} a_i y^{n+1-i} = f(t_{n+1}, y^{n+1}).$$

В частности, при $m = 3$ имеем схему

$$\frac{11y^{n+1} - 18y^n + 9y^{n-1} - 2y^{n-2}}{6\tau} = f(t_{n+1}, y^{n+1}),$$

которая имеет третий порядок аппроксимации.

12.3 Упражнения

Упражнение 12.1 *Напишите программу для численного решения задачи Коши для системы обыкновенных дифференциальных уравнений явным методом Рунге–Кутта четвертого порядка. Продемонстрируйте работоспособность этой программы при решении задачи Коши*

$$\frac{d^2u}{dt^2} = -\sin(u), \quad 0 < t < 4\pi,$$

$$u(0) = 1, \quad \frac{du}{dt}(0) = 0.$$

В модуле rungeKutta функция rungeKutta() реализует решение задачи Коши для системы ОДУ методом Рунге–Кутта четвертого порядка.

Модуль rungeKutta

```
import numpy as np
def rungeKutta(f, t0, y0, tEnd, tau):
    """
    Solve the initial value problem y' = f(t,y)
    by 4th-order Runge-Kutta method.
    t0,y0 are the initial conditions,
    tEnd is the terminal value of t,
    tau is the step.
    """
    def increment(f, t, y, tau):
        k0 = tau * f(t,y)
        k1 = tau * f(t + tau/2., y + k0/2.)
        k2 = tau * f(t + tau/2., y + k1/2.)
        k3 = tau * f(t + tau, y + k2)
        return (k0 + 2.*k1 + 2.*k2 + k3) / 6.
    t = []
    y = []
    t.append(t0)
    y.append(y0)
    while t0 < tEnd:
        tau = min(tau, tEnd - t0)
        y0 = y0 + increment(f, t0, y0, tau)
```

```
        t0 = t0 + tau
        t.append(t0)
        y.append(y0)
    return np.array(t), np.array(y)
```

При приближенном решении модельной задачи Коши для уравнения второго порядка сначала переходим от одного уравнения второго порядка к системе из двух уравнений

$$\frac{dy_1}{dt} = y_2, \quad \frac{dy_2}{dt} = -\sin(y_1), \quad 0 < t < 4\pi.$$

Решение задачи при заданном шаге интегрирования τ обеспечивает следующая программа.

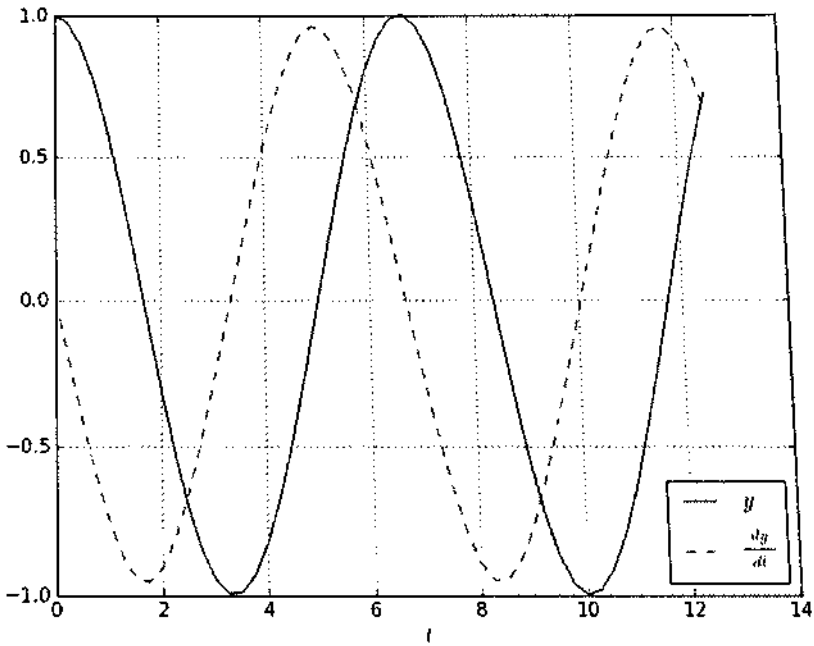

Рис. 12.1 Приближенное решение задачи при $\tau = 0.25$

exer−12.1.py

```
import numpy as np
import math as mt
import matplotlib.pyplot as plt
from rungeKutta import rungeKutta
def f(t, y):
    f = np.zeros((2),'float')
```

```
    f[0] = y[1]
    f[1] = - mt.sin(y[0])
    return f
t0 = 0.
tEnd = 4.*np.pi
y0 = np.array([1., 0.])
tau = 0.25
t, y = rungeKutta(f, t0, y0, tEnd, tau)
for n in range(0, 2):
    r = y[:, n]
    st = '$y$'
    sg = '-'
    if n == 1:
        st = '$\\frac{d y}{dt}$'
        sg = '--'
    plt.plot(t, r, sg, label=st)
plt.legend(loc=0)
plt.xlabel('$t$')
plt.grid(True)
plt.show()
```

В рассматриваемой задаче (колебания маятника) при малых y имеем $\sin(y) \approx y$ и решение уравнение имеет период равный 2π. Нелинейность проявляется, в частности, в увеличении периода (см. рис. 12.1).

Упражнение 12.2 *Напишите программу для численного решения задачи Коши для системы обыкновенных дифференциальных уравнений с использованием двухслойной схемы с весом при решении системы нелинейных уравнений на новом временном слое методом Ньютона. Используйте эту программу для решения задачи Коши (модель Лотка—Вольтерра)*

$$\frac{dy_1}{dt} = y_1 - y_1 y_2, \quad \frac{dy_2}{dt} = -y_2 + y_1 y_2, \quad 0 < t \le 10,$$

$$y_1(0) = 2, \quad y_2(0) = 2.$$

Реализация двухслойной (одношаговой) схемы с весом проводится функцией `oneStep()` в модуле oneStep. Для решения системы нелинейных уравнений используется модуль newton.

Модуль oneStep

```
import numpy as np
from newton import newton
def oneStep(f, t0, y0, tEnd, nTime, theta):
    """
    Solve the initial value problem y' = f(t,y)
    by one-step methods implisit  method.
```

```
    t0,y0 are the initial conditions,
    tEnd is the terminal value of t,
    nTime is the number of steps.
    """
    tau = (tEnd - t0) / nTime
    def f1(y1):
        f1 = y1 - tau * theta * f(t0+tau,y1)   \
           - y0 - tau * (1.-theta) * f(t0+tau,y0)
        return f1
    t = []
    y = []
    t.append(t0)
    y.append(y0)
    for i in range(nTime):
        r = y0 - tau * f(t0+tau,y0)
        y1, iter = newton(f1, r)
        y0 = y1
        t0 = t0 + tau
        t.append(t0)
        y.append(y0)
    return np.array(t), np.array(y)
```

Решение модельной задачи Коши при постоянном шаге по времени обеспечивает следующая программа.

exer_12_2.py

```
import numpy as np
import matplotlib.pyplot as plt
from oneStep import oneStep
def f(t, y):
    f = np.zeros((2),'float')
    f[0] = y[0] - y[0]*y[1]
    f[1] = -y[1] + y[0]*y[1]
    return f
t0 = 0.
tEnd = 10.
y0 = np.array([2., 2.])
nTime = 50
theta = 0.5
t, y = oneStep(f, t0, y0, tEnd, nTime, theta)
for n in range(0, 2):
    r = y[:, n]
    st = '$y_1$'
    sg = '-'
    if n == 1:
```

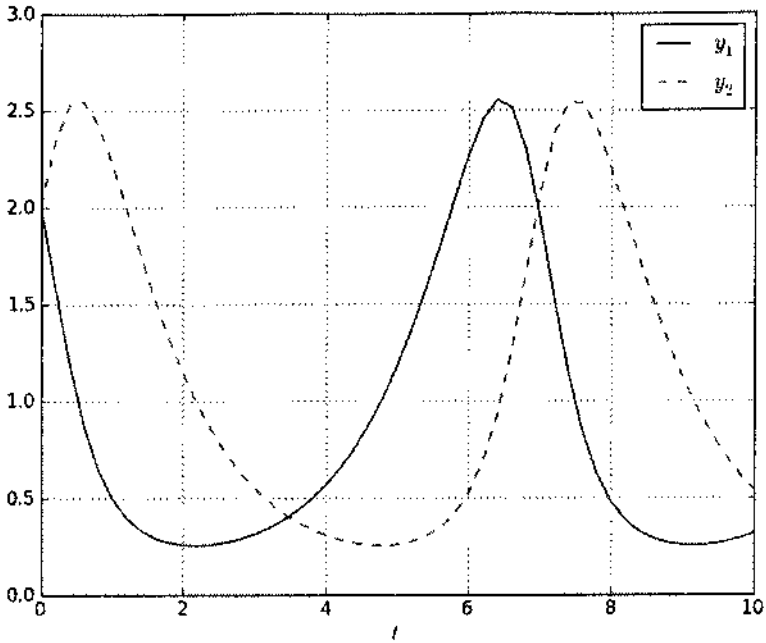

Рис. 12.2 Решение задачи при $\theta = 0.5$

```
        st = '$y_2$'
        sg = '--'
    plt.plot(t, r, sg, label=st)
plt.legend(loc=0)
plt.xlabel('$t$')
plt.grid(True)
plt.show()
```

Решении задачи при 50 шагах по времени и использовании симметричной схемы $\theta = 0.5$ (второй порядок точности) показано на рис. 12.2. Использование чисто неявной схемы $\theta = 1$ приводит к заметному уменьшению амплитуды колебаний.

12.4 Задачи

Задача 12.1 *Для контроля шага интегрирования при использовании явного метода Рунге–Кутта используется следующая процедура (метод Рунге–Кутта–Фельберга). Сначала вычисляются следующие значения*

$$k_1 = f(t_n, y^n),$$

$$k_3 = f\left(t_n + \frac{3}{8}\tau, y^n + \frac{3}{32}\tau k_1 + \frac{9}{32}\tau k_2\right),$$

$$k_4 = f\left(t_n + \frac{12}{13}\tau, y^n + \frac{1932}{2197}\tau k_1 - \frac{7200}{2197}\tau k_2 + \frac{7296}{2197}\tau k_3\right),$$

$$k_5 = f\left(t_n + \tau, y^n + \frac{439}{216}\tau k_1 - 8\tau k_2 + \frac{3680}{513}\tau k_3 - \frac{845}{4104}\tau k_4\right),$$

$$k_6 = f\left(t_n + \frac{1}{2}\tau, y^n - \frac{8}{27}\tau k_1 + 2\tau k_2 - \frac{3544}{2565}\tau k_3 + \frac{1859}{4104}\tau k_4 - \frac{11}{40}\tau k_5\right).$$

Затем ищется приближенное решение методом Рунге—Кутта четвертого порядка, когда

$$\frac{\overline{y}^{n+1} - y^n}{\tau} = \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5.$$

Это решение сравнивается с более точным, которое мы получаем методом Рунге—Кутта пятого порядка точности:

$$\frac{y^{n+1} - y^n}{\tau} = \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6.$$

Оптимальная величина шага есть $s\tau$, где

$$s = \left(\frac{\varepsilon\tau}{2|y^{n+1} - \overline{y}^{n+1}|}\right)^{1/4},$$

а ε — заданная допустимая погрешность приближенного решения. Напишите программу, которая реализует подобную стратегию контроля шага по времени для численного решения задачи Коши для системы обыкновенных дифференциальных уравнений. Проведите численные эксперименты решения с различной точностью ε задачи

$$\frac{du}{dt} = 1 + u^2, \quad t > 0, \quad u(0) = 0.$$

Задача 12.2 *Напишите программу, которая реализует метод предиктор-корректора:*

$$\frac{\overline{y}^{n+1} - y^n}{\tau} = \frac{1}{24}(55f^n - 59f^{n-1} + 37 - 9f^{n-3}),$$

$$\frac{y^{n+1} - y^n}{\tau} = \frac{1}{24}(9f(t_{n+1}, \overline{y}^{n+1}) + 19f^n - 5f^{n-1} + f^{n-2}).$$

С помощью этой программы решите задачу

$$\frac{dy_1}{dt} = y_2 - y_3, \quad \frac{dy_2}{dt} = y_1 + ay_2, \quad \frac{dy_3}{dt} = b + y_3(y_1 - c), \quad 0 < t \le 100,$$

$$y_1(0) = 1, \quad y_2(0) = 1, \quad y_3(0) = 1$$

при $a, b, c = 0.2, 0.2, 2.5$ *и* $a, b, c = 0.2, 0.2, 5$.

Задача 12.3 *Напишите программу для приближенного решения задачи Коши для уравнения второго порядка*

$$\frac{d^2u}{dt^2} = f(t, u), \quad 0 < t < T,$$

$$u(0) = u^0, \quad \frac{du}{dt}(0) = v^0$$

методом Штёрмера третьего порядка точности:

$$\frac{y^{n+1} - 2y^n + y^{n-1}}{\tau^2} = \frac{1}{12}(13f(t_n, y^n) - 2f(t_{n-1}, y^{n-1}) + f(t_{n-2}, y^{n-2})).$$

Используйте эту программу для решения задачи с правой частью $f(t, u) = -\sin(u)$ *при* $T = 4\pi$ *и начальных условиях* $u^0 = 1$ *и* $v^0 = 0$.

Задача 12.4 *Напишите программу для приближенного решения задачи Коши для системы обыкновенных дифференциальных уравнений первого порядка методом Гира четвертого порядка точности:*

$$\frac{25y^{n+1} - 48y^n + 36y^{n-1} - 16y^{n-2} + 3y^{n-3}}{12\tau} = f(t_{n+1}, y^{n+1}).$$

С ее помощью найдите решение задачи

$$\frac{dy_1}{dt} = y_2, \quad \frac{dy_2}{dt} = \mu(1 - y_1^2)y_2 - y_1, \quad 0 < t \leq 100,$$

$$y_1(0) = 2, \quad y_2(0) = 0$$

при $\mu = 50$.

Краевые задачи для обыкновенных дифференциальных уравнений

Наиболее важным классом краевых задач для обыкновенных дифференциальных уравнений являются задачи для уравнения второго порядка. Отмечены основные подходы к построению дискретных аналогов краевых задач с различными граничными условиями. Рассмотрены вопросы сходимости приближенного решения к точному и вычислительной реализации на основе использования прямых методов линейной алгебры. Помимо уравнений второго порядка кратко обсуждаются краевые задачи для модельного обыкновенного дифференциального уравнения четвертого порядка. Основное внимание уделяется разностным методам приближенного решения краевых задач.

Основные обозначения

$u = u(x),\ x \in [0,l]$	—	неизвестная функция
$0 = x_0, x_1, \ldots, x_N = l$	—	узлы сетки
h	—	шаг равномерной сетки
ω	—	множество внутренних узлов
$\partial\omega$	—	множество граничных узлов
H	—	гильбертово пространство сеточных функций
$(\cdot,\cdot)$	—	скалярное произведение в H
$\|\cdot\|$	—	норма в H
$y_x = (y(x+h) - y(x))/h$	—	правая разностная производная в точке x
$y_{\bar{x}} = (y(x) - y(x-h))/h$	—	левая разностная производная в точке x
$y_{\mathring{x}} = \frac{1}{2}(y_x + y_{\bar{x}})$	—	центральная разностная производная в точке x
$y_{\bar{x}x} = (y_x - y_{\bar{x}})/h$	—	вторая разностная производная в точке x

13.1 Краевые задачи

В качестве базового рассматривается обыкновенное дифференциальное уравнение второго порядка

$$-\frac{d}{dx}\left(k(x)\frac{du}{dx}\right)+q(x)u=f(x),\quad 0<x<l \tag{13.1}$$

с переменными коэффициентами

$$k(x)\geq\kappa>0,\quad q(x)\geq 0.$$

Для однозначного определения неизвестной функции $u(x)$ уравнение (13.1) дополняется двумя граничными условиями на концах отрезка $[0,l]$. Задаваться может функция, например, $u(x)$ (граничное условие первого рода), поток $w(x)=-k(x)\dfrac{du}{dx}(x)$ (граничное условие второго рода) или же их линейная комбинация (граничное условие третьего рода):

$$u(0)=\mu_1,\quad u(l)=\mu_2, \tag{13.2}$$

$$-k(0)\frac{du}{dx}(0)=\mu_1,\quad k(l)\frac{du}{dx}(l)=\mu_2, \tag{13.3}$$

$$-k(0)\frac{du}{dx}(0)+\sigma_1 u(0)=\mu_1,\quad k(l)\frac{du}{dx}(l)+\sigma_2 u(l)=\mu_2. \tag{13.4}$$

Эллиптические уравнения второго порядка, прототипом которых является уравнение (13.1), используются при моделирование многих физико-механических процессов.

В задачах с разрывными коэффициентами (контакт двух сред) формулируются дополнительные условия. Простейшие из них (условие идеального контакта) для уравнения (13.1) связывается с непрерывностью решения и потока в точке контакта $x=x^*$:

$$[u(x)]=0,\quad \left[k(x)\frac{du}{dx}\right]=0,\quad x=x^*,$$

где использованы обозначения

$$[g(x)]=g(x+0)-g(x-0).$$

Отдельного рассмотрения заслуживают задачи с несамосопряженным оператором, когда, например,

$$-\frac{d}{dx}\left(k(x)\frac{du}{dx}\right)+v(x)\frac{du}{dx}+q(x)u=f(x),\quad 0<x<l. \tag{13.5}$$

Уравнение конвекции-диффузии (13.5) является модельным при исследование процессов в механике сплошной среды.

При описании деформаций пластин и оболочек, задач гидродинамики математические модели включают эллиптические уравнения четвертого порядка.

Их рассмотрение необходимо начать с краевой задачи для обыкновенно дифференциального уравнения четвертого порядка. Простейшим такой з дачей является задача для уравнения

$$\frac{d^4u}{dx^4}(x) = f(x), \quad 0 < x < l. \tag{13.6}$$

В этом случае задаются по два граничных условиях на концах отрезка. Н пример, уравнение (13.6) дополняется условиями первого рода:

$$u(0) = \mu_1, \quad u(l) = \mu_2, \tag{13.7}$$

$$\frac{du}{dx}(0) = \nu_1, \quad \frac{du}{dx}(l) = \nu_2. \tag{13.8}$$

При формулировке других типов краевых задач для уравнения (13.6) в гр ничных точках могут участвовать вторая и третья производные.

13.2 Численные методы решения краевых задач

При построении вычислительных алгоритмов для приближенного решени краевых задач для обыкновенных дифференциальных уравнений основно внимание уделяется вопросам аппроксимации уравнений, краевых услови и условий сопряжения для задач с разрывными коэффициентами. Прово дится исследование точности приближенного решения в различных норма обсуждаются особенности прямых методов решения сеточных уравнений дл рассматриваемого класса задач.

Аппроксимация краевых задач

Обозначим через $\bar{\omega}$ *равномерную сетку*, для простоты, с шагом h на инте вале $[0, l]$:

$$\bar{\omega} = \{x \mid x = x_i = ih, \quad i = 0, 1, \ldots, N, \quad Nh = l\},$$

причем ω — множество внутренних узлов, а $\partial\omega$ — множество граничных у лов.

Будем использовать безындексные обозначения, когда $u = u_i = u(x_i)$. Дл левой разностной производной имеем

$$u_{\bar{x}} \equiv \frac{u_i - u_{i-1}}{h} = \frac{du}{dx}(x_i) - \frac{h}{2}\frac{d^2u}{dx^2}(x_i) + O(h^2).$$

Тем самым левая разностная производная $u_{\bar{x}}$ аппроксимирует первую прои водную du/dx с первым порядком (погрешность аппроксимации $O(h)$ в каж дом внутреннем узле) при $u(x) \in C^{(2)}(\Omega)$. Аналогично для правой разностно производной получим

$$u_x \equiv \frac{u_{i+1} - u_i}{h} = \frac{du}{dx}(x_i) + \frac{h}{2}\frac{d^2u}{dx^2}(x_i) + O(h^2).$$

Для трехточечного шаблона (узлы x_{i-1}, x_i, x_{i+1}) можно использовать центральную разностную производную:

$$u_{\mathring{x}} \equiv \frac{u_{i+1} - u_{i-1}}{2h} = \frac{du}{dx}(x_i) + \frac{h^2}{3}\frac{d^3u}{dx^3}(x_i) + O(h^3),$$

которая аппроксимирует производную du/dx со вторым порядком при $u(x) \in C^{(3)}(\Omega)$.

Для второй производной d^2u/dx^2 получим

$$u_{\mathring{x}x} = \frac{u_x - u_{\bar{x}}}{h} = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}.$$

Этот разностный оператор аппроксимирует в узле $x = x_i$ вторую производную со вторым порядком при $u(x) \in C^{(4)}(0, l)$.

Для внутренних узлов сетки аппроксимируем дифференциальный оператор

$$\mathcal{L}u = -\frac{d}{dx}\left(k(x)\frac{du}{dx}\right) + q(x)u, \quad x \in (0, l) \tag{13.9}$$

с достаточно гладкими коэффициентами и решением разностным оператором

$$Ly = -(ay_{\bar{x}})_x + cy, \quad x \in \omega. \tag{13.10}$$

Для аппроксимации со вторым порядком необходимо выбрать коэффициенты разностного оператора так, чтобы

$$\frac{a_{i+1} - a_i}{h} = \frac{dk}{dx}(x_i) + O(h^2), \tag{13.11}$$

$$\frac{a_{i+1} + a_i}{2} = k(x_i) + O(h^2), \tag{13.12}$$

$$c_i = q(x_i) + O(h^2). \tag{13.13}$$

В соответствии с (13.13) положим, например, $c_i = q(x_i)$, а условиям (13.11), (13.12) удовлетворяют, в частности, следующие формулы для определения a_i:

$$a_i = k_{i-1/2} = k(x_i - 0.5h),$$

$$a_i = \frac{k_{i-1} + k_i}{2},$$

$$a_i = 2\left(\frac{1}{k_{i-1}} + \frac{1}{k_i}\right)^{-1}$$

Метод формальной замены дифференциальных операторов разностными может использоваться и при аппроксимации граничных условий. Для построения разностных схем в задачах с разрывными коэффициентами необходимо ориентироваться на использование интегро-интерполяционного метода (метода баланса).

При построении разностных схем естественно исходить из законов сохранения (балансов) для отдельных ячеек разностной сетки. В уравнении (13.1) выделим контрольные объемы в виде отрезков $x_{i-1/2} \le x \le x_{i+1/2}$, где $x_{i-1/2} = (i-1/2)h$. Интегрирование уравнения (13.1) по контрольному объему дает

$$q_{i+1/2} - q_{i-1/2} + \int\limits_{x_{i-1/2}}^{x_{i+1/2}} q(x)u(x)dx = \int\limits_{x_{i-1/2}}^{x_{i+1/2}} f(x)dx.$$

Для получения разностного уравнения из этого балансного соотношения необходимо использовать те или иные восполнения сеточных функций. Само решение будем искать в целых узлах ($y(x)$, $x = x_i$), а потоки — в полуцелых ($q(x)$, $x = x_{i+1/2}$). Это приводит нас к разностному уравнению

$$Ly = \varphi, \quad x \in \omega, \tag{13.14}$$

в котором оператор L определен согласно (13.10) с коэффициентами

$$a_i = \left(\frac{1}{h} \int\limits_{x_{i-1}}^{x_i} \frac{dx}{k(x)} \right)^{-1}, \tag{13.15}$$

$$c_i = \frac{1}{h} \int\limits_{x_{i-1/2}}^{x_{i+1/2}} q(x)dx.$$

Правая часть уравнения (13.14) есть

$$\varphi_i = \frac{1}{h} \int\limits_{x_{i-1/2}}^{x_{i+1/2}} f(x)dx.$$

Аналогично проводятся аппроксимации уравнения (13.1) и на неравномерных сетках.

Построение дискретных аналогов краевых задач для уравнения (13.1) может осуществляться на основе метода конечных элементов. Используя простейшие кусочно-линейные элементы, представим приближенное решение в виде

$$y(x) = \sum_{i=1}^{N-1} y_i w_i(x), \tag{13.16}$$

где пробные функции $w_i(x)$ имеют вид

$$w_i(x) = \begin{cases} 0, & x < x_{i-1}, \\ (x - x_{i-1})/h, & x_{i-1} \le x \le x_i, \\ (x_{i+1} - x)/h, & x_i \le x \le x_{i+1}, \\ 0, & x > x_{i+1}. \end{cases}$$

Коэффициенты разложения в методе Галеркина определяются из системы линейных уравнений, которую мы получаем после умножении исходного уравнения (13.1) на функцию $w_i(x)$ и интегрирования по всей области. С учетом финитности пробных функций получим

$$\int\limits_{x_{i-1}}^{x_{i+1}} k(x)\frac{dy}{dx}\frac{dw_i}{dx}dx + \int\limits_{x_{i-1}}^{x_{i+1}} q(x)y(x)w_i(x)dx = \int\limits_{x_{i-1}}^{x_{i+1}} f(x)w_i(x)dx.$$

Подстановка представления приближенного решения (13.16) приводит к трехточечному разностному уравнению (13.14), в котором

$$a_i = \frac{1}{h}\int\limits_{x_{i-1}}^{x_i} k(x)dx - \frac{1}{h}\int\limits_{x_{i-1}}^{x_i} q(x)(x - x_{i-1})(x_i - x)dx,$$

$$c_i = \frac{1}{h^2}\left(\int\limits_{x_{i-1}}^{x_i} q(x)(x - x_{i-1})dx + \int\limits_{x_{i-1}}^{x_i} q(x)(x_{i+1} - x)dx\right),$$

$$\varphi_i = \frac{1}{h^2}\left(\int\limits_{x_{i-1}}^{x_i} f(x)(x - x_{i-1})dx + \int\limits_{x_{i-1}}^{x_i} f(x)(x_{i+1} - x)dx\right).$$

Наиболее просто аппроксимируются граничные условия (13.2):

$$y_0 = \mu_1, \quad y_N = \mu_2. \tag{13.17}$$

Для аппроксимации граничных условий второго и третьего рода со вторым порядком в граничных узлах $x = x_0 = 0$ и $x = x_N = l$ привлекается уравнение (13.1) — аппроксимация на решениях задачи. В случае уравнения (13.1) краевые условия (13.4) аппроксимируются разностными соотношениями

$$-a_1 y_{\bar{x},1} + \left(\sigma_1 + \frac{h}{2}q_0\right)y_0 = \mu_1 + \frac{h}{2}f_0,$$

$$a_N y_{x,N} + \left(\sigma_2 + \frac{h}{2}q_N\right)y_N = \mu_2 + \frac{h}{2}f_N.$$

К подобным аппроксимациям мы приходим при использовании интегро-интерполяционного метода и построении схем конечных элементов.

Сходимость разностных схем

Исследование сходимости приближенного решения к точному при численном решении краевых задач базируется на основе априорных оценок в сеточном гильбертовом пространстве. При исследовании сходимости в равномерной норме привлекается принцип максимума и разностная функция Грина.

На множестве внутренних узлов ω и на сетке

$$\omega^+ = \{x \mid x = x_i = ih, \quad i = 1, 2, \ldots, N, \quad Nh = l\}$$

определим скалярные произведения

$$(y, w) \equiv \sum_{x \in \omega} y(x) w(x) h,$$

$$(y, w)^+ \equiv \sum_{x \in \omega^+} y(x) w(x) h.$$

В сеточных гильбертовых пространствах H и H^+ норму введем соотношением

$$\|y\| = (y, y)^{1/2}, \quad \|y\|^+ = ((y, y)^+)^{1/2}.$$

Рассмотрим разностное уравнение (13.14) при однородных краевых условиях первого рода:

$$y_0 = 0, \quad y_N = 0. \tag{13.18}$$

Для любых сеточных функций, обращающихся в нуль на $\partial\omega$, верно неравенство (разностное неравенство Фридрихса)

$$\|y\|^2 \le M_0 (\|y_{\bar{x}}\|^+)^2, \quad M_0 = \frac{l^2}{8}. \tag{13.19}$$

С учетом этого на множестве сеточных функций, удовлетворяющих (13.18), разностный оператор L, определяемый согласно (13.10), является самосопряженным и положительно определенным:

$$L = L^* \ge \frac{\kappa}{M_0} E. \tag{13.20}$$

Для исследования точности разностной схемы (13.14), (13.17) рассмотрим задачу для погрешности приближенного решения

$$z(x) = y(x) - u(x), \quad x \in \bar{\omega}.$$

Для погрешности приближенного решения задачи (13.1), (13.2) получим разностную задачу

$$Lz = \psi(x), \quad x \in \omega,$$

$$z_0 = 0, \quad z_N = 0,$$

где $\psi(x)$ — погрешность аппроксимации:

$$\psi(x) = \varphi(x) - Lu, \quad x \in \omega.$$

В случае достаточно гладких коэффициентов и решения для погрешности аппроксимации получим

$$\psi(x) = O(h^2), \quad x \in \omega.$$

Для погрешности рассматриваемой разностной схемы справедлива априорная оценка

$$\|z_{\bar{x}}\|^{+} \le \frac{M_0^{1/2}}{\kappa}\|\psi\|,$$

которая обеспечивает сходимость разностного решения к точному решению дифференциальной задачи со вторым порядком.

При рассмотрении одномерных задач конвекции-диффузии мы ориентируемся на использовании трехточечных разностных схем, которые запишем для внутренних узлов в виде

$$-\alpha_i y_{i-1} + \gamma_i y_i - \beta_i y_{i+1} = \varphi_i, \quad i = 1, 2, \ldots, N-1. \tag{13.21}$$

Для граничных узлов считаем выполненными условия (13.18).

Будем рассматривать разностные схемы (13.18), (13.21), в которых

$$\alpha_i > 0, \quad \beta_i > 0, \quad \gamma_i > 0, \quad i = 1, 2, \ldots, N-1.$$

Сформулируем критерий монотонности разностной схемы, т.е. сформулируем условия, при которых разностная схема удовлетворяет разностному принципу максимума.

Пусть в разностной схеме (13.18), (13.21) $\varphi_i \ge 0$ для всех $i = 1, 2, \ldots, N-1$ (или же $\varphi_i \le 0$ для $i = 1, 2, \ldots, N-1$). Тогда при выполнении условий

$$\gamma_i \ge \alpha_i + \beta_i, \quad i = 1, 2, \ldots, N-1 \tag{13.22}$$

имеет место $y_i \ge 0, \quad i = 1, 2, \ldots, N-1$ ($y_i \le 0, \quad i = 1, 2, \ldots, N-1$).

Для разностных схем (13.18), (13.21), для которых выполнены условия монотонности (13.22), доказывается сходимость в равномерной норме. Исследование базируется на применении соответствующих теорем сравнения и построении мажорантных функций.

Пусть для разностной схемы (13.18), (13.21) выполнены условия (13.22) и $w(x)$ — решение задачи

$$-\alpha_i w_{i-1} + \gamma_i w_i - \beta_i w_{i+1} = \phi_i, \quad i = 1, 2, \ldots, N-1,$$

$$w_0 = 0, \quad w_N = 0.$$

Тогда при

$$|\varphi_i| \le \phi_i, \quad i = 1, 2, \ldots, N-1$$

справедлива оценка

$$|y_i| \le w_i, \quad i = 1, 2, \ldots, N-1.$$

Функция $w(x)$ называется мажорантной функцией для решения задачи (13.18), (13.21). Если удается построить мажоранту, то это значит, что получена априорная оценка для решения задачи в $L_\infty(\omega)$:

$$\|y(x)\|_\infty \le \|w(x)\|_\infty, \tag{13.23}$$

где на множестве сеточных функций, обращающихся в нуль на $\partial\omega$,

$$\|y(x)\|_\infty \equiv \max_{x\in\omega} |y(x)|.$$

На основе рассмотрения задачи для погрешности с использованием оценки (13.23) устанавливается сходимость исследуемой разностной схемы.

Другие задачи

Среди более общих чем (13.1), (13.4) краевых задач отметим задачи для уравнения (13.5). Простейшая центральноразностностная аппроксимация члена с первой производной дает разностное уравнение

$$-(ay_{\bar{x}})_x + by_x^\circ + cy = \varphi, \quad x \in \omega, \tag{13.24}$$

где, например, $b_i = v(x_i)$. Разностная схема (13.17), (13.24) аппроксимирует краевую задачу (13.2), (13.5) со вторым порядком. Ее основной недостаток связан с тем, что эта схема монотонна только при достаточно малых шагах сетки h.

Безусловно монотонные разностные схемы для уравнения (13.5) можно построить при использовании для конвективного слагаемого аппроксимаций первого порядка направленными разностями. Вместо (13.24) рассмотрим разностное уравнение

$$-(ay_{\bar{x}})_x + b^+y_{\bar{x}} + b^-y_x + cy = \varphi, \quad x \in \omega, \tag{13.25}$$

где

$$b(x) = b^+(x) + b^-(x),$$

$$b^+(x) = \frac{1}{2}(b(x) + |b(x)|) \geq 0,$$

$$b^-(x) = \frac{1}{2}(b(x) - |b(x)|) \leq 0.$$

К сожалению, схема (13.17), (13.25) имеет только первый порядок аппроксимации.

При разностной аппроксимации краевой задачи для обыкновенного дифференциального уравнения четвертого порядка (13.8)–(13.11) удобно использовать расширенную сетку с дополнительными (фиктивными) узлами $x_{-1} = -h$, $x_{N+1} = l + h$. Тогда дифференциальному уравнению (13.8) можно сопоставить разностное уравнение

$$y_{\bar{x}x\bar{x}x} = \varphi(x), \quad x \in \omega. \tag{13.26}$$

Аппроксимация краевых условий (13.9) и (13.10) дает

$$y_0 = \mu_1, \quad y_N = \mu_2, \tag{13.27}$$

$$\frac{y_1 - y_{-1}}{2h} = \nu_1, \quad \frac{y_{N+1} - y_{N-1}}{2h} = \nu_2. \tag{13.28}$$

При вычислительной реализации значения в фиктивных и граничных узлах находятся из (13.27), (13.28) непосредственно, а для определения приближенного решения в узлах $x \in \omega$ из (13.26) получим пятидиагональную систему линейных алгебраических уравнений.

Решение сеточных уравнений

Для нахождения приближенного решения краевой задачи для обыкновенного дифференциального уравнения необходимо решить соответствующую систему линейных алгебраических уравнений. Для нахождения разностного решения используются традиционные прямые методы линейной алгебры. Излагаемый метод прогонки (алгоритм Томаса), как хорошо известно, является классическим методом Гаусса для матриц специальной ленточной структуры.

Для примера рассмотрим разностное трехточечное уравнение (13.21) с однородными условиями (13.18). В подобном виде записываются и разностные схемы для задачи с краевыми условиями третьего рода (на расширенной сетке с $y_{-1} = 0$, $y_{N+1} = l + h$. В матричном виде рассматриваемая разностная задача имеет вид

$$Ay = \varphi, \quad x \in \omega,$$

где

$$A = \begin{bmatrix} \gamma_1 & -\beta_1 & 0 & & 0 \\ -\alpha_2 & \gamma_2 & -\beta_2 & & 0 \\ 0 & -\alpha_3 & \gamma_3 & & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & \gamma_{N-1} \end{bmatrix}.$$

Для нахождения решения сеточной задачи используются следующие рекуррентные формулы для вычисления прогоночных коэффициентов (прямая прогонка):

$$\xi_{i+1} = \frac{\beta_i}{\gamma_i - \alpha_i \xi_i}, \quad i = 1, 2, \dots, N-1,$$

$$\vartheta_{i+1} = \frac{\varphi_i + \alpha_i \vartheta_i}{\gamma_i - \alpha_i \xi_i}, \quad i = 1, 2, \dots, N-1$$

при

$$\xi_1 = 0, \quad \vartheta_1 = 0.$$

Для решения имеем (обратная прогонка)

$$y_i = \xi_{i+1} y_{i+1} + \vartheta_{i+1}, \quad i = 0, 1, \dots, N-1, \quad y_N = 0.$$

Пусть для системы уравнений (13.21), (13.18) выполнены условия

$$|\alpha_i| > 0, \quad |\beta_i| > 0, \quad i = 1, 2, \ldots, N-1,$$

$$|\gamma_i| \geq |\alpha_i| + |\beta_i|, \qquad i = 1, 2, \ldots, N-1.$$

Тогда алгоритм прогонки корректен, т.е. в расчетных формулах $\gamma_i - \alpha_i \xi_i \neq 0$.

В настоящее время существует ряд вариантов метода прогонки, ориентированных на определенный класс сеточных задач. Среди них отметим прогонку для задач с периодическими граничными условиями, метод прогонки для пятиточечных разностных уравнений.

13.3 Упражнения

Упражнение 13.1 *Метод стрельбы (пристрелки) при приближенном решении краевой задачи*

$$\frac{d^2u}{dx^2} = f\left(x, u, \frac{du}{dx}\right), \quad 0 < x < l,$$

$$u(0) = \mu_1, \quad u(l) = \mu_2$$

основан на решении задач Коши для системы уравнений первого порядка

$$\frac{du}{dx} = v, \quad \frac{dv}{dx} = f(x, u, v), \quad 0 < x < l$$

с начальными условиями

$$u(0) = \mu_1, \quad v(0) = \theta.$$

Величина θ подбирается так, чтобы выполнялось граничное условие $u(l) = \mu_2$. Напишите программу, которая реализует этот метод при решении краевой задачи для нелинейного уравнения второго порядка. Для нахождения параметра пристрелки θ используйте метод бисекции для решения соответствующего нелинейного уравнения, а для решения задачи Коши для системы обыкновенных дифференциальных уравнений — явный метод Рунге—Кутта четвертого порядка точности. Продемонстрируйте работоспособность этой программы при решении краевой задачи

$$\frac{d^2u}{dx^2} = 100u(u-1), \quad 0 < t < 1,$$

$$u(0) = 0, \quad u(1) = 2$$

на различных сетках.

В модуле shooting функция `shooting()` реализует решение краевой задачи на основе решения задач Коши для системы обыкновенных дифференциальных уравнений методом Рунге—Кутта четвертого порядка с использованием модуля rungeKutta. Решение нелинейной задачи для параметра пристрелки проводится с использованием функция `bisection()` модуля bisection.

Модуль shooting

```
import numpy as np
from bisection import bisection
from rungeKutta import rungeKutta
def shooting(f, mu1, mu2, a, b, n, theta1, theta2):
    """
    Solution of the Dirichlet problem for
    second-order equation by the method of shooting.
    Bisection method for finding the boundary conditions.
    Runge-Kutta fourth order for the solution of the Cauchy problem.
    """
    h = (b - a) / n
    # Initial values
    def y0(theta):
        return np.array([mu1, theta])
    # Boundary condition residual
    def r(theta):
        x, y = rungeKutta(f, a, y0(theta), b, h)
        yb = y[len(y) - 1]
        r = yb[0] - mu2
        return r
    theta = bisection(r, theta1, theta2)
    x, y = rungeKutta(f, a, y0(theta), b, h)
    return x, y[:, 0]
```

Решение модельной задачи обеспечивается следующей программой.

exer-13-1.py

```
import numpy as np
import matplotlib.pyplot as plt
from shooting import shooting
def f(x,y):
    f = np.zeros((2),'float')
    f[0] = y[1]
    f[1] = 100*(y[0]-1)*y[0]
    return f
mu1 = 0.
mu2 = 2.
a = 0.
b = 1.
theta1 = 0.
theta2 = 10.
nList = [5, 10, 100]
sglist = ['-', '--', ':']
```

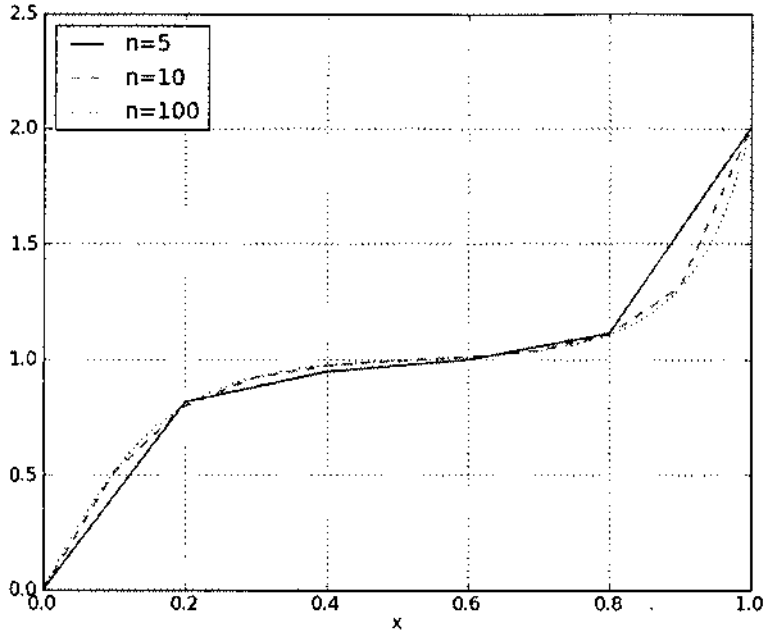

Рис. 13.1 Решение нелинейной краевой задачи

```
for k in range(len(nList)):
    n = nList[k]
    x, y = shooting(f, mu1, mu2, a, b, n, theta1, theta2)
    sl = 'n=' + str(n)
    sg = sglist[k]
    plt.plot(x, y, sg, label=sl)
plt.xlabel('x')
plt.grid(True)
plt.legend(loc=2)
plt.show()
```

Сходимость приближенного решения к точному иллюстрируется рис. 13.1, на котором приведены результаты расчетов при разбиении расчетного интервала на $n = 5, 10, 100$ частей.

Упражнение 13.2 *Напишите программу для приближенного решения разностным методом задачи конвекции-диффузии:*

$$-\frac{d^2u}{dx^2} + v(x)\frac{du}{dx} = f(x), \quad 0 < x < l, \quad u(0) = \mu_1, \quad u(l) = \mu_2.$$

Для аппроксимации конвективного слагаемого используйте центрально-разностные аппроксимации и аппроксимации направленными разностями. С помощью этой программы решите краевую задачу

$$-\frac{d^2u}{dx^2} + 100\frac{du}{dx} = 0, \quad 0 < t < 1,$$

$$u(0) = 0, \quad u(1) = 1$$

на равномерных сетках с $h = 0.1, 0.01$.

В модуле fdm функция fdm() дает сеточное решение краевой задачи для уравнения конвекции-диффузии на равномерной сетке при использовании двух отмеченных аппроксимаций конвективного слагаемого. Решение сеточной задачи дается функцией solveLU3() модуля lu3.

Модуль fdm

```
import numpy as np
from lu3 import solveLU3
def fdm(v, f, mu1, mu2, a, b, n, p):
    """
    Finite difference method for solving the Dirichlet problem
    for the convection-diffusion equation.
    When p = 0 we use the scheme with upwind
              difference approximation,
    when p = 1 - central-difference approximation.
    """
     h = (b - a) / n
     x = np.linspace(a, b, n+1)
     a = np.ones((n+1), 'float')
     b = np.zeros((n+1), 'float')
     c = np.zeros((n+1), 'float')
     r = np.zeros((n+1), 'float')
     if p == 0:
         for i in range(1, n):
             vp = max(0, v(x[i]))
             vm = min(0, v(x[i]))
             a[i] = 2. + (vp-vm)*h
             b[i] = - 1. + h * vm
             c[i] = - 1. - h * vp
             r[i] = f(x[i])*h**2
     else:
         for i in range(1, n):
             a[i] = 2.
             b[i] = - 1. + h * v(x[i]) / 2.
             c[i] = - 1. - h * v(x[i]) / 2.
             r[i] = f(x[i])*h**2
```

```
    r[0] = mu1
    r[n] = mu2
    y = solveLU3(a, b, c, r)
    return x, y
```

Для численного решения тестовой задачи используется следующая программа.

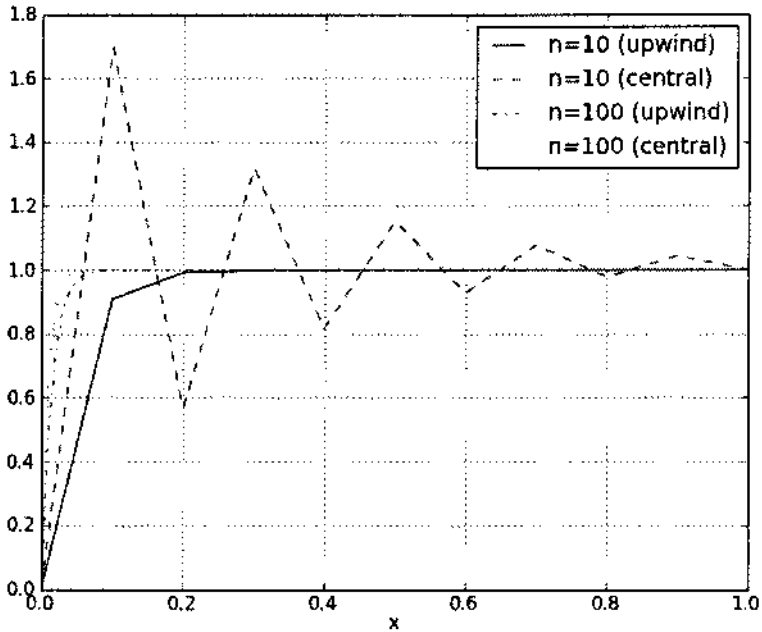

Рис. 13.2 Решение задачи конвекции-диффузии

exer−13 2 py

```
import matplotlib.pyplot as plt
from fdm import fdm
r = 100.
def f(x):
    return 0.
def v(x):
    return - r
mu1 = 0.
mu2 = 1.
a = 0.
b = 1.
nList = [10, 100]
```

```
sglist = ['-', '--', '-.', ':']
k = 0
for n in nList:
    x, y = fdm(v, f, mu1, mu2, a, b, n, 0)
    sl = 'n=' + str(n) + ' (upwind)'
    sg = sglist[k]
    k = k+1
    plt.plot(x, y, sg, label=sl)
    x, y = fdm(v, f, mu1, mu2, a, b, n, 1)
    sl = 'n=' + str(n) + ' (central)'
    sg = sglist[k]
    k = k+1
    plt.plot(x, y, sg, label=sl)
plt.xlabel('x')
plt.grid(True)
plt.legend(loc=0)
plt.show()
```

При использовании центрально-разностных аппроксимаций для конвективного слагаемого на грубых сетках (в нашем примере при $n = 10$) проявляется немонотонность (см. рис. 13.2) разностной схемы. Схема с направленными разностями является безусловно монотонной, но имеет только первый порядок точности.

13.4 Задачи

Задача 13.1 *Напишите программу для приближенного решения задачи*

$$\frac{d^2u}{dx^2} = f(x,u), \quad 0 < x < l,$$

$$u(0) = \mu_1, \quad u(l) = \mu_2$$

разностным методом на равномерной сетке при итерационном решении нелинейной сеточной задачи методом Ньютона. Используйте эту программу для решения задачи с правой частью $f(t,u) = -e^{-u}$ *при* $l = 1$ *и граничных условиях* $\mu_1 = \mu_2 = 0$.

Задача 13.2 *Напишите программу для разностного решения краевой задачи*

$$\frac{d^4u}{dx^4} = f(x), \quad 0 < x < l,$$

$$u(0) = \mu_1, \quad \frac{du}{dx}(0) = \nu_1, \quad u(l) = \mu_2 \quad \frac{du}{dx}(l) = \nu_2.$$

Для решения соответствующей сеточной задачи используйте вариант LU-разложения для пятидиагональной матрицы. С помощью этой программы

найдите решение задачи при $f(x) = 1$, $l = 1$ *и* $\mu_1 = \mu_2 = 0$, $\nu_1 = \nu_2 = 0$ *на различных сетках.*

Задача 13.3 *Разностная схема*

$$-(ay_{\bar{x}})_x = \varphi(x), \quad x \in \omega,$$

в которой

$$a_i = \left(\frac{1}{h} \int\limits_{x_{i-1}}^{x_i} \frac{dx}{k(x)} \right)^{-1},$$

$$\varphi_i = \frac{1}{h^2} \left(a_{i+1} \int\limits_{x_i}^{x_{i+1}} \frac{dx}{k(x)} \int\limits_{x_i}^{x} f(s)ds + a_i \int\limits_{x_{i-1}}^{x_i} \frac{dx}{k(x)} \int\limits_{x}^{x_i} f(s)ds \right)$$

для приближенного решения задачи

$$-\frac{d}{dx}\left(k(x)\frac{du}{dx}\right) = f(x), \quad 0 < x < l,$$

$$u(0) = \mu_1, \quad u(l) = \mu_2$$

является точной. Напишите программу разностного решения краевых задач с использованием точной разностной схемы при применении квадратурных формул трапеций для вычисления коэффициентов. Примените эту программу для решения краевой задачи (при $l = 1$ *и* $\mu_1 = \mu_2 = 0$*) с разрывным коэффициентом*

$$f(x) = \begin{cases} 1, & 0 < x < 0.5, \\ 100, & 0.5 < x < 1 \end{cases}$$

и правой частью $f(x) = 0$. *Сравните приближенное решение на последовательности трех вложенных друг в друга сетках* $h = h_0, h_0/2, h_0/4$.

Задача 13.4 *Напишите программу, которая реализует метод конечных элементов для приближенного решения краевой задачи*

$$-\frac{d}{dx}\left(k(x)\frac{du}{dx}\right) + q(x)u = f(x), \quad 0 < x < l,$$

$$u(0) = 0, \quad u(l) = 0.$$

Постройте схему конечных элементов с представлением решения в виде (13.16) на основе минимизации функционала (метод Ритца)

$$J(v) = \frac{1}{2} \int_0^l \left(k(x) \left(\frac{dv}{dx} \right)^2 + q(x)v^2(x) \right) dx - \int_0^l f(x)v(x)dx.$$

Используйте эту программу для того, чтобы найти численное решение задачи для $l = 1$ *и* $k(x) = 1, g(x) = x(x-1), f(x) = 0$ *при различных* N.

Краевые задачи для эллиптический уравнений

Среди стационарных задач математической физики наибольшее внимание уделяется краевым задачам для эллиптических уравнений второго порядка. Рассматриваются вопросы аппроксимации таких уравнений и краевых условий, формулируется принцип максимума для сеточных уравнений. Проводится исследование сходимости приближенного решения к точному в различных нормах. Отмечаются некоторые основные итерационные методы решения сеточных уравнений.

Основные обозначения

$u = u(\mathbf{x}),\ \mathbf{x} = (x_1, x_2)$	—	неизвестная функция
h_1, h_2	—	шаги равномерной сетки
ω	—	множество внутренних узлов
$\partial\omega$	—	множество граничных узлов
H	—	гильбертово пространство сеточных функций
$(\cdot,\cdot)$	—	скалярное произведение в H
$\|\cdot\|$	—	норма в H
$y_x = (y(x+h) - y(x))/h$	—	правая разностная производная в точке x
$y_{\bar{x}} = (y(x) - y(x-h))/h$	—	левая разностная производная в точке x
$y_{\mathring{x}} = \frac{1}{2}(y_x + y_{\bar{x}})$	—	центральная разностная производная в точке x
$y_{\bar{x}x} = (y_x - y_{\bar{x}})/h$	—	вторая разностная производная в точке x

14.1 Двумерные краевые задачи

Будем рассматривать двумерные краевые задачи, когда расчетная область есть прямоугольник

$$\Omega = \{\mathbf{x} \mid \mathbf{x} = (x_1, x_2),\ 0 < x_\alpha < l_\alpha,\ \alpha = 1, 2\}.$$

Основным объектом нашего исследования будет эллиптическое уравнение второго порядка

$$-\sum_{\alpha=1}^{2} \frac{\partial}{\partial x_\alpha}\left(k(\mathbf{x})\frac{\partial u}{\partial x_\alpha}\right) + q(\mathbf{x})u = f(\mathbf{x}), \quad \mathbf{x} \in \Omega. \tag{14.1}$$

На коэффициенты уравнения накладываются ограничения

$$k(\mathbf{x}) \geq \kappa > 0, \quad q(\mathbf{x}) \geq 0, \quad \mathbf{x} \in \Omega.$$

Характерным примером является уравнение Пуассона

$$-\Delta u \equiv -\sum_{\alpha=1}^{2} \frac{\partial^2 u}{\partial x_\alpha^2} = f(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{14.2}$$

т.е. в уравнении (14.1) $k(\mathbf{x}) = 1$, $q(\mathbf{x}) = 0$.

Для уравнения (14.1) будем рассматривать граничные условия первого рода

$$u(\mathbf{x}) = \mu(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega. \tag{14.3}$$

На границе области или ее части могут задаваться и граничные условия второго и третьего рода, например,

$$k(\mathbf{x})\frac{\partial u}{\partial n} + \sigma(\mathbf{x})u = \mu(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega,$$

где n -- внешняя по отношению к Ω нормаль.

14.2 Численное решение краевых задач

Приведем некоторые факты по аппроксимации краевых задач для эллиптических уравнений, сформулируем достаточные условия для выполнения принципа максимума для сеточных функций, рассмотрим вопросы оценки точности приближенного решения и проблемы решения сеточных уравнений.

Аппроксимация краевых задач для эллиптических уравнений

Будем использовать равномерную по каждому направлению сетку. Для сеток по отдельным направлениям x_α, $\alpha = 1, 2$ используем обозначения

$$\overline{\omega}_\alpha = \{x_\alpha \mid x_\alpha = i_\alpha h_\alpha, \quad i_\alpha = 0, 1, \ldots, N_\alpha, \quad N_\alpha h_\alpha = l_\alpha\},$$

где

$$\omega_\alpha = \{x_\alpha \mid x_\alpha = i_\alpha h_\alpha, \quad i_\alpha = 1, 2, \ldots, N_\alpha - 1, \quad N_\alpha h_\alpha = l_\alpha\},$$

$$\omega_\alpha^+ = \{x_\alpha \mid x_\alpha = i_\alpha h_\alpha, \quad i_\alpha = 1, 2, \ldots, N_\alpha, \quad N_\alpha h_\alpha = l_\alpha\}.$$

Для сетки в Ω положим

$$\overline{\omega} = \overline{\omega}_1 \times \overline{\omega}_2 = \{\mathbf{x} \mid \mathbf{x} = (x_1, x_2),\ x_\alpha \in \overline{\omega}_\alpha, \quad \alpha = 1, 2\},$$

$$\omega = \omega_1 \times \omega_2.$$

Для гладких коэффициентов уравнения (14.1) разностная схема строится на основе непосредственного перехода от дифференциальных операторов к разностным. Подобно одномерному случаю для краевой задачи (14.1), (14.3) поставим в соответствие разностное уравнение

$$Ly = \sum_{\alpha=1}^{2} L^{(\alpha)} y = \varphi(\mathbf{x}), \quad \mathbf{x} \in \omega, \tag{14.4}$$

где

$$L^{(\alpha)} y = -(a^{(\alpha)} y_{\bar{x}_\alpha})_{x_\alpha} + \theta_\alpha c(\mathbf{x}) y, \quad \alpha = 1, 2, \quad \mathbf{x} \in \omega, \tag{14.5}$$

где $\theta_1 + \theta_2 = 1$.

Для коэффициентов при старших производных можно положить

$$a^{(1)}(\mathbf{x}) = k(x_1 - 0,5h_1, x_2), \quad x_1 \in \omega_1^+, \quad x_2 \in \omega_2,$$

$$a^{(2)}(\mathbf{x}) = k(x_1, x_2 - 0,5h_2), \quad x_1 \in \omega_1, \quad x_2 \in \omega_2^+.$$

Для младшего коэффициента и правой части (14.4), (14.5) имеем

$$c(\mathbf{x}) = q(\mathbf{x}), \quad \varphi(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} \in \omega.$$

В общем случае применяется интегро-интерполяционный метод. Интегрирование по контрольному объему для отдельного узла $\mathbf{x}$ сетки ω

$$\Omega_\mathbf{x} = \{\mathbf{s} \mid \mathbf{s} = (s_1, s_2),\ x_1 - 0,5h_1 \le s_1 \le x_1 + 0,5h_1,$$

$$x_2 - 0,5h_2 \le s_2 \le x_2 + 0,5h_2\}$$

даст, например,

$$a^{(1)}(\mathbf{x}) = \frac{1}{h_2} \int\limits_{x_2 - 0,5h_2}^{x_2 + 0,5h_2} \left(\frac{1}{h_1} \int\limits_{x_1 - h_1}^{x_1} \frac{ds_1}{k(\mathbf{s})} \right)^{-1} ds_2,$$

$$a^{(2)}(\mathbf{x}) = \frac{1}{h_1} \int\limits_{x_1 - 0,5h_1}^{x_1 + 0,5h_1} \left(\frac{1}{h_2} \int\limits_{x_2 - h_2}^{x_2} \frac{ds_2}{k(\mathbf{s})} \right)^{-1} ds_1.$$

Для граничных узлов $\partial\omega$ ($\overline{\omega} = \omega \bigcup \partial\omega$) используется аппроксимация

$$y(\mathbf{x}) = \mu(\mathbf{x}), \quad \mathbf{x} \in \partial\omega \tag{14.6}$$

краевых условий (14.3).

Принцип максимума

Разностное уравнение (14.4), (14.5) запишем в виде

$$Sy(\mathbf{x}) = \varphi(\mathbf{x}), \quad \mathbf{x} \in \omega, \tag{14.7}$$

где линейный оператор S определяется формулой

$$Sv(\mathbf{x}) = A(\mathbf{x})v(\mathbf{x}) - \sum_{\xi \in W'(\mathbf{x})} B(\mathbf{x},\xi)v(\xi). \tag{14.8}$$

Здесь $W(\mathbf{x})$ — шаблон, а $W' \equiv W \backslash \{\mathbf{x}\}$ — окрестность узла $\mathbf{x} \in \omega$.

Будем считать, что для рассматриваемых эллиптических уравнений второго порядка шаблон W содержит узлы $(x_1 \pm h_1, x_2)$, $(x_1, x_2 \pm h_2)$ (шаблон, как минимум, пятиточечный), а коэффициенты удовлетворяют условиям

$$\begin{gathered} A(\mathbf{x}) > 0, \quad B(\mathbf{x},\xi) > 0, \quad \xi \in W'(\mathbf{x}), \\ D(\mathbf{x}) = A(\mathbf{x}) - \sum_{\xi \in W'(\mathbf{x})} B(\mathbf{x},\xi) > 0, \quad \mathbf{x} \in \omega. \end{gathered} \tag{14.9}$$

Для разностного уравнения (14.7), (14.8) при выполнении (14.9) справедлив принцип максимума. В частности, если сеточная функция $y(\mathbf{x})$, удовлетворяет граничным условиям

$$y(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\omega, \tag{14.10}$$

а правая часть

$$\varphi(\mathbf{x}) \le 0 \quad (\varphi(\mathbf{x}) \ge 0), \quad \mathbf{x} \in \omega,$$

то $y(\mathbf{x}) \le 0$ $(y(\mathbf{x}) \ge 0)$.

На основе принципа максимума устанавливаются теоремы сравнения для решений сеточных эллиптических уравнений. Рассмотрим, например, задачу

$$Sw(\mathbf{x}) = \phi(\mathbf{x}), \quad \mathbf{x} \in \omega,$$

$$w(\mathbf{x}) = \nu(\mathbf{x}), \quad \mathbf{x} \in \partial\omega$$

и пусть

$$|\varphi(\mathbf{x})| \le \phi(\mathbf{x}), \quad \mathbf{x} \in \omega,$$

$$|\mu(\mathbf{x})| \le \nu(\mathbf{x}), \quad \mathbf{x} \in \partial\omega.$$

Тогда для решения задачи (14.6), (14.7) справедлива оценка

$$|y(\mathbf{x})| \le w(\mathbf{x}), \quad \mathbf{x} \in \overline{\omega}.$$

Отсюда следует, что для решения однородного уравнения (14.6) ($\varphi(\mathbf{x}) = 0$, $\mathbf{x} \in \omega$) с граничными условиями (14.7) имеет место априорная оценка устойчивости

$$\max_{\mathbf{x}\in\omega} |y(\mathbf{x})| \le \max_{\mathbf{x}\in\partial\omega} |\mu(\mathbf{x})|.$$

С привлечением подобных априорных оценок доказывается сходимость разностных схем в равномерной норме. Будем использовать для приближенного решения задачи Дирихле для уравнения Пуассона (14.2), (14.3) разностное уравнение

$$-y_{\bar{x}_1 x_1} - y_{\bar{x}_2 x_2} = \varphi(\mathbf{x}), \quad \mathbf{x} \in \omega, \tag{14.11}$$

дополнив его граничными условиями (14.6). Для погрешности $z(\mathbf{x}) = y(\mathbf{x}) - u(\mathbf{x})$, $\mathbf{x} \in \overline{\omega}$ получим задачу

$$-z_{\bar{x}_1 x_1} - z_{\bar{x}_2 x_2} = \psi(\mathbf{x}), \quad \mathbf{x} \in \omega,$$

$$z(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\omega,$$

где $\psi(\mathbf{x}) = O(h_1^2 + h_2^2)$ — погрешность аппроксимации. Выбрав в качестве мажорантной функции

$$w(\mathbf{x}) = \frac{1}{4}(l_1^2 + l_2^2 - x_1^2 - x_2^2)\|\psi(\mathbf{x})\|_\infty,$$

где

$$\|v(\mathbf{x})\|_\infty = \max_{\mathbf{x}\in\omega} |v(\mathbf{x})|,$$

для погрешности получим оценку

$$\|y(\mathbf{x}) - u(\mathbf{x})\|_\infty \le \frac{1}{4}(l_1^2 + l_2^2)\|\psi(\mathbf{x})\|_\infty.$$

Тем самым разностная схема (14.6), (14.11) сходится в $L_\infty(\omega)$ со вторым порядком.

Разностные уравнения в гильбертовом пространстве

Остановимся на решении уравнения (14.1) с однородными граничными условиями первого рода ($\mu(\mathbf{x}) = 0$ в (14.2)), которой ставится в соответствие разностная схема (14.4), (14.5), (14.10). Для сеточных функций, обращающихся в нуль на множестве граничных узлов $\partial\omega$, определим гильбертово пространство $H = L_2(\omega)$, в котором скалярное произведение и норма задаются следующим образом:

$$(y, w) \equiv \sum_{\mathbf{x}\in\omega} y(\mathbf{x})w(\mathbf{x})h_1h_2, \quad \|y\| \equiv (y, y)^{1/2}.$$

Определим для двумерных разностных функций, обращающихся в нуль на $\partial\omega$ сеточный аналог нормы в $W_2^1(\omega)$:

$$\|\nabla y\|^2 \equiv \sum_{x_1\in\omega_1^+}\sum_{x_2\in\omega_2}(y_{\bar{x}_1})^2 h_1 h_2 + \sum_{x_1\in\omega_1}\sum_{x_2\in\omega_2^+}(y_{\bar{x}_2})^2 h_1 h_2.$$

На H разностный оператор L самосопряжен и справедлива при наших предположениях о коэффициентах уравнения (14.1) оценка

$$(Ly, y) \geq \kappa\|\nabla y\|^2. \tag{14.12}$$

Для сеточных функций $y(\mathbf{x})$, обращающихся в нуль на $\partial\omega$, верно неравенство (неравенство Фридрихса для двумерных сеточных функций)

$$\|y\|^2 \leq M_0\|\nabla y\|^2, \quad M_0^{-1} = \frac{8}{l_1^2} + \frac{8}{l_2^2}. \tag{14.13}$$

Из (14.12), (14.13) следует оценка оператора L снизу

$$L \geq \kappa\lambda_0 E, \quad \lambda_0 = M_0^{-1}. \tag{14.14}$$

Приведем также оценку оператора L сверху:

$$L \leq M_1 E \tag{14.15}$$

с постоянной

$$M_1 = \frac{4}{h_1^2}\max_{\mathbf{x}\in\omega}\frac{a^{(1)}(\mathbf{x}) + a^{(1)}(x_1+h_1, x_2)}{2} +$$

$$+\frac{4}{h_2^2}\max_{\mathbf{x}\in\omega}\frac{a^{(2)}(\mathbf{x}) + a^{(2)}(x_1, x_2+h_2)}{2} + \max_{\mathbf{x}\in\omega}|c(\mathbf{x})|.$$

Задача для погрешности разностного решения

$$z(\mathbf{x}) = y(\mathbf{x}) - u(\mathbf{x}), \quad \mathbf{x}\in\bar{\omega}$$

имеет вид

$$Lz = \psi(\mathbf{x}), \quad \mathbf{x}\in\omega,$$

где $\psi(\mathbf{x})$, как обычно, погрешность аппроксимации:

$$\psi(\mathbf{x}) = \varphi(\mathbf{x}) - Lu, \quad \mathbf{x}\in\omega.$$

Будем считать, что решение краевой задачи имеет достаточно гладкое классическое решение. На равномерной прямоугольной сетке погрешность аппроксимации в этих условиях при использовании разностного оператора (14.4), (14.5) имеет второй порядок:

$$\psi(\mathbf{x}) = O(|h|^2), \quad |h|^2 \equiv h_1^2 + h_2^2, \quad \mathbf{x}\in\omega.$$

Для рассматриваемой разностной схемы (14.4)–(14.6) справедлива априорная оценка для погрешности

$$\|\nabla z\| \le \frac{M_0^{1/2}}{\kappa}\|\psi\|.$$

В силу этого разностная схема сходится в $W_2^1(\omega)$ со вторым порядком.

Решение сеточных уравнений

Исходная дифференциальная задача при аппроксимации заменяется сеточной. Соответствующие разностные (сеточные) уравнения есть система линейных алгебраических уравнений для неизвестных значений сеточной функции. Для их нахождения используются методы линейной алгебры, которые максимально учитывают специфику сеточных задач. Особенности сеточных задач проявляются в том, что соответствующая матрица системы алгебраических уравнений является разреженной, т.е. содержит много нулевых элементов, имеет ленточную структуру. При решении многомерных задач матрица имеет очень большой порядок, равный общему числу узлов сетки.

Классических подход к решению простейших линейных задач математической физики связан с использованием метода разделения переменных. Естественно ожидать, что аналогичная идея получит свое развитие и применительно к сеточным уравнениям. Рассмотрим сеточную задачу для уравнения Пуассона (14.10) с однородными краевыми условиями (14.11).

Для применения метода Фурье для решения этой двумерной задачи рассмотрим задачу на собственные значения для разностного оператора второй производной по переменной x_1:

$$-v_{\bar{x}_1 x_1} + \lambda v = 0, \quad x_1 \in \omega_1,$$

$$v_0 = 0, \quad v_{N_1} = 0.$$

Соответствующие собственные значения и собственные функции обозначим λ_k, $v^{(k)}(x_1)$, $k = 1, 2, \ldots, N_1 - 1$:

$$\lambda_k = \frac{4}{h_1^2}\sin^2\frac{k\pi h_1}{2l_1},$$

$$v^{(k)}(x_1) = \sqrt{\frac{2}{l_1}}\sin\frac{k\pi x_1}{l_1}, \quad k = 1, 2, \ldots, N_1 - 1.$$

Будем искать приближенное решение задачи (14.10), (14.11) в виде разложения:

$$y(\mathbf{x}) = \sum_{k=1}^{N_1-1} c^{(k)}(x_2)v^{(k)}(x_1), \quad \mathbf{x} \in \omega. \tag{14.16}$$

Пусть $\varphi^{(k)}(x_2)$ -- коэффициенты Фурье правой части:

$$\varphi^{(k)}(x_2) = \sum_{k=1}^{N_1-1} \varphi(\mathbf{x}) v^{(k)}(x_1) h_1. \tag{14.17}$$

Для определения $c^{(k)}(x_2)$ получим трехточечные задачи:

$$-c^{(k)}_{\bar{x}_2 x_2} - \lambda c^{(k)} = \varphi^{(k)}(x_2), \quad x_2 \in \omega_2, \tag{14.18}$$

$$c^{(k)}_0 = 0, \quad c^{(k)}_{N_2} = 0. \tag{14.19}$$

Разностная задача (14.18), (14.19) при каждом $k = 1, 2, \ldots, N_1 - 1$ решается методом прогонки.

Таким образом метод Фурье основан на определении собственных функций и собственных значений одномерной сеточной задачи, вычислении коэффициентов Фурье правой части согласно (14.17), решении задач (14.18), (14.19) для коэффициентов разложения и, наконец, решение задачи определяется по формулам суммирования (14.16).

Эффективные вычислительные алгоритмы метода разделения переменных связаны с быстрым преобразованием Фурье (FFT). В этом случае можно вычислить коэффициенты Фурье правой части и восстановить решение при затратах $Q = O(N_1 N_1 \log N_1)$. Для задач с постоянными коэффициентами можно использовать преобразование Фурье по обоим переменным (разложение по собственным функциям двумерного сеточного оператора L).

Для приближенного решения многомерных сеточных эллиптических задач с переменными коэффициентами используются итерационные методы. Основные понятия теории итерационных методов решения систем линейных уравнений обсуждались выше. Здесь мы отметим только наиболее важные особенности итерационного решения краевых задач для эллиптических уравнений, которые касаются выбора оператора B (переобуславливателя) при переходе на новое итерационное приближение.

Для разностной задачи (14.4), (14.5), (14.10) запишем соответствующую систему линейных уравнений

$$Ay = \varphi \tag{14.20}$$

для нахождения сеточного решения $y(\mathbf{x})$, $\mathbf{x} \in \omega$. Здесь A рассматривается как линейный оператор, действующий в конечномерном гильбертовом пространстве $H = L_2(\omega)$, а $\varphi(\mathbf{x})$ -- заданный элемент H.

Для приближенного решения уравнения (14.20) с

$$A = A^* > 0$$

будем использовать двухслойный итерационный метод

$$B\frac{y^{k+1} - y^k}{\tau_{k+1}} + Ay^k = \varphi, \quad k = 0, 1, \ldots. \tag{14.21}$$

Особенности итерационных методов для решения эллиптических задач проявляются при построении оператора B.

Пусть априорная информация об операторах B и A задана в виде двухстороннего операторного неравенства

$$\gamma_1 B \le A \le \gamma_2 B, \quad \gamma_1 > 0, \tag{14.22}$$

т.е. операторы B и A энергетически эквивалентны с постоянными энергетической эквивалентности γ_α, $\alpha = 1, 2$. В итерационном методе (14.21) при оптимальным значением итерационного параметра

$$\tau = \tau_0 = \frac{2}{\gamma_1 + \gamma_2}$$

для числа итераций K, необходимых для достижения точности ε, справедлива оценка

$$K \ge K_0(\varepsilon) = \frac{\ln \varepsilon}{\ln \varrho_0}, \tag{14.23}$$

где

$$\varrho_0 = \frac{1-\xi}{1+\xi}, \quad \xi = \frac{\gamma_1}{\gamma_2}.$$

При использовании чебышевского набора итерационных параметров и для метода сопряженных градиентов имеем

$$K \ge K_0(\varepsilon) = \frac{\ln(2\varepsilon^{-1})}{\ln \varrho_1^{-1}}, \tag{14.24}$$

где

$$\varrho_1 = \frac{1-\xi^{1/2}}{1+\xi^{1/2}}, \quad \xi = \frac{\gamma_1}{\gamma_2}.$$

Для явного итерационного метода $B = E$ и в силу (14.14), (14.15) ($A = L$) для постоянных энергетической эквивалентности получим

$$\gamma_1 = \kappa M_0^{-1} = O(1), \quad \gamma_2 = M_1 = O(|h|^{-2}).$$

В методе простой итерации при оптимальном значении итерационного параметра из (14.23) получим

$$K_0(\varepsilon) = O\left(\frac{1}{|h|^2} \ln \frac{1}{\varepsilon}\right).$$

Для метода сопряженных градиентов оценка (14.24) дает

$$K_0(\varepsilon) = O\left(\frac{1}{|h|} \ln \frac{1}{\varepsilon}\right). \tag{14.25}$$

При применении попеременно-треугольного итерационного метода используется разложение

$$A = A_1 + A_2 = A^* > 0, \quad A_2^* = A_1$$

и оператор B задается в виде

$$B = (G + \nu A_1)G^{-1}(G + \nu A_2), \tag{14.26}$$

где $G = G^* > 0$. При априорной информации

$$A \geq \delta G, \quad \delta > 0, \quad A_1 G^{-1} A_2 \leq \frac{\Delta}{4} A \tag{14.27}$$

оптимальным является выбор параметра

$$\nu = \nu_0 = \frac{2}{\sqrt{\Delta\delta}},$$

причем для числа итераций верна оценка

$$K \geq K_0(\varepsilon) = \frac{1}{2\sqrt{2}\sqrt[4]{\eta}} \ln \frac{2}{\varepsilon}, \quad \eta = \frac{\delta}{\Delta} \tag{14.28}$$

при использовании чебышевского набора итерационных параметров или метода сопряженных градиентов.

Для эллиптических уравнений второго порядка имеет место следующая зависимость от шагов сетки

$$\delta = O(1), \quad \Delta = O(|h|^{-2}).$$

Поэтому для числа итераций попеременно-треугольного итерационного метода получим

$$K_0(\varepsilon) = O\left(\frac{1}{\sqrt{|h|}} \ln \frac{1}{\varepsilon}\right).$$

Оптимизация метода достигается за счет выбора оператора $D = D^* > 0$.

14.3 Упражнения

Упражнение 14.1 *Напишите программу численного решения на равномерной по каждому направлению сетке задачи Дирихле для уравнения Пуассона в прямоугольнике методом разделения переменных. Для прямого и обратного преобразований Фурье используйте возможности пакета NumPy. Продемонстрируйте работоспособность этой программы при решении краевой задачи*

$$-\sum_{\alpha=1}^{2} \frac{\partial^2 u}{\partial x_\alpha^2} = \frac{32}{l_1^2 l_2^2}(x_1(l_1 - x_1) + x_2(l_2 - x_2)), \quad \mathbf{x} \in \Omega,$$

$$u(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\Omega.$$

Приближенное решение соответствующей сеточной эллиптической задачи получим на основе алгоритма (14.16)–(14.19). Синус-преобразование Фурье проводится с помощью функций `fft.rfft()` (прямое преобразование) и

fft.irfft() (обратное преобразование) пакета NumPy. Исходная вещественная сеточная функция продлевается нечетным образом, к которой затем применяется дискретное преобразование Фурье. В модуле fftPoisson функция fftPoisson() реализует численное решение сеточной эллиптической задачи разделением переменных с преобразованием Фурье по первой переменной. Решение систем уравнений с трехдиагональной матрицей обеспечивается функцией solveLU3() модуля lu3.

Модуль fftPoisson

```
import numpy as np
import math as mt
from lu3 import solveLU3
def fftPoisson(f, l1, l2, n1, n2):
    """
    Numerical Solution of the Dirichlet problem
    for Poisson equation in a rectangle.
    Fast Fourier transform in the variable x.
    """
    h1 = l1 / n1
    h2 = l2 / n2
    # Fourier coefficients of the right side.
    y = np.zeros((n1+1, n2+1), 'float')
    r = np.zeros((2*n1), 'float')
    tt = np.zeros((n1+1), 'cfloat')
    for j in range(1,n2):
        for i in range(1,n1):
            r[i] = f(i*h1, j*h2)
            r[2*n1-i] = - r[i]
        rt = np.fft.rfft(r).imag
        y[0:n1+1, j] = rt[0:n1+1]
    # Fourier coefficients for the solution.
    a = np.ones((n2+1), 'float')
    b = np.zeros((n2+1), 'float')
    c = np.zeros((n2+1), 'float')
    q = np.zeros((n2+1), 'float')
    for i in range(1,n1):
        for j in range(1,n2):
            a[j] = 2. + (2.*mt.sin(i*mt.pi/(2*n1))/h1)**2*h2**2
            b[j] = - 1.
            c[j] = - 1.
            q[j] = y[i,j]*h2**2
        p = solveLU3(a, b, c, q)
        y[i,:] = p
    # Inverse Fourier transform.
    for j in range(1,n2):
```

```
        for i in range(1,n1):
            tt[i] = y[i,j]*1.j
        yt = np.fft.irfft(tt)
        y[0:n1, j] = yt[0:n1]
    return y
```

Решение модельной задачи с точным решением

$$u(\mathbf{x}) = \frac{16}{l_1^2 l_2^2} x_1(l_1 - x_1)x_2(l_2 - x_2)$$

обеспечивается следующей программой.

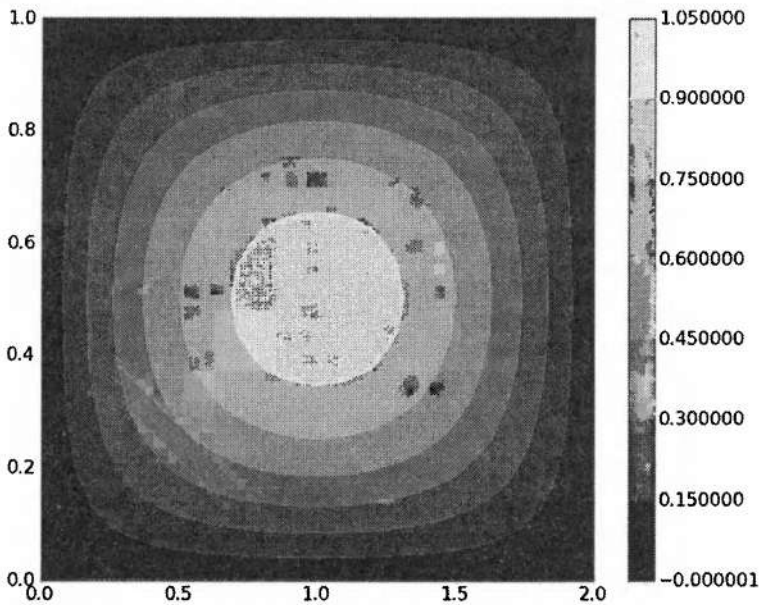

Рис. 14.1 Решение задачи для уравнения Пуассона

exer −14 1 py

```
import numpy as np
from fftPoisson import fftPoisson
import matplotlib.pyplot as plt
l1 = 2.
l2 = 1.
def f(x,y):
    return 32.*(x*(l1-x)+y*(l2-y))/(l1*l2)**2
n1 = 32
```

```
n2 = 16
y = fftPoisson(f, l1, l2, n1, n2).T
print 'max y =', np.amax(y)
x1 = np.linspace(0., l1, n1+1)
x2 = np.linspace(0., l2, n2+1)
X1, X2 = np.meshgrid(x1, x2)
plt.contourf(X1, X2, y, cmap=plt.cm.gray)
plt.colorbar()
plt.show()
```

```
max y = 1.0
```

На рис. 14.1 показано численное решение при $l_1 = 2, l_2 = 1$ сетке (33×17).

Упражнение 14.2 *Напишите программу численного решения на равномерной по каждому направлению сетке задачи Дирихле для уравнения конвекции-диффузии*

$$-\sum_{\alpha=1}^{2} \frac{\partial^2 u}{\partial x_\alpha^2} + b(\mathbf{x}) \frac{\partial u}{\partial x_1} = f(\mathbf{x}), \quad \mathbf{x} \in \Omega,$$

$$u(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\Omega.$$

Используйте аппроксимации центральными разностями для конвективного слагаемого и итерационный метод релаксации. С помощью этой программы исследуйте зависимость скорости сходимости от параметра релаксации при численном решении задачи с $f(\mathbf{x}) = 1$ *и* $b(\mathbf{x}) = 0,10$.

В сеточной задаче

$$Ay = f$$

представим матрицу A в виде суммы диагональной, нижней треугольной и верхней треугольных матриц

$$A = D + L + U.$$

Метод релаксации соответствует использованию итерационного метода

$$(D + \omega L)\frac{y^{k+1} - y^k}{\omega} + Ay^k = f.$$

При $\omega > 1$ говорят о верхней релаксации, при $\omega < 1$ — о нижней релаксации.

Приближенное решение эллиптической задачи с использованием заданного параметра релаксации ω реализуется функцией `relaxation()` модуля relaxation.

Модуль `relaxation`

```
import numpy as np
import math as mt
def relaxation(b, f, l1, l2, n1, n2, omega, tol = 1.e-8):
```

```
    """
    Numerical Solution of the Dirichlet problem
    for convection-diffusion equation in a rectangle.
    Method of relaxation.
    """
    h1 = l1 / n1
    h2 = l2 / n2
    d = 2. / h1**2 + 2. / h2**2
    y = np.zeros((n1+1, n2+1), 'float')
    ff = np.zeros((n1+1, n2+1), 'float')
    bb = np.zeros((n1+1, n2+1), 'float')
    for j in range(1,n2):
        for i in range(1,n1):
            ff[i,j] = f(i*h1, j*h2)
            bb[i,j] = b(i*h1, j*h2)
    # the maximum number of iterations is 10000
    for k in range(1,10001):
        rn = 0.
        for j in range(1,n2):
            for i in range(1,n1):
               rr = - (y[i-1,j] - 2.*y[i,j] + y[i+1,j]) / h1**2 \
                - (y[i,j-1] - 2.*y[i,j] + y[i,j+1]) / h2**2 \
                + bb[i,j]*(y[i+1,j] - y[i-1,j]) / (2.*h1) - ff[i,j]
               rn = rn + rr**2
               y[i,j] = y[i,j] - omega * rr / d
        rn = rn*h1*h2
        if rn < tol**2: return y, k
    print 'Relaxation method failed to converge:'
    print 'after 10000 iteration residual =', mt.sqrt(rn)
```

Решение модельной задачи в единичном квадрате на сетке $h_1 = h_2 = 0.04$ обеспечивается следующей программой.

exer −14.2.py

```
import numpy as np
from relaxation import relaxation
import matplotlib.pyplot as plt
bcList = [0., 10.]
sglist = ['-', '--']
kk = 0
for bc in bcList:
    l1 = 1.
    l2 = 1.
    def f(x,y):
        return 1.
```

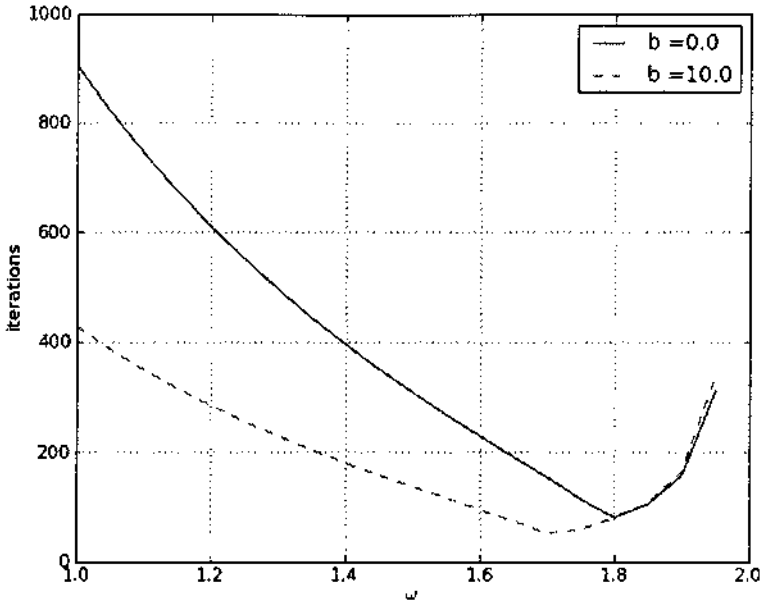

Рис. 14.2 Число итераций метода релаксации

```
    def b(x,y):
        return bc
    n1 = 25
    n2 = 25
    m = 20
    om = np.linspace(1., 1.95, m)
    it = np.zeros((m), 'float')
    for k in range(m):
      omega = om[k]
      y, iter = relaxation(b, f, l1, l2, n1, n2, omega, tol=1.e-6)
      it[k] = iter
    sl = 'b =' + str(bc)
    sg = sglist[kk]
    kk = kk+1
    plt.plot(om, it, sg, label = sl)
plt.xlabel('$\\omega$')
plt.ylabel('iterations')
plt.legend(loc=0)
plt.grid(True)
plt.show()
```

На рис. 14.2 показана зависимость числа итераций от параметра релаксации для уравнения Пуассона ($b(\mathbf{x}) = 0$) и уравнения конвекции-диффузии ($b(\mathbf{x}) = 10$). Для сеточного уравнения Пуассона оптимальное значение параметра релаксации находится аналитически, а итерационный метод сходиться при $0 < \omega < 2$.

14.4 Задачи

Задача 14.1 *Напишите программу численного решения на равномерной по каждому направлению сетке задачи Дирихле для уравнения Пуассона в прямоугольнике методом разделения переменных с использованием быстрого преобразования Фурье по обоим переменным. С помощью этой программы найдите численное решение на последовательности сгущающихся сеток для задачи в единичном квадрате с точным решением*

$$u(\mathbf{x}) = e^{x-y} \quad \mathbf{x} \in \partial\Omega.$$

Задача 14.2 *Напишите программу для приближенного решения уравнение Пуассона в круге*

$$-\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right) - \frac{1}{r^2}\frac{\partial^2 u}{\partial \varphi^2} = f(r,\varphi), \quad 0 < r < R, \quad 0 \le \varphi \le 2\pi$$

с граничными условиями

$$\lim_{r\to 0} r\frac{\partial u}{\partial r}(r,\varphi) = 0, \quad u(R,\varphi) = 0$$

и условиями периодичности

$$u(r,\varphi) = u(r,2\pi+\varphi), \quad 0 < r < R, \quad 0 \le \varphi < 2\pi$$

Используйте равномерную по каждому направлению сетку с преобразованием Фурье по углу. Работоспособность программы продемонстрируйте на решении задачи с $f(r,\varphi) = r\sin(\varphi), R = 1$.

Задача 14.3 *Напишите программу численного решения на равномерной по каждому направлению сетке задачи*

$$-\sum_{\alpha=1}^{2}\frac{\partial}{\partial x_\alpha}\left(k(\mathbf{x})\frac{\partial u}{\partial x_\alpha}\right) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega$$

$$u(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\Omega$$

в прямоугольнике Ω. Используйте итерационный метод Якоби ($B = D$) и выбор итерационных параметров по методу сопряженных градиентов. С помощью этой программы решите краевую задачу в единичном квадрате с

$f(\mathbf{x}) = 1$ *и*

$$k(\mathbf{x}) = \begin{cases} 1, & (x_1 - 0.5)^2 + (x_2 - 0.5)^2 > 0.125, \\ \chi, & (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \le 0.125 \end{cases}$$

при $\chi = 10^{-2}, 1, 10^2$.

Задача 14.4 *Напишите программу численного решения на равномерной по каждому направлению сетке задачи*

$$-\sum_{\alpha=1}^{2} \frac{\partial^2 u}{\partial x_\alpha^2} + q(\mathbf{x})u = f(\mathbf{x}), \quad \mathbf{x} \in \Omega$$

$$u(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\Omega$$

в прямоугольнике Ω. *Используйте попеременно-треугольный итерационный метод с*

$$B = (E + \nu A_1)E + \nu A_2), \quad A = A_1 + A_2, \quad A_1 = A_2^*$$

и с выбором итерационных параметров по методу сопряженных градиентов. Параметр ν *задается равным*

$$\nu = \nu_0 = \frac{2}{\sqrt{\Delta\delta}},$$

где

$$\delta = \frac{4}{h_1^2}\sin^2\frac{\pi h_1}{2l_1} + \frac{4}{h_2^2}\sin^2\frac{\pi h_2}{2l_2},$$

$$\Delta = \frac{4}{h_1^2} + \frac{4}{h_2^2}$$

и является оптимальным для сеточного оператора Лапласа. Возможности программы продемонстрируйте на численном решении краевой задачу в единичном квадрате с $f(\mathbf{x}) = 1$ *и*

$$q(\mathbf{x}) = \begin{cases} 1, & (x_1 - 0.5)^2 + (x_2 - 0.5)^2 > 0.125, \\ \chi, & (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \le 0.125 \end{cases}$$

при $\chi = 0, 10, 10^2$.

Нестационарные задачи математической физики

Рассматриваются разностные методы приближенного решения краевых задач для нестационарных уравнений с частными производными. Основное внимание уделяется построению и исследованию разностных схем для параболических уравнений второго порядка. Теоретической основой при исследовании сходимости разностных схем является общая теория устойчивости операторно-разностных схем. Приведены основные результаты об устойчивости двух- и трехслойных разностных схем по начальным данным и правой части. Отмечаются особенности исследования схем для гиперболических уравнений второго порядка. Строятся экономичные разностные схемы для приближенного решения многомерных нестационарных задач математической физики.

Основные обозначения

$u = u(x,t)$	—	неизвестная функция
h	—	шаг равномерной сетки по x
τ	—	шаг сетки по времени t
ω_h	—	множество внутренних узлов по пространству
$\partial\omega_h$	—	множество граничных узлов
H	—	гильбертово пространство сеточных функций
$(\cdot,\cdot)$	—	скалярное произведение в H
$\|\cdot\|$	—	норма в H
$y_x = (y(x+h) - y(x))/h$	—	правая разностная производная в точке x
$y_{\bar{x}} = (y(x) - y(x-h))/h$	—	левая разностная производная в точке x
y^n	—	решение на момент времени $t = t_n$
$y_t = (y^{n+1} - y^n)/\tau$	—	производная вперед по переменной t
$y_{\bar{t}} = (y^n - y^{n-1})/\tau$	—	производная назад по переменной t

15.1 Нестационарные краевые задачи

В качестве базового нестационарного уравнения математической физики выступает одномерное параболическое уравнение второго порядка. В прямоугольнике

$$\overline{Q}_T = \overline{\Omega} \times [0,T], \quad \overline{\Omega} = \{x \mid 0 \le x \le l\}, \quad 0 \le t \le T,$$

рассматривается уравнение

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right) + f(x,t), \quad 0 < x < l, \quad 0 < t \le T. \tag{15.1}$$

Оно дополняется (первая краевая задача) граничными

$$u(0,t) = 0, \quad u(l,t) = 0, \quad 0 < t \le T, \tag{15.2}$$

и начальными

$$u(x,0) = v^0(x), \quad 0 \le x \le l, \tag{15.3}$$

условиями.

Для простоты изложения мы ограничились однородными граничными условиями и зависимостью коэффициента k только от пространственной переменной, причем $k(x) \ge \kappa > 0$.

Вместо условий первого рода (15.2) могут задаваться другие граничные условия. Например, во многих прикладных задачах необходимо ориентироваться на формулирование граничных условий третьего рода:

$$\begin{aligned} &-k(0)\frac{du}{dx}(0,t) + \sigma_1(t)u(0,t) = \mu_1(t), \\ &k(l)\frac{du}{dx}(l,t) + \sigma_2(t)u(l,t) = \mu_2(t), \quad 0 < t \le T. \end{aligned} \tag{15.4}$$

Среди других нестационарных краевых задач необходимо выделить гиперболическое уравнение второго порядка. В одномерном по пространству случае ищется решение уравнения

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right) + f(x,t), \quad 0 < x < l, \quad 0 < t \le T. \tag{15.5}$$

Для однозначного определения решения этого уравнения помимо граничных условий (15.2) задаются два начальных условия

$$u(x,0) = u_0(x), \quad \frac{\partial u}{\partial t}(0,t) = u_1(x), \quad 0 \le x \le l. \tag{15.6}$$

Особое внимание необходимо уделять методам численного решения многомерных нестационарных задач математической физики. Примером может

служить двумерное параболическое уравнение. Будем искать в прямоугольнике

$$\Omega = \{\mathbf{x} \mid \mathbf{x} = (x_1, x_2),\ 0 < x_\alpha < l_\alpha,\ \alpha = 1, 2\}$$

функцию $u(\mathbf{x}, t)$, удовлетворяющую уравнению

$$\frac{\partial u}{\partial t} = \sum_{\alpha=1}^{2} \frac{\partial}{x_\alpha}\left(k(\mathbf{x})\frac{\partial u}{x_\alpha}\right) - q(\mathbf{x}, t)u + f(\mathbf{x}, t), \quad \mathbf{x} \in \Omega, \quad 0 \le t \le T \tag{15.7}$$

и условиям

$$u(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \partial\Omega \quad 0 < t \le T, \tag{15.8}$$

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}), \qquad \mathbf{x} \in \Omega. \tag{15.9}$$

Аналогично формулируются и другие нестационарные многомерные краевые задачи для уравнений с частными производными.

15.2 Разностные методы решения нестационарных задач

Прежде чем проводить исследование конкретных разностных схем для нестационарных уравнений математической физики введем базовые понятия теории устойчивости операторно-разностных схем, рассматриваемых в конечномерных гильбертовых пространствах. На основе приведенных оценок устойчивости двух- и трехслойных разностных схем по начальным данным и правой части проводится исследование разностных схем для уравнений параболического и гиперболического типов.

Устойчивость двухслойных операторно-разностных схем

Дадим некоторые основные понятия и определения общей теории устойчивости операторно-разностных схем. Пусть задано вещественное конечномерное гильбертово пространство H и сетка по времени

$$\bar{\omega}_\tau = \omega_\tau \cup \{T\} = \{t_n = n\tau, \quad n = 0, 1, ..., N_0; \quad \tau N_0 = T\}.$$

Обозначим через $A, B : H \to H$ линейные операторы в H и пусть они, для простоты, не зависят от τ, t_n. Рассмотрим задачу Коши для операторно-разностного уравнения

$$B\frac{y^{n+1} - y^n}{\tau} + Ay^n = \varphi^n, \quad t_n \in \omega_\tau, \tag{15.10}$$

$$y^0 = u_0, \tag{15.11}$$

где $y^n = y(t_n) \in H$ — искомая функция, а $\varphi^n, u_0 \in H$ — заданы.

Определим двухслойную разностную схему как множество задач Коши (15.10), (15.11), зависящих от параметра τ, а запись (15.10) (15.11) будем называть канонической формой двухслойных схем.

Двухслойная схема называется устойчивой, если существуют такие положительные постоянные m_1 и m_2, не зависящие от τ и выбора u_0, φ, что при любых $u_0 \in H$, $\varphi \in H$, $t \in \bar{\omega}_\tau$ для решения задачи (15.10), (15.11) справедлива оценка

$$\|y^{n+1}\| \le m_1\|u_0\| + m_2 \max_{0\le k\le n} \|\varphi^k\|_*, \quad t_n \in \omega_\tau, \tag{15.12}$$

где $\|\cdot\|$ и $\|\cdot\|_*$ — некоторые нормы в пространстве H.

Неравенство (15.12) отражает свойство непрерывной зависимости решения задачи (15.10), (15.11) от входных данных. Обычно разделяют понятия устойчивости по начальным данным и устойчивости по правой части.

Разностная схема

$$B\frac{y^{n+1}-y^n}{\tau} + Ay^n = 0, \quad t_n \in \omega_\tau, \tag{15.13}$$

$$y^0 = u_0 \tag{15.14}$$

называется устойчивой по начальным данным, если для решения задачи (15.13), (15.14) выполняется оценка

$$\|y^{n+1}\| \le m_1\|u_0\|, \quad t_n \in \omega_\tau. \tag{15.15}$$

Двухслойная разностная схема

$$B\frac{y^{n+1}-y^n}{\tau} + Ay^n = \varphi^n, \quad t_n \in \omega_\tau, \tag{15.16}$$

$$y^0 = 0 \tag{15.17}$$

устойчива по правой части, если для решения выполняется неравенство

$$\|y^{n+1}\| \le m_2 \max_{0\le k\le n} \|\varphi^k\|_*, \quad t_n \in \omega_\tau. \tag{15.18}$$

Получение оценок устойчивости чаще всего базируется на априорных оценках разностного решения при переходе с одного временного слоя на другой. Для самосопряженного положительного оператора R через H_R обозначим гильбертово пространство, состоящее из элементов пространства H и снабженное скалярным произведением и нормой

$$(y,w)_R = (Ry,w), \quad \|y\|_R = \sqrt{(Ry,y)}.$$

Разностная схема (15.13), (15.14) называется ϱ-устойчивой (равномерно устойчивой) по начальным данным в H_R, если существуют постоянная $\varrho > 0$ и

постоянная m_1, не зависящая от τ, n, такие, что при любых n и при всех $y^n \in H$ для решения y^{n+1} разностного уравнения (15.13) справедлива оценка

$$\|y^{n+1}\|_R \le \varrho \|y^n\|_R, \quad t_n \in \omega_\tau, \tag{15.19}$$

причем $\varrho^n \le m_1$.

В теории разностных схем в качестве константы ϱ выбирается обычно одна из величин

$$\varrho = 1,$$
$$\varrho = 1 + c\tau, \quad c > 0,$$
$$\varrho = \exp(c\tau),$$

где постоянная c не зависит от τ, n.

Из оценки разностного решения на слое

$$\|y^{n+1}\| \le \varrho \|y^n\| + \tau \|\varphi^n\|_*$$

следует априорная оценка (разностный аналог леммы Гронуолла)

$$\|y_{n+1}\| \le \varrho^{n+1} \|y_0\| + \sum_{k=0}^{n} \tau \varrho^{n-k} \|\varphi_k\|_*.$$

Сформулируем основные критерии устойчивости двухслойных операторно-разностных схем по начальным данным. Основным является следующий результат о точных (совпадающих необходимых и достаточных) условиях устойчивости в H_A.

Пусть в уравнении (15.13) оператор A является самосопряженным положительным оператором. Условие

$$B \ge \frac{\tau}{2} A, \quad t \in \omega_\tau \tag{15.20}$$

необходимо и достаточно для устойчивости в H_A, т.е. для выполнения оценки

$$\|y_{n+1}\|_A \le \|u_0\|_A, \quad t \in \omega_\tau. \tag{15.21}$$

При рассмотрении общих нестационарных задач необходимо ориентироваться на условия ϱ-устойчивости.

Пусть

$$A = A^*, \quad B = B^* > 0,$$

тогда условия

$$\frac{1-\varrho}{\tau} B \le A \le \frac{1+\varrho}{\tau} B \tag{15.22}$$

необходимы и достаточны для ϱ-устойчивости в H_B схемы (15.13), (15.14), т.е. для выполнения

$$\|y_{n+1}\|_B \le \varrho \|y_n\|_B.$$

Из устойчивости разностной схемы по начальным данным в H_R, $R = R^* > 0$ следует и устойчивость схемы по правой части. Более точно это утверждение формулируется следующим образом.

Пусть разностная схема (15.10), (15.11) ϱ–устойчива в H_R по начальным данным, т.е. имеет место оценка (15.19) при $\varphi^n = 0$. Тогда разностная схема (15.10), (15.11) устойчива по правой части и для решения справедлива априорная оценка

$$\|y^{n+1}\|_R \le \varrho^{n+1}\|u_0\|_R + \sum_{k=0}^{n} \tau \varrho^{n-k} \|B^{-1}\varphi^k\|_R. \tag{15.23}$$

Приведем оценку устойчивости по начальным данным и правой части при загрублении критерия устойчивости (15.20).

Пусть A — самосопряженный и положительный оператор, а B удовлетворяет условию

$$B \ge \frac{1+\varepsilon}{2}\tau A \tag{15.24}$$

с некоторой постоянной $\varepsilon > 0$, не зависящей от τ. Тогда для разностной схемы (15.10), (15.11) справедлива априорная оценка

$$\|y^{n+1}\|_A^2 \le \|u_0\|_A^2 + \frac{1+\varepsilon}{2\varepsilon} \sum_{k=0}^{n} \tau \|\varphi^k\|_{B^{-1}}^2. \tag{15.25}$$

Оценки устойчивости по правой части используются при исследовании точности разностных схем для нестационарных задач.

Устойчивость трехслойных разностных схем

При приближенном решении нестационарных задач математической физики наряду с двухслойными разностными схемами часто используют и трехслойные. Здесь мы формулируем некоторые основные условия устойчивости трехслойных операторно-разностных схем.

Используется следующая каноническая форма трехслойных разностных схем:

$$B\frac{y^{n+1} - y^{n-1}}{2\tau} + R(y^{n+1} - 2y^n + y^{n-1}) + Ay^n = \varphi^n, \quad n = 1, 2, \ldots \tag{15.26}$$

при заданных

$$y^0 = u_0, \quad y^1 = u_1. \tag{15.27}$$

Сформулируем условия устойчивости по начальным данным при постоянных, не зависящих от n, самосопряженных операторах A, B, R, т.е. вместо (15.26)

будем рассматривать

$$B\frac{y^{n+1}-y^{n-1}}{2\tau}+R(y^{n+1}-2y^n+y^{n-1})+Ay^n=0, \quad n=1,2,\ldots \tag{15.28}$$

При выполнении условий

$$B\geq 0, \quad A>0, \quad R>\frac{1}{4}A \tag{15.29}$$

для разностной схемы (15.27), (15.29) имеет место априорная оценка

$$\frac{1}{4}\|y_{n+1}+y_n\|_A^2+\|y_{n+1}-y_n\|_R^2-\frac{1}{4}\|y_{n+1}-y_n\|_A^2\leq$$

$$\leq\frac{1}{4}\|y_n+y_{n-1}\|_A^2+\|y_n-y_{n-1}\|_R^2-\frac{1}{4}\|y_n-y_{n-1}\|_A^2, \tag{15.30}$$

т.е. операторно-разностная схема (15.27), (15.29) устойчива по начальным данным.

Устойчивость рассматриваемых трехслойных операторно-разностных схем установлена в гильбертовых пространствах со сложной составной нормой (см. (15.30)). Можно получить оценки устойчивости в более простых за счет несколько более жестких условий устойчивости.

Пусть в операторно-разностной схеме (15.28) операторы R и A являются самосопряженными. Тогда при выполнении условий

$$B\geq 0, \quad A>0, \quad R>\frac{1+\varepsilon}{4}A \tag{15.31}$$

с $\varepsilon>0$ имеют место априорные оценки

$$\|y_{n+1}\|_A^2\leq 2\frac{1+\varepsilon}{\varepsilon}(\|y_0\|_A^2+\|y_1-y_0\|_R^2), \tag{15.32}$$

$$\|y_{n+1}\|_A^2+\|y_n-y_{n-1}\|_R^2\leq\frac{4+3\varepsilon}{\varepsilon}(\|y_0\|_A^2+\|y_1-y_0\|_R^2). \tag{15.33}$$

Для разностной схемы (15.26) при тех же предположениях об операторах R и A при выполнении операторных неравенств

$$B\geq\varepsilon E, \quad A>0, \quad R>\frac{1}{4}A \tag{15.34}$$

с постоянной $\varepsilon>0$ для разностного решения справедливы априорные оценки

$$\mathcal{E}_{n+1}\leq\mathcal{E}_1+\frac{1}{2\varepsilon}\sum_{k=1}^{n}\tau\|\varphi_k\|^2, \tag{15.35}$$

$$\mathcal{E}_{n+1}\leq\mathcal{E}_1+\frac{1}{2}\sum_{k=1}^{n}\tau\|\varphi_k\|_{B^{-1}}^2. \tag{15.36}$$

Здесь

$$\mathcal{E}_{n+1} = \frac{1}{4}(A(y_{n+1}+y_n), y_{n+1}+y_n) + (R(y_{n+1}-y_n), y_{n+1}-y_n) -$$

$$-\frac{1}{4}(A(y_{n+1}-y_n), y_{n+1}-y_n).$$

При сформулированных ограничениях величина $\mathcal{E}_n$ задает норму.

Разностные схемы для параболического уравнения

Рассмотрим разностные схемы для одномерного параболического уравнения (15.1). По пространству будем использовать равномерную сетку

$$\bar{\omega}_h = \{x \mid x = x_i = ih, \quad i = 0, 1, \ldots, N, \quad Nh = l\},$$

и пусть ω_h — множество внутренних узлов ($i = 1, 2, \ldots, N-1$), а $\partial\omega_h$ — множество граничных узлов.

При приближенном решении задачи (15.1)–(15.3) определим сеточный оператор

$$Ay = -(ay_{\bar{x}})_x, \quad x \in \omega_h, \tag{15.37}$$

для сеточных функций $y = 0$, $x \notin \omega_h$. Для задания коэффициента можно использовать выражения

$$a_i = k(x_{i-1/2}), \quad x_{i-1/2} = x_i - \frac{h}{2},$$

$$a_i = 0,5(k(x_{i-1}) + k(x_i)),$$

$$a_i = \left[\frac{1}{h}\int_{x_{i-1}}^{x_i} \frac{dx}{k(x)}\right]^{-1}$$

В $H = L_2(\omega_h)$ скалярное произведение и норму введем соотношениями

$$(y, v) = \sum_{i=1}^{N-1} y_i v_i h, \quad \|y\| = (y, y)^{1/2}.$$

Оператор A является самосопряженным и положительным:

$$A^* = A > 0. \tag{15.38}$$

Приведем также оценки оператора A снизу и сверху:

$$\delta E \le A \le \Delta E, \tag{15.39}$$

где

$$\delta = \frac{8}{l^2}\min_{0\le x\le l} k(x), \quad \Delta = \frac{4}{h^2}\max_{0\le x\le l} k(x).$$

Исходной дифференциальной задаче (15.1)–(15.3) поставим в соответствие задачу Коши для дифференциально-разностного уравнения:

$$\frac{dv}{dt} + Av = \varphi(t),$$

$$v(0) = u_0.$$

Для ее решения используем схему с весами

$$\frac{y^{n+1} - y^n}{\tau} + A(\sigma y^{n+1} + (1-\sigma)y^n) = \varphi^n, \quad n = 0, 1, \ldots, \tag{15.40}$$

$$y^0 = u_0. \tag{15.41}$$

Схема с весами будет устойчивой в H_A при

$$\sigma \geq \sigma_0, \quad \sigma_0 = \frac{1}{2} - \frac{1}{\tau \|A\|}. \tag{15.42}$$

В частности, схема с $\sigma \geq 0,5$ абсолютно (при всех $\tau > 0$) устойчива.

Рассмотрим вопрос о точности разностной схемы с весами (15.40), (15.41). Сформулируем соответствующую задачу для погрешности приближенного решения

$$z^n(x) = y^n(x) - u(x, t_n), \quad x \in \omega_h, \quad t_n \in \omega_\tau$$

с учетом

$$z^n(x) = 0, \quad x \in \partial\omega_h, \quad t_n \in \omega_\tau.$$

Начальное условие задается точно и поэтому положим

$$z_0(x) = 0, \quad x \in \omega_h.$$

Для погрешности из (15.41) следует

$$\frac{z^{n+1} - z^n}{\tau} + A(\sigma z^{n+1} + (1-\sigma)z^n) = \psi^n, \quad n = 0, 1, \ldots$$

Предполагая достаточную гладкость точного решения и коэффициентов уравнения (15.1) для погрешности аппроксимации будем иметь

$$\psi^n(x) = O(|h|^2 + \tau^\nu), \quad x \in \omega, \quad t_n \in \omega_\tau,$$

где $\nu = \nu(\sigma) = 2$ при $\sigma = 0,5$ и $\nu = 1$, если $\sigma \neq 0,5$.

Для погрешности верна априорная оценка

$$\|z^{n+1}\|_A \leq \|\psi^n\|_{A^{-1}} + \sum_{k=1}^{n} \tau \|\psi_t^k\|_{A^{-1}},$$

где использованы обозначения

$$y_t = \frac{y^n - y^{n-1}}{\tau}.$$

Следовательно разностная схема с весами сходится в H_A со скоростью $O(h^2+\tau^\nu)$.

На основе использования оценок устойчивости по правой части устанавливается сходимость и в других нормах.

Гиперболические уравнения

Рассмотрим теперь разностные схемы для решения краевой задачи для одномерного гиперболического уравнения второго порядка (15.2), (15.5), (15.6). После дискретизации по пространству придем к дифференциально-разностной задаче

$$\frac{d^2v}{dt^2}+Av=\varphi(t),\quad v(0)=u_0,\quad \frac{dv}{dt}(0)=u_1$$

с ранее рассмотренным разностным оператором A.

Будем использовать разностную уравнение

$$\frac{y^{n+1}-2y^n+y^{n-1}}{\tau^2}+A(\sigma y^{n+1}+(1-2\sigma)y^n+\sigma y^{n-1})=\varphi^n,\quad n=1,2,\dots, \tag{15.43}$$

которое аппроксимирует (15.5) со вторым порядком по времени и по пространству. Схема (15.43) записывается в каноническом виде (15.28) при

$$B=0,\quad R=\frac{1}{\tau^2}E+\sigma A.$$

Условия устойчивости (15.29) дают следующие ограничения на вес:

$$\sigma\geq\sigma_0,\quad \sigma_0=\frac{1}{4}-\frac{1}{\tau^2\|A\|}.$$

С привлечением априорных оценок устойчивости по правой части исследуется задача для погрешности и устанавливается сходимость разностной схемы (15.43).

Многомерные задачи

Будем рассматривать краевую задачу для параболического уравнения второго порядка (15.7)–(15.9) в прямоугольнике Ω. Введем равномерную прямоугольную сетку с шагами h_1 и h_2, так что

$$\omega_h=\{\mathbf{x}\mid \mathbf{x}=(x_1,x_2),\ x_\alpha=i_\alpha h_\alpha,\ i_\alpha=1,2,\dots,N_\alpha,\ N_\alpha h_\alpha=l_\alpha,\ \alpha=1,2\}.$$

Определим разностный оператор

$$A=\sum_{\alpha=1}^{2}A^{(\alpha)}, \tag{15.44}$$

где $A^{(\alpha)}$, $\alpha = 1, 2$ — одномерные разностные операторы

$$A^{(\alpha)} y = -(a^{(\alpha)} y_{\bar{x}_\alpha})_{x_\alpha}, \quad \alpha = 1, 2, \quad \mathbf{x} \in \omega_h, \tag{15.45}$$

определенные для сеточных функций $y(\mathbf{x}) = 0$, $\mathbf{x} \notin \omega_h$. Для коэффициентов положим, например,

$$a^{(1)}(\mathbf{x}) = k(x_1 - 0.5h_1, x_2), \quad a^{(2)}(\mathbf{x}) = k(x_1, x_2 - 0.5h_2).$$

Вычислительная реализация неявных схем ($\sigma \neq 0$) (15.40), (15.41) для численного решения задачи (15.7)–(15.9) связана с решением сеточной эллиптической задачи. В экономичных разностных схемах переход на новый временной слой осуществляется с вычислительными затратами на один узел, не зависящими от общего числа узлов дискретизации по пространству. Экономичные схемы строятся на основе аддитивного представления (15.44), с переходом к последовательности более простых задач с операторами $A^{(\alpha)}$, $\alpha = 1, 2$. Приведем примеры некоторых схем расщепления.

Для правой части уравнения используется аддитивное представление

$$\varphi^n = \varphi_1^n + \varphi_2^n.$$

Классическая разностная схема переменных направлений (схема Писмена-Рэкфорда) при расщеплении (15.44), (15.45) состоит из двух шагов. Сначала по известному y^n находиться вспомогательная сеточная функция, которую мы обозначим $y^{n+1/2}$, из уравнения

$$\frac{y^{n+1/2} - y^n}{0.5\tau} + A^{(1)} y^{n+1/2} + A^{(2)} y^n = 2\varphi_1^n. \tag{15.46}$$

Интерпретируя $y^{n+1/2}$ как решение на момент времени $t = t_{n+1/2}$, можем заметить, что (15.46) при $2\varphi_1^n = \varphi_n$ соответствует определению решения по чисто неявной схеме по переменной x_1 (оператор $A^{(1)}$) и по явной схеме по переменной x_2 (оператор $A^{(2)}$).

$$\frac{y^{n+1} - y^{n+1/2}}{0.5\tau} + A^{(1)} y^{n+1/2} + A^{(2)} y^{n+1} = 2\varphi_2^n. \tag{15.47}$$

Тем самым второй шаг связывается с использованием явной схемы по первой переменной и чисто неявной — по второй переменной.

Сформулируем условия устойчивости схемы переменных направлений. Пусть в схеме (15.46), (15.47) постоянные операторы $A^{(\alpha)} \geq 0$, $\alpha = 1, 2$. Тогда для разностного решения имеет место следующая оценка устойчивости по начальным данным и правой части:

$$\left\| \left(E + \frac{\tau}{2} A^{(2)} \right) y^{n+1} \right\| \leq \left\| \left(E + \frac{\tau}{2} A^{(2)} \right) y^0 \right\| + \sum_{k=0}^{n} \tau (\|\varphi_1^k\| + \|\varphi_2^k\|).$$

На основе этой оценки устанавливается, что схема переменных направлений сходится со скоростью $O(\tau^2 + |h|^2)$ в соответствующей, зависящей от операторов расщепления норме.

Необходимо выделить аддитивные схемы, которые относятся к классу безусловно устойчивых разностных схем при расщеплении на произвольное число операторов — схемы многокомпонентного расщепления. Аддитивные разностные схемы для задач с расщеплением на три и более попарно некоммутируемых операторов традиционно строятся на основе понятия суммарной аппроксимации – схемы покомпонентного расщепления (локально-одномерные схемы).

Для двумерной задачи (15.7)–(15.9) имеем

$$\frac{y^{n+\alpha/2}-y^{n+(\alpha-1)/2}}{\tau}+A^{(\alpha)}(\sigma_\alpha y^{n+\alpha/2}+(1-\sigma_\alpha)y^{n+(\alpha-1)/2})= \\ =\varphi_\alpha^n, \quad \alpha=1,2, \quad n=0,1,\ldots \tag{15.48}$$

При $\sigma_\alpha \geq 0.5$ схема покомпонентного расщепления (15.48) безусловно устойчива. Приведем соответствующую априорную оценку устойчивости по начальным данным и правой части.

Для правых частей φ_α^n, $\alpha=1,2$ будем использовать специальное представление

$$\varphi_{(\alpha)}^n = \overset{\circ}{\varphi}{}_\alpha^n + \overset{*}{\varphi}{}_\alpha^n, \quad \alpha=1,2, \quad \sum_{\alpha=1}^{2}\overset{\circ}{\varphi}{}_\alpha^n = 0. \tag{15.49}$$

При $0.5 \leq \sigma_\alpha \leq 2$, $\alpha=1,2$ и $\tau>0$ для решения задачи (15.48), (15.49) выполняется априорная оценка

$$\|y^{n+1}\| \leq \|y^0\| + \sum_{k=0}^{n}\tau\sum_{\alpha=1}^{2}\left(\|\overset{*}{\varphi}{}_\alpha^k\| + \tau\|A^{(\alpha)}\sum_{\beta=\alpha}^{2}\overset{\circ}{\varphi}{}_\alpha^k\|\right).$$

При исследовании сходимости локально-одномерных схем существенно учитывается специальное представление для погрешности погрешности типа (15.49). Отметим также, что устойчивость локально-одномерных схем устанавливается не только в гильбертовых пространствах сеточных функций, но и при использовании принципа максимума — в равномерной норме.

15.3 Упражнения

Упражнение 15.1 *Напишите программу численного решения на равномерной по пространству и времени сетке задачи*

$$\frac{\partial u}{\partial t}=\frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right)+f(x,t), \quad 0<x<l, \quad 0<t\leq T,$$

$$u(0,t)=0, \quad u(l,t)=0, \quad 0<t\leq T,$$

$$u(x,0)=v(x), \quad 0\leq x\leq l$$

при использовании двухслойной схемы с весом σ (см. (15.40), (15.41)). С помощью этой программы проведите численные эксперименты по приближенному решению задачи с $k(x) = 1$, $f(x,t) = 0$ и точным решением ($l = 1$)

$$u(x,t) = e^{-\pi^2 t}\sin(\pi x) + e^{-16\pi^2 t}\sin(4\pi x)$$

на различных сетках по времени при использовании симметричной ($\sigma = 0.5$) и чисто неявной ($\sigma = 1$) разностных схем.

В модуле parabolic1D функция `parabolic1D()` реализует численное решение краевой задачи для одномерного параболического уравнения с однородными граничными условиями первого рода с использованием двухслойной разностной схемы с весом. Для решение систем уравнений с трехдиагональной матрицей привлекается функция `solveLU3()` модуля lu3.

Модуль parabolic1D

```
import numpy as np
from lu3 import solveLU3
def parabolic1D(k, f, v, l, tEnd, n, tau, sigma):
    """
    Numerical Solution of the Dirichlet problem
    for one-dimensional parabolic equation.
    Use two-layer scheme with the weight of sigma.
    """
    h = l / n
    a = np.ones((n+1), 'float')
    b = np.zeros((n+1), 'float')
    c = np.zeros((n+1), 'float')
    d = np.zeros((n+1), 'float')
    q = np.zeros((n+1), 'float')
    t0 = 0.
    y0 = np.zeros((n+1), 'float')
    for i in range(1,n):
        x = i*h
        y0[i] = v(x)
        d[i] = k(x-h/2.)
    d[n] = k(l-h/2.)
    t = []
    y = []
    t.append(t0)
    y.append(y0)
    while t0 < tEnd - tau:
        for i in range(1,n):
            a[i] = 1. / tau + sigma*(d[i+1] + d[i])/h**2
            b[i] = - sigma*d[i+1]/h**2
            c[i] = - sigma*d[i]/h**2
```

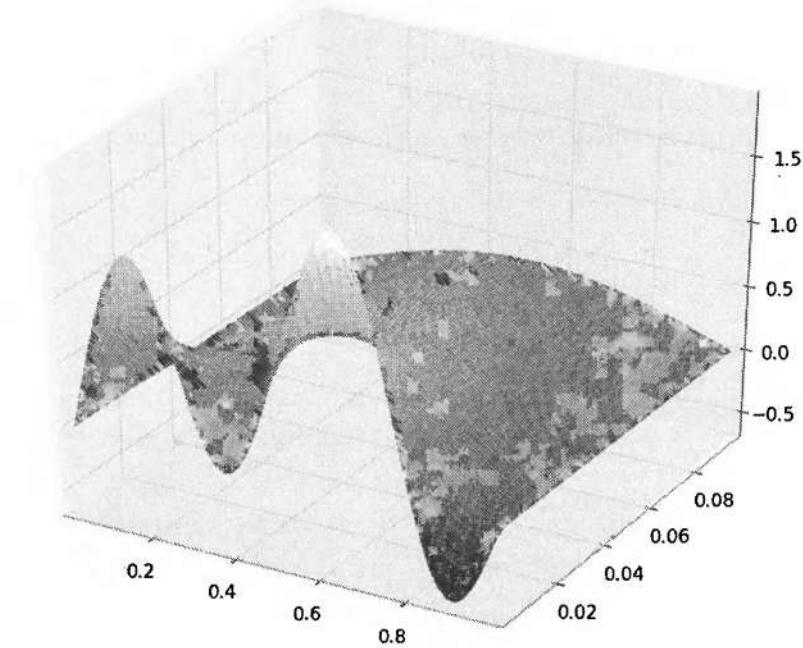

Рис. 15.1 Решение параболической задачи

```
            q[i] = f(i*h, t0+tau/2.) + y0[i]/tau \
                 + (1.-sigma)*(d[i+1]*(y0[i+1]-y0[i]) \
                 - d[i]*(y0[i]-y0[i-1]))/h**2
        y0 = solveLU3(a, b, c, q)
        t0 = t0 + tau
        t.append(t0)
        y.append(y0)
    return np.array(t), np.array(y)
```

Для приближенного решение модельной задачи при разных шагах по времени используется следующая программа.

exer −15 1 py

```
import numpy as np
import math as mt
from parabolic1D import parabolic1D
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
def k(x):
```

```
    return 1.
def f(x,t):
    return 0. #mt.exp(x)
def v(x):
    return mt.sin(mt.pi*x) + mt.sin(4.*mt.pi*x)
l = 1.
n = 50
h = l / n
tEnd = 0.1
u = np.zeros((n+1), 'float')
tauList = [0.01, 0.005, 0.0025]
for tau in tauList:
    print 'tau =', tau
    for sigma in [0.5, 1.0]:
        t, y = parabolic1D(k, f, v, l, tEnd, n, tau, sigma)
        erMax = 0.
        for m in range(len(t)):
            yk = y[m,:]
            for i in range(1,n):
                x = i*h
                pt = mt.pi**2*t[m]
                u[i] = mt.sin(mt.pi*x)*mt.exp(-pt) \
                     + mt.sin(4.*mt.pi*x)*mt.exp(-16.*pt)
            er = mt.sqrt(np.dot(yk-u,yk-u)*h)
            if er > erMax:
                erMax = er
        print '    sigma =', sigma, 'er =', erMax
fig = plt.figure()
ax = Axes3D(fig)
xx = np.linspace(0., l, n+1)
t, y = parabolic1D(k, f, v, l, tEnd, n, 0.001, 0.5)
X, T = np.meshgrid(xx, t)
ax.plot_surface(X, T, y, rstride=1, cstride=1, cmap='gray')
plt.show()
```

```
tau = 0 01
    sigma = 0 5 er = 0 0607892675115
    sigma = 1 0 er = 0 129310302279
tau = 0 005
    sigma = 0 5 er = 0 0127223352908
    sigma = 1 0 er = 0 07606480772
tau = 0 0025
    sigma = 0.5 er = 0 00199840149212
    sigma = 1 0 er = 0 0454283258452
```

Приведенные данные иллюстрируют теоретические оценки скорости сходимости для схемы с весами при $\sigma = 0.5$ и $\sigma = 1$. Само численное решение показано на рис. 15.1.

Упражнение 15.2 *Напишите программу численного решения на равномерной по пространству и времени сетке двумерной задачи*

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} + f(\mathbf{x}, t), \quad \mathbf{x} \in \Omega, \quad 0 < t \leq T,$$

$$u(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \partial\Omega \quad 0 < t \leq T,$$

$$u(\mathbf{x}, 0) = v(\mathbf{x}), \quad \mathbf{x} \in \Omega$$

в прямоугольнике Ω при использовании схемы переменных направлений Писмена–Рэкфорда (см. (15.46), (15.47)). Работоспособность программы продемонстрируйте на численном решении задачи с точным решением

$$u(\mathbf{x}, t) = tx(l_1 - x)y(l_2 - y)$$

на различных сетках по времени.

В модуле parabolic2D функция `parabolic2D()` предназначена для решения краевой задачи для двумерного параболического уравнения с однородными граничными условиями первого рода при использовании схемы Писмена–Рэкфорда. Решение систем уравнений с трехдиагональной матрицей проводится функцией `solveLU3()` модуля lu3.

Модуль parabolic2D

```
import numpy as np
from lu3 import solveLU3
def parabolic2D(f, v, l1, l2, tEnd, n1, n2, tau):
    """
    Numerical Solution of the Dirichlet problem
    for two-dimensional parabolic equation.
    Use additive difference scheme of alternating directions.
    """
    h1 = l1 / n1
    h2 = l2 / n2
    a1 = np.ones((n1+1), 'float')
    b1 = np.zeros((n1+1), 'float')
    c1 = np.zeros((n1+1), 'float')
    q1 = np.zeros((n1+1), 'float')
    a2 = np.ones((n2+1), 'float')
    b2 = np.zeros((n2+1), 'float')
    c2 = np.zeros((n2+1), 'float')
    q2 = np.zeros((n2+1), 'float')
    t0 = 0.
```

```
    y0 = np.zeros((n1+1,n2+1), 'float')
    for i in range(1,n1):
        for j in range(1,n2):
            y0[i,j] = v(i*h1,j*h2)
    y = np.copy(y0)
    while t0 < tEnd - 0.001*tau:
        tau = min(tau, tEnd - t0)
        # x1 direction
        for j in range(1,n2):
            for i in range(1,n1):
                a1[i] = 2. / tau + 2./h1**2
                b1[i] = - 1./h1**2
                c1[i] = - 1./h1**2
                q1[i] = f(i*h1, j*h2, t0+tau/2.) + 2.*y[i,j]/tau \
                      + (y[i,j+1]-2.*y[i,j]+y[i,j-1])/h2**2
            y0[:,j] = solveLU3(a1, b1, c1, q1)
        # x2 direction
        for i in range(1,n1):
            for j in range(1,n2):
                a2[j] = 2. / tau + 2./h2**2
                b2[j] = - 1./h2**2
                c2[j] = - 1./h2**2
                q2[j] = f(i*h1, j*h2, t0+tau/2.) + 2.*y0[i,j]/tau \
                      + (y0[i+1,j]-2.*y0[i,j]+y0[i-1,j])/h1**2
            y[i,:] = solveLU3(a2, b2, c2, q2)
        t0 = t0 + tau
    return t0, y
```

Численное решение модельной задачи на трех различных сетках по времени дается следующей программой.

exer-15.2.py

```
import numpy as np
from parabolic2D import parabolic2D
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
l1 = 1.
l2 = 1.
def f(x,y,t):
    return x*(l1-x)*y*(l2-y)+2.*t*(x*(l1-x)+y*(l2-y))
def v(x,y):
    return 0.
def u(x,y,t):
    return t*x*(l1-x)*y*(l2-y)
n1 = 50
```

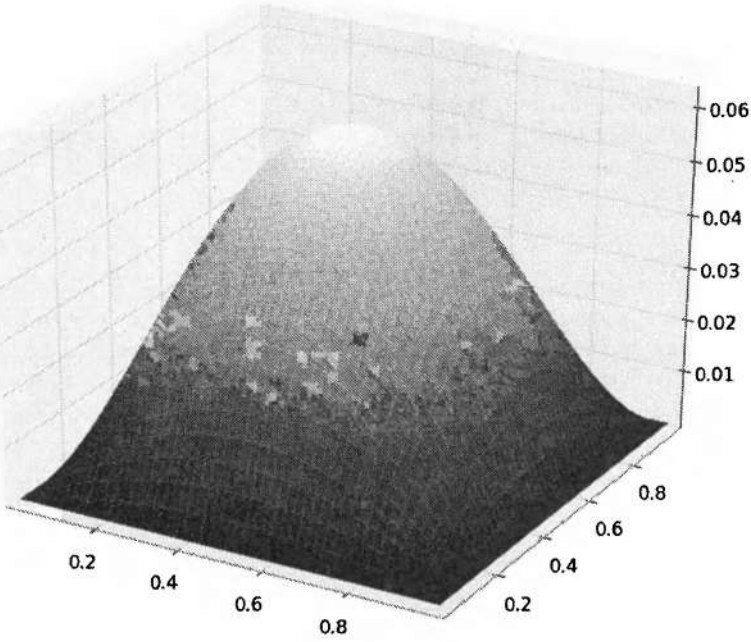

Рис. 15.2 Решение двумерной задачи

```
n2 = 50
tEnd = 1.
x = np.linspace(0., l1, n1+1)
y = np.linspace(0., l2, n2+1)
ut = np.zeros((n1+1,n2+1), 'float')
for i in range(1,n1):
    for j in range(1,n2):
        ut[i,j] = u(x[i],y[j],tEnd)
tauList = [0.1, 0.05, 0.025]
for tau in tauList:
    t, U = parabolic2D(f, v, l1, l2, tEnd, n1, n2, tau)
    print 'tau =', tau, 'error =', abs(np.amax(ut-U))
fig = plt.figure()
ax = Axes3D(fig)
X, Y = np.meshgrid(x, y)
ax.plot_surface(X, Y, U, rstride=1, cstride=1, cmap='gray')
plt.show()
```

```
tau = 0.1 error = 0.000736477377029
```

```
tau = 0.05  error = 0.000184120363324
tau = 0.025  error = 4.60300908713e-05
```

Приближенное решение на конечный момент времени ($T = 1$) приведено на рис. 15.2. Сходимость при уменьшении шага по времени иллюстрируется расчетными данными для трех значений шага по времени τ.

15.4 Задачи

Задача 15.1 *Напишите программу численного решения на равномерной по каждому направлению сетке задачи для уравнений переноса*

$$\frac{\partial u}{\partial t} + b(x,t)\frac{\partial u}{\partial x} = 0, \quad 0 < x < l, \quad 0 < t \leq T$$

при начальном условии

$$u(x,0) = v(x), \quad 0 \leq x \leq l$$

и периодических условиях по пространству

$$u(x+l,t) = u(x,t) = 0, \quad 0 < t \leq T.$$

Используйте явную двухслойную схему при аппроксимации конвективного слагаемого центральными и направленными разностями. С помощью этой программы найдите численное решение на разных сетках задачи при $l = 1$, $b(x,t) = 1$ и начальном условии

$$v(x) = \begin{cases} 1, & 0.1 < x < 0.3, \\ 0, & x \leq 0.1, \ x \geq 0.3. \end{cases}$$

Задача 15.2 *Напишите программу для численного решения краевой задачи для одномерного параболического уравнения с краевыми условиями третьего рода:*

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + f(x,t), \quad 0 < x < l, \quad 0 < t \leq T,$$

$$-\frac{du}{dx}(0,t) + \sigma_1(t)u(0,t) = \mu_1(t), \quad \frac{du}{dx}(l,t) + \sigma_2(t)u(l,t) = \mu_2(t), \quad 0 < t \leq T,$$

$$u(x,0) = v(x), \quad 0 \leq x \leq l$$

на равномерной по пространству и времени сетке при использовании чисто неявной разностной схемы. Продемонстрируйте возможности программы при приближенном решении задачи ($l = 1$, $\sigma_1 = \sigma_2 = 0, 10, 100$) с точным решением

$$u(x,t) = \sin(t)(1 + 2x - 3x^2).$$

Задача 15.3 *Напишите программу для численного решения одномерного уравнения конвекции-диффузии*

$$\frac{\partial u}{\partial t} + b(x,t)\frac{\partial u}{\partial x} = \frac{\partial^2 u}{\partial x^2} + f(x,t), \quad 0 < x < l, \quad 0 < t \le T,$$

которое дополнено граничными и начальными условиями

$$u(0,t) = 0, \quad u(l,t) = 0, \quad 0 < t \le T,$$

$$u(x,0) = v(x), \quad 0 \le x \le l.$$

Используйте двухслойную явно-неявную схему, когда на равномерной по пространству и времени сетке диффузионное слагаемое берется с верхнего слоя по времени, а конвективное слагаемое – с нижнего при центрально-разностных аппроксимациях конвективного слагаемого. Используйте эту программу для приближенного решения задачи при $l = 1$, $f(x,t) = 0$ *и* $b(x) = \nu = 0, 10, 100$.

Упражнение 15.3 *Напишите программу численного решения на равномерной по пространству и времени сетке задачи*

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right) + f(x,t), \quad 0 < x < l, \quad 0 < t \le T,$$

$$u(0,t) = 0, \quad u(l,t) = 0, \quad 0 < t \le T,$$

$$u(x,0) = v(x), \quad \frac{\partial u}{\partial t}(x,0) = w(x), \quad 0 \le x \le l$$

при использовании трехслойной схемы с весом σ *(см. (15.43)). Примените эту программу для приближенного решению задачи с* $k(x) = 0$, $f(x,t) = 0$ *и точным решением (*$l = 1$*)*

$$u(x,t) = \cos(2\pi t)\sin(2\pi x)$$

на различных сетках по времени при $\sigma = 0, 0.25$.

Список литературы

[1] Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. *Численные методы.* — М.: Бином, 2008.

[2] Бахвалов Н. С., Лапин А. В., Чижонков Е. В. *Численные методы в задачах и упражнениях.* — М.: Высшая школа, 2000.

[3] Березин И. С., Жидков Н. П. *Методы вычислений.* — М.: Наука, 1966, т.1; Физматгиз, 1962, т.2.

[4] Биэли Д.М. *Язык программирования Python. Справочник.* — Киев.: ДиаСофт, 2000.

[5] Васильев Ф. П. *Численные методы решения экстремальных задач.* — М.: Наука, 1988.

[6] Волков Е. А. *Численные методы.* — М.: Лань, 2004.

[7] Гаврилов М. К. *Лекции по методам вычислений.* М.: Наука, 1971.

[8] Годунов С. К., Рябенький В. С. *Разностные схемы.* — М.: Наука, 1977.

[9] Дробышевич В. И., Дымников В. П., Ривин Г. С. *Задачи по вычислительной математике.* — М.: Наука, 1980.

[10] Завьялов Ю. С., Квасов Б. И., Мирошниченко В. Л. *Методы сплайн-функций.* — М.: Наука, 1980.

[11] Калиткин Н. Н. *Численные методы.* — М.: Наука, 1978.

[12] Кахапер Д., Моулер К., Нэш С. *Численные методы и программное обеспечение.* — М.: Мир, 2001.

[13] Коллатц Л., Альбрехт Ю. *Задачи по прикладной математике.* — М,: Мир, 1978.

[14] Коновалов А. Н. *Введение в вычислительные методы линейной алгебры.* — Новосибирск: Наука, 1993.

[15] Крылов В. И., Бобков В. В., Монастырный П. И. *Вычислительные методы.* — М.: Наука, 1976, т.1; 1977, т.2.

[16] Лейнингем И. *Освой самостоятельно Python за 24 часа.* — Киев: Вильямс, 2001.

[17] Лутц Марк. *Изучаем Python.* — СПб: Символ-Плюс, 2009.

[18] Лутц Марк. *Программирование на Python.* — СПб: Символ-Плюс, 2002.

[19] Ляшко И. И., Макаров В. Л., Скоробогатько А. А. *Методы вычислений.* — Киев: Высшая школа, 1977.

[20] Марчук Г. И. *Методы вычислительной математики.* — М.: Лань, 2009.

[21] Марчук Г. И. *Методы расщепления.* —М.: Наука, 1988.

[22] Марчук Г. И., Агошков В. И. *Введение в проекционно-сеточные методы.* — М.: Наука, 1981.

[23] Мэтьюз Джон Г., Финк Куртис Д. *Численные методы. Использование MATLAB.* — М.: Вильямс, 2001.

[24] Ортега Дж., Пул У. *Введение в численные методы решения дифференциальных уравнений.* — М.: Наука, 1986.

[25] Ортега Дж., Рейнболдт В. *Итерационные методы решения нелинейных систем уравнений со многими неизвестными.* — М.: Мир, 1975.

[26] Петров И. Б., Лобанов А. И. *Лекции по вычислительной математике.* — М.: Бином, 2009.
[27] Плохотников К. Э. *Вычислительные методы. Теория и практика в среде MATLAB.* — М.: Горячая Линия - Телеком, 2009.
[28] *Сборник от задачи по числени методи.* — София: Наука и изкуство, 1986.
[29] Самарский А. А. *Введение в численные методы.* — М.: Лань, 2009.
[30] Самарский А. А. *Теория разностных схем.* — М.: Наука, 1989.
[31] Самарский А. А., Андреев В. Б. *Разностные методы для эллиптических уравнений.* — М.: Наука, 1976.
[32] Самарский А. А., Вабищевич П.Н. *Аддитивные схемы для задач математической физики.* — М.: Наука, 1999.
[33] Самарский А. А., Вабищевич П.Н. *Численные методы решения задач конвекция-диффузии.* — М.: URSS, 2009.
[34] Самарский А. А., Вабищевич П.Н. *Численные методы решения обратных задач математической физики.* — М.: URSS, 2009.
[35] Самарский А. А., Вабищевич П.Н., Самарская Е. А. *Задачи и упражнения по численным методам.* — М.: URSS, 2009.
[36] Самарский А. А., Гулин А. В. *Устойчивость разностных схем.* — М.: URSS, 2009.
[37] Самарский А. А., Гулин А. В. *Численные методы.* — М.: Научный мир, 2000.
[38] Самарский А. А., Николаев Е. С. *Методы решения сеточных уравнений.* — М.: Наука, 1978.
[39] Саммерфилд Марк. *Программирование на Python 3. Подробное руководство.* — СПб.: Символ-Плюс, 2009.
[40] *Сборник задач по методам вычислений.* — М.: Физматлит, 1994.
[41] Соболь Б.В., Месхи Б.Ч., Пешхоев И. *Практикум по вычислительной математике.* — М.: Феникс, 2008.
[42] *Современные численные методы решения обыкновенных дифференциальных уравнений.* — М.: Мир, 1979.
[43] Стренг Г., Фикс Дж. *Теория метода конечных элементов.* — М.: Мир, 1980.
[44] Сузи Р. А. *Язык программирования Python.* — М.: Бином, 2007.
[45] Сузи Роман. *Python. Наиболее полное руководство.* — СПб.: БХВ-Петербург, 2002.
[46] Сухарев А. Г., Тимохов А. В., Федоров В. В. *Курс методов оптимизации.* — М.: Физматлит, 2005.
[47] Тихонов А. Н., Арсенин В. Я. *Методы решения некорректных задач.* — М.: Наука, 1986.
[48] Тыртышников Е. Е. *Методы численного анализа.* — М.: Академия, 2007.
[49] Фаддеев Д. К., Фаддеева В. П. *Вычислительные методы линейной алгебры.* — М.: Лань, 2009.
[50] Хайрер Э., Ваннер Г. *Решение обыкновенных дифференциальных уравнений: Жесткие и дифференциально-алгебраические задачи алгебры.* — М.: Мир, 1999.
[51] Хайрер Э., Нерсетт С., Ваннер Г. *Решение обыкновенных дифференциальных уравнений: Нежесткие задачи.* — М.: Мир, 1990.
[52] Яненко Н. Н. *Метод дробных шагов решения многомерных задач математической физики.* — Новосибирск: Наука, 1967.