Введение в Scilab

Michaël Baudin

Перевод Artem Glebov

March 2013

Аннотация

В настоящем документе рассматриваются основные возможности пакета Scilab и навыки, необходимые для того, чтобы приступить к работе как можно быстрее. В первой главе показано, как получить дистрибутив и установить Scilab на персональный компьютер, а также где искать помощи в случае затруднений. Вторая глава представляет обзор графической и текстовой среды пакета Scilab. Основные конструкции встроенного языка программирования Scilab рассматриваются в главе 3. Глава 4 посвящена работе с матрицами и основным функциям библиотеки линейной алгебры. В пятой главе представлены основные конструкции структурного программирования в Scilab. Вопросы определения пользовательских функций, управления входными и выходными аргументами, а также создания библиотек функций составляют шестую главу. Завершает обзор глава 7, где дано краткое введение в графические возможности Scilab, включая построение двухмерных графиков и экспорт изображений.

Содержание

1 Введение

Данный раздел представляет краткий обзор целей создания и основных особенностей пакета Scilab. Здесь мы рассмотрим способы получения и установки дистрибутивов Scilab, основные справочные источники, включая встроенную справочную систему пакета, а также интерактивные демонстрации, поставляемые в составе дистрибутивов.

1.1 О данном руководстве

Данный документ, как и программный продукт, который он описывает, является проектом с открытым исходным кодом. Исходный текст в разметке LaTeX доступен в репозитории Scilab Forge:

http://forge.scilab.org/index.php/p/docintrotoscilab/

Исходный текст LaTeX распространяется в соответствии с лицензией Creative Commons Attribution-ShareAlike 3.0 Unported License:

http://creativecommons.org/licenses/by-sa/3.0

Скрипты Scilab, используемые в данном руководстве, содержатся в папке `scripts` проекта. Правила распространения скриптов определяются лицензией CeCiLL:

http://www.cecill.info/licences/Licence_CeCILL_V2-en.txt

1.2 Обзор пакета Scilab

Программный пакет Scilab объединяет в себе развитый язык программирования и обширную библиотеку численных алгоритмов, охватывающую многие области научных и технических вычислений.

Язык программирования Scilab относится к числу интерпретируемых языков высокого уровня, предоставляя пользователю возможность напрямую манипулировать математическими конструкциями, такими как матрицы или полиномы. Тем самым достигается большая скорость и простота написания программ. Язык Scilab допускает расширение посредством определения пользовательских типов данных. При этом стандартным операциям, например, арифметическим операторам или операторам сравнения, возможно придать особый смысл применительно к пользовательским типам данных. Пользователи пакета могут разрабатывать собственные модули расширения для решения конкретных задач. Возможен также вызов из Scilab функций, реализованных на других языках программирования, в частности Fortran или C, благодаря чему сторонние библиотеки могут быть использованы, как если бы они были частью встроенных средств пакета. Scilab также предоставляет возможности для взаимодействия с программным комплексом LabVIEW компании National Instruments, предназначенным для визуального проектирования измерительных систем, а также сбора и анализа экспериментальных данных.

Разрабатываемый в соответствии с принципами свободного программного обеспечения, Scilab распространяется бесплатно на основе лицензии Cecill [2]. Дистрибутив Scilab включает исходный код, поэтому заинтересованный пользователь может самостоятельно исследовать внутреннее устройство пакета и особенности его работы. Скомпилированные версии пакета Scilab доступны для операционных систем Windows, Linux и Mac OS. Справочная документация переведена на многие языки мира.

Scilab предоставляет чрезвычайно богатый набор средств для научных и инженерных расчетов. Хотя первоначальный акцент при разработке пакета был сделан на матричную алгебру, вскоре функциональные возможности расширились настолько, что охватили большинство разделов научных вычислений, включая:

- линейную алгебру и разреженные матрицы,
- полиномы и рациональные функции,
- интерполяцию и аппроксимацию,
- линейную, квадратичную и нелинейную оптимизацию,
- обыкновенные дифференциальные уравнения, дифференциально-алгебраические уравнения,
- классическое и робастное управление, решение линейных матричных неравенств,
- оптимизацию дифференцируемых и недифференцируемых функций,
- обработку сигналов,
- математическую статистику.

Кроме того, Scilab содержит значительное число функций для построения графиков, а также мощное средство визуального моделирования Xcos, которое объединяет в себе возможности редактора моделей и симулятора.

1.3 Как получить дистрибутив и установить Scilab

Дистрибутивы, содержащие исполняемые файлы для каждой из поддерживаемых платформ (Windows, Linux и Mac OS), доступны на домашней странице Scilab

http://www.scilab.org

а также в разделе Download

http://www.scilab.org/download

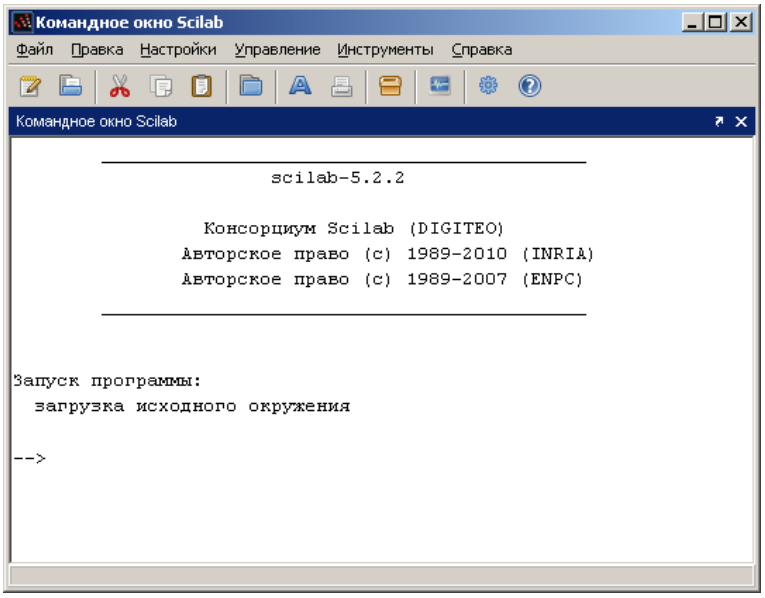

Рис. 1. Консоль Scilab в ОС Windows.

Предлагаются версии для 32- и 64-битовых платформ.

Scilab также может быть загружен в форме исходных кодов и скомпилирован пользователем самостоятельно. Самостоятельная компиляция будет интересна в том случае, если пользователь желает более подробно ознакомиться с особенностями внутреннего функционирования Scilab, а также при необходимости отладки или добавления новых возможностей. Для компиляции Scilab понадобятся дополнительные файлы, которые также можно загрузить в разделе Download. Кроме того, потребуется компилятор Fortran или C. Инструкции по компиляция можно найти в разделе "Compilation of Scilab" wiki-энциклопедии Scilab, размещающейся по адресу

http://wiki.scilab.org

1.3.1 Установка Scilab в Windows

Версия Scilab для ОС Windows содержит программу-установщик, которая значительно облегчает установку и настройку пакета. В ходе установки на рабочий стол добавляется ярлык приложения, при выборе которого отображается консоль Scilab (рис. 1).

Для платформ на базе процессоров Intel в Scilab используется библиотека Intel Math Kernel Library (MKL) [6], позволяющая существенно повысить скорость вычислений.

1.3.2 Установка Scilab в Linux

Для операционных систем семейства Linux бинарные версии Scilab предлагаются в виде сжатых tar-файлов (`*.tar.gz`). Специальной программы-установщика

в данном случае не требуется - достаточно распаковать архив в выбранный каталог, после чего Scilab готов к запуску. Исполняемый файл расположен по адресу `<path>/scilab-5.x.x/bin/scilab`, где `<path>` - путь к каталогу, куда был распакован архив. При выполнении данного скрипта отображается окно консоли, которая полностью аналогична консоли в версии для Windows.

Отметим, что Scilab также распространяется в виде пакетов в дистрибутивах Linux, основанных на Debian (например, Ubuntu). Установка в этом случае чрезвычайно проста, однако поскольку между выходом новой версии Scilab и обновлением соответствующих пакетов в дистрибутивах проходит некоторое время, загруженная таким образом версия пакета может оказаться не самой свежей.

В настоящий момент в версии Scilab для Linux используется библиотека линейной алгебры, которая обеспечивает независимость от конкретной операционной системы. При этом из Scilab для Linux исключена бинарная версия библиотеки ATLAS [1], поэтому функции линейной алгебры могут выполняться медленнее, чем в Windows.

1.3.3 Установка Scilab в Mac OS

Версия Scilab для Mac OS предлагается в виде файла формата `.dmg`. Данный формат поддерживается версиями Mac OS начиная с 10.5. Для распаковки используется классический инсталлятор Mac OS. Архитектуры на основе Power PC в Scilab не поддерживаются.

В версии Scilab 5.2 для Mac OS по техническим причинам отключена библиотека Tcl / Tk, в результате чего существуют определенные ограничения на использование Scilab в рамках данной платформы. В частности, не работает интеграция Scilab/Tcl (TclSci), графический редактор и редактор переменных. Эти возможности будут реализованы на языке Java в будущих версиях Scilab, после чего ограничения будут сняты.

Несмотря на указанные особенности, использовать Scilab в Mac OS достаточно просто, чему немало способствуют "горячие" клавиши, привычные пользователям данной платформы. Например, как в консоли, так и в редакторе можно пользоваться клавишей Cmd, имеющейся на клавиатуре Mac. А поскольку платформа не поддерживает щелчков правой клавишей мыши, в Scilab для Mac OS вместо нее применяется комбинация клавиши Control и щелчка мышью.

Как и в случае с Linux, версия Scilab для Mac OS содержит библиотеку функций линейной алгебры, однако не комплектуется бинарной версией библиотеки ATLAS [1], поэтому функции линейной алгебры могут выполняться несколько медленнее, чем в Windows.

1.4 Как получить помощь

Наиболее простым способом получить справку по возможностям пакета Scilab является функция `help`. Окно справки Scilab показано на рис. 2. Для его отображения наберите `help` в консоли и нажмите клавишу <Enter>:

```
-->help
```

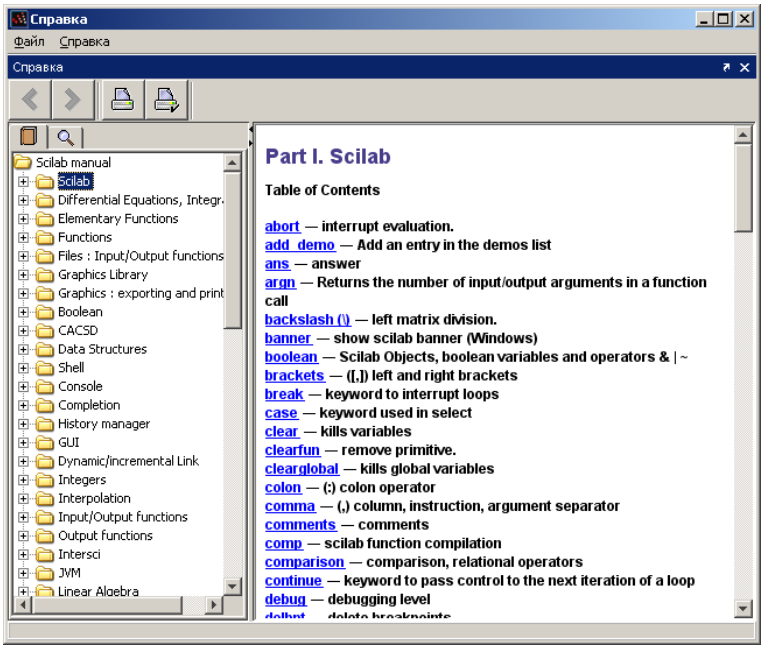

Рис. 2. Окно справки Scilab. Левая панель представляет список разделов справки, а правая - содержимое выбранного раздела.

Если вас интересует информация о конкретной функции (например, `optim`), вы можете пролистать содержание справки, найти раздел, посвященный оптимизации, и выбрать `optim`, после чего будет отображено окно со справкой по данной функции. Однако более удобным способом получить информацию о конкретной функции, если вы знаете ее название, является использование команды `help` с указанием имени интересующей функции:

```
-->help optim
```

В этом случае Scilab автоматически отобразит окно справки, содержащее информацию о выбранной функции. Если функции с указанным именем не существует, будет отображено сообщение об ошибке.

Также вы можете использовать страницы справки на сайте Scilab:

http://www.scilab.org/product/man

Эта страница всегда содержит сведения о наиболее свежей версии Scilab. Используя возможности поиска вашего Интернет-обозревателя, вы можете быстро найти необходимую информацию. Также можно одновременно держать открытыми окна со справкой по нескольким командам Scilab. Например, изучив описание команд `derivative` и `optim`, можно записать целевую функцию, основанную на вычислении производных посредством команды `derivative` и пригодную для оптимизации при помощи `optim`.

На домашней страничке Scilab также можно найти список книг, онлайн-руководств и статей, посвященных пакету:

http://www.scilab.org/publications

1.5 Списки рассылки, wiki и отчеты о дефектах

Список рассылки *users@lists.scilab.org* предназначен для общих вопросов, касающихся использования Scilab. Для того чтобы подписаться на него, отправьте письмо на *users-subscribe@lists.scilab.org* (содержание и тема письма не имеют значения, поэтому можно оставить то и другое пустым). Список рассылки *dev@lists.scilab.org* посвящен вопросам разработки Scilab, в том числе вычислительного ядра и модулей, активно взаимодействующих с ним. Для того чтобы подписаться, отправьте пустое письмо на *dev-subscribe@lists.scilab.org*.

Архивы данных рассылок доступны по адресам

http://dir.gmane.org/gmane.comp.mathematics.scilab.user

и

http://dir.gmane.org/gmane.comp.mathematics.scilab.devel

Перед тем как задавать вопрос в списке рассылки, настоятельно рекомендуется для начала обратиться к архиву - возможно, на этот вопрос уже был дан ответ.

Специфические вопросы, касающиеся технических моментов и не предназначенные для широкой аудитории, следует направлять по адресу электронной почты *scilab.support@scilab.org*, где они будут рассмотрены разработчиками Scilab и получат квалифицированные ответы.

Wiki-энциклопедия Scilab, представляющая открытый источник для обмена опытом между пользователями и разработчиками пакета, располагается по адресу

http://wiki.scilab.org

Программисты смогут здесь найти пошаговые инструкции для компиляции Scilab, информацию о библиотеках, необходимых для работы различных версий пакета, указания по работе с хранилищем исходного кода и т.д.

Для регистрации обнаруженных дефектов консорциум Scilab использует систему Bugzilla (*http://bugzilla.scilab.org*). Если вы обнаружили ошибку в функционировании пакета, разработчики Scilab будут вам признательны, если вы сообщите о ней, заполнив размещенную по этому адресу форму. Может случиться, что данную ошибку уже обнаружил кто-то другой, поэтому целесообразно провести поиск в базе данных перед тем, как сообщать о новой ошибке. Если ошибка пока не зарегистрирована, пожалуйста, сообщите о ней, сопроводив ваш отчет описанием ситуации, в которой она возникает, и последовательностью инструкций, позволяющей ее воспроизвести. Последовательность инструкция должна быть по возможности простой, что позволит быстро обнаружить и устранить дефект.

Эффективным способом получения актуальной информации о Scilab является RSS-подписка:

http://www.scilab.org/en/rss_en.xml

В данном канале регулярно публикуются пресс-релизы и объявления общего характера, которые могут заинтересовать пользователей Scilab.

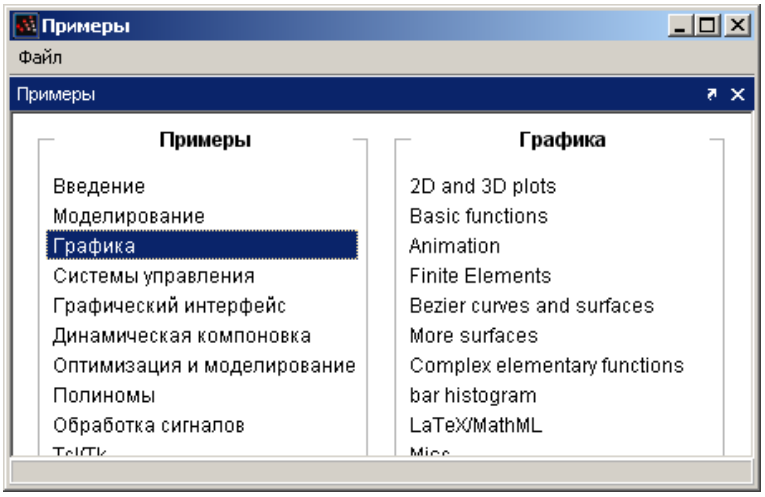

Рис. 3. Окно демонстраций Scilab.

1.6 Демонстрационные примеры Scilab

Дистрибутив Scilab включает набор демонстрационных примеров, доступных для запуска из консоли либо при выборе пункта меню *Справка* > *Примеры* (*?* > *Scilab Demonstrations*)[1]. На рис. 3 показано окно демонстраций. Некоторые демонстрации являются графическими, другие предполагают пошаговое выполнение, ожидая нажатия клавиши <Enter> для перехода к следующему шагу.

Сопутствующие демонстрационные скрипты размещаются в соответствующих подкаталогах каталога Scilab. Например, демонстрация, связанная с модулем *optimization*, находится в файле

```
<path>\scilab-<version>\modules\optimization\demos\datafit\datafit.dem.sce
```

где `<path>` - это путь к каталогу файловой системы, куда установлен Scilab (например, `C:/Program Files` в Windows), а `<version>` - версия пакета (например, 5.2.2).

Разбор содержимого этих демонстрационных файлов может помочь начинающему пользователю избежать многих типичных ошибок, а также способствовать пониманию внутренних особенностей работы пакета.

Полезную информацию может дать также анализ исходного кода встроенных функций Scilab. Например, исходный текст функции `derivative` размещается в файле

```
<path>\scilab-<version>\modules\optimization\macros\derivative.sci
```

В большинстве своем функции реализованы на высоком профессиональном уровне, с учетом всех возможных комбинаций входных и выходных значений. Часто, рассматривая исходный код встроенных функций, можно почерпнуть для себя отдельные приемы, которые впоследствии пригодятся при разработке собственных алгоритмов.

[1]Здесь и далее в скобках указаны названия элементов меню англоязычной версии Scilab. О том, как выбрать язык интерфейса Scilab, см. раздел 2.5.

1.7 Упражнения

Упражнение 1.1 (*Установка Scilab*) Установите текущую версию Scilab на свой компьютер (на момент написания данного руководства текущей является версия Scilab 5.2). Полезно также установить более старую версию Scilab для сравнения. Установите Scilab 4.1.2 и рассмотрите отличия между двумя версиями.

Упражнение 1.2 (*Интерактивная справка:* `derivative`) Функция `derivative` предназначена для вычисления производной. Цель данного упражнения состоит в том, чтобы научиться пользоваться различными видами интерактивной справки. Откройте окно справки Scilab, используя меню *Справка* > *Содержание* (*? > Scilab Help*), найдите статью, посвященную функции `derivative`. Затем используйте для этой же цели консоль Scilab.

Упражнение 1.3 (*Использование форума Scilab*) Если после прочтения данного руководства у вас останутся вопросы по использованию Scilab, воспользуйтесь списком рассылки *users@lists.scilab.org* для того, чтобы получить ответы.

2 Основы работы в Scilab

В этом разделе мы сделаем наши первые шаги в Scilab и рассмотрим различные пути работы с пакетом:

- используя консоль Scilab в режиме диалога,
- используя функцию `exec` для выполнения предварительно написанных алгоритмов,
- используя терминал операционной системы и возможности *пакетной* обработки.

2.1 Консоль Scilab

Простейшим способом использования Scilab является непосредственный ввод команд в консоли. Результат выполнения команды при этом отображается сразу же после ее ввода и выполнения. Все примеры в настоящем руководстве можно выполнить, копируя соответствующие команды в консоль, так что читатель может самостоятельно экспериментировать с рассматриваемыми возможностями Scilab. Пошаговое выполнение является наиболее эффективным способом для того, чтобы понять поведение готовых программ, и чаще всего позволяет достаточно быстро перейти к разработке собственных алгоритмов.

В следующем примере мы воспользуемся функцией `disp` для отображения строки "Hello World!":

```
-->s = "Hello World!"
 s  =
 Hello World!
-->disp(s)
 Hello World!
```

Символы "`-->`" представляют собой *приглашение* Scilab и отображаются автоматически, когда Scilab ожидает ввода очередной команды от пользователя. Набрав инструкцию `s="Hello World!"` и нажав клавишу <Enter>, мы укажем Scilab выполнить команду: создать переменную `s`, содержащую заданный текст.

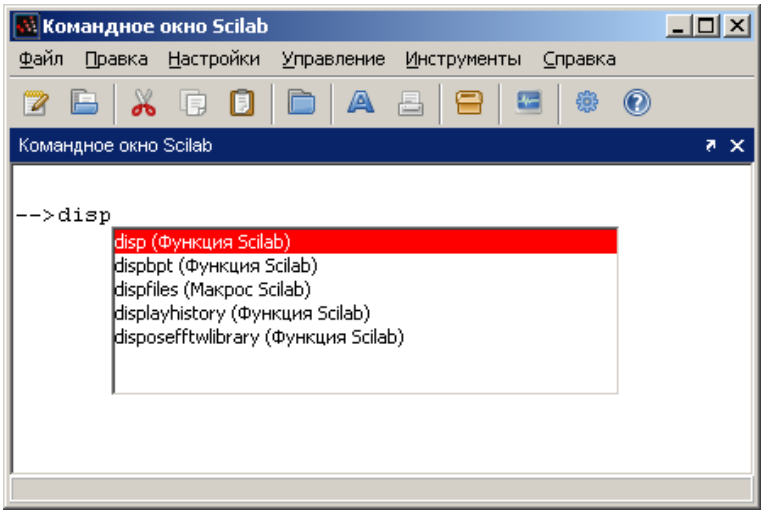

Рис. 4. Подсказка отображается при нажатии клавиши <Tab>.

Реакция Scilab на ввод команды подтверждает, что переменная была создана и ее значение установлено. Теперь, набрав `disp(s)`, можно отобразить содержимое созданной переменной.

Для ввода команд используется клавиатура, точно так же как в обычном текстовом редакторе. Для перемещения курсора в пределах строки используются клавиши ← и →. Для удаления неверно введенного символа используются кнопки <Backspace> и <Suppr>.

Клавиши ↑ и ↓ позволяют перемещаться по истории ранее введенных команд, в том числе в рамках предыдущих сеансов работы с пакетом.

При нажатии на клавишу <Tab> отображается подсказка (рис. 4), где перечислены команды, имена которых начинаются с введенных символов (в данном случае `disp`). С помощью клавиш ↑, ↓ и <Enter> можно выбрать из списка необходимую команду. Подсказки отображаются для имен функций, переменных, файлов и графических дескрипторов, тем самым ускоряя и упрощая работу со Scilab.

2.2 Встроенный редактор скриптов

Реализованный ранее и сохраненный в файл алгоритм, представляющий последовательность инструкций для решения некоторой задачи, называется скриптом.

Версия Scilab 5.2 предлагает новый редактор, упрощающий разработку собственных скриптов. Внешний вид окна редактора показан на рис. 5. В данном случае пользователь редактирует скрипт, содержащий команды рассмотренного выше примера, где в консоль выводилась строка ”Hello World!”.

Запустить редактор можно из меню *Инструменты* > *Текстовый редактор* (*Applications* > *Editor*) либо из консоли, набрав команду

```
-->editor()
```

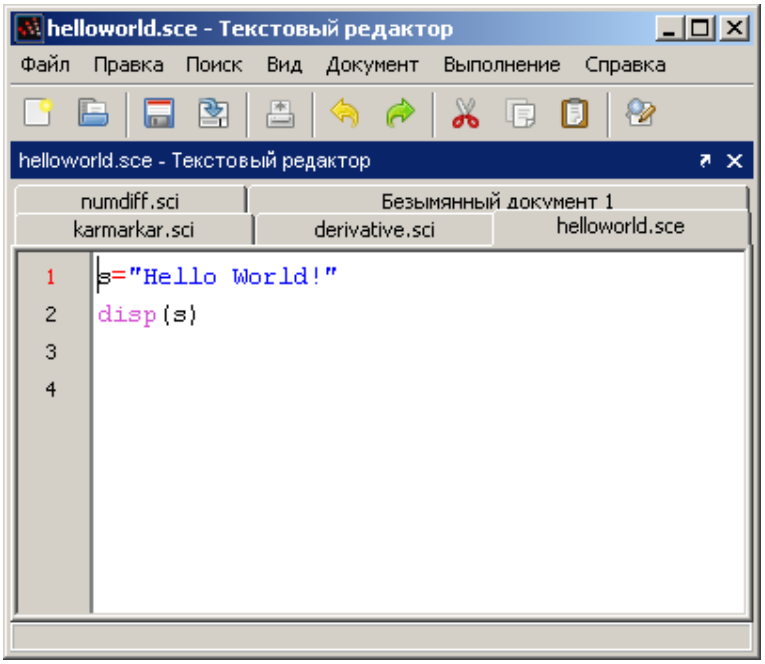

Рис. 5. Окно редактора скриптов.

Редактор позволяет работать с несколькими открытыми файлами одновременно: на рис. 5 открыты 5 файлов.

Наиболее часто используемые команды редактора располагаются в меню *Выполнение* (*Execute*):

- *Загрузить в Scilab* (*Load into Scilab*) - позволяет выполнить все команды некоторого скрипта так, как будто эти команды последовательно вводятся в консоли. При этом результат выполнения инструкций, оканчивающихся символом ";", не отображается.

- *Вычислить выделенное* (*Evaluate Selection*) - позволяет выполнить выделенные инструкции.

- *Выполнить файл в Scilab* (*Execute File Into Scilab*) - загружает на исполнение файл подобно тому, как это делается с использованием функции `exec` (см. раздел 2.4). При этом в консоль будут выводиться лишь результаты работы печатающих функций, например, `disp`.

Меню *Правка* (*Edit*) предлагает полезную возможность автоматического форматирования отступов *Исправить отступы* (*Correct Indentation*). Эта возможность позволяет структурировать текст программы, что существенно упрощает чтение блоков в таких конструкциях как `if`, `for` и т.д.

Выделив несколько строк и нажав правую кнопку мыши (или комбинацию Control+Click в Mac OS), можно отобразить контекстное меню, представленное на рис. 6. Контекстное меню содержит ряд полезных команд:

- *Вычислить выделенное* (*Execute selection in Scilab*) - выполнить выделенные команды;

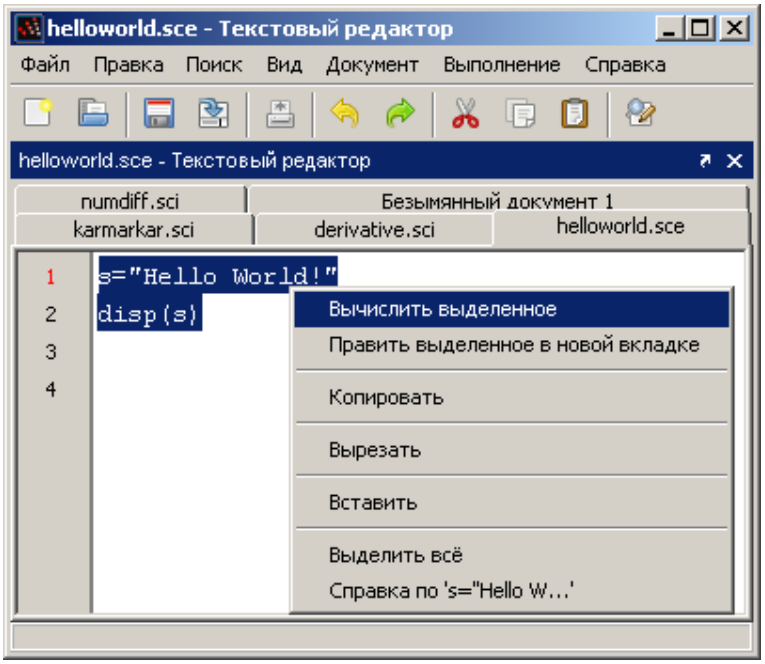

Рис. 6. Контекстное меню в редакторе скриптов.

- *Править выделенное в новой вкладке* (*Edit selection in a new tab*) - открыть новую вкладку и скопировать туда выделенный фрагмент;
- *Справка по '...'* (*Help about '...'*) - отобразить страницу помощи, связанную с выделенной командой.

2.3 Расположение панелей

Графические возможности Scilab начиная с пятой версии основаны на использовании Java, что дает ряд преимуществ, в том числе широкие возможности для управления расположением панелей. Для реализации этого механизма используется библиотека Flexdock [8], разработанная в рамках проекта с открытым исходным кодом и основанная на стандартном наборе классов Swing.

Предположим, что в какой-то момент открыты окно консоли и редактора, как это показано на рис. 7. Переключение между двумя окнами может представлять неудобство, поэтому реализована возможность разместить окно редактора в качестве панели в пределах основного окна Scilab. Все окна в Scilab, включая консоль, текстовый редактор, окно справки и окна вывода графических результатов, могут быть сгруппированы подобным образом. На рис. 8 представлена ситуация, где совместно размещены четыре таких окна.

Для того чтобы поместить некоторое окно рядом с другим, необходимо перетянуть его в область, занимаемую вторым окном. Для этого следует нажать левой клавишей мыши на заголовок перемещаемого окна и, удерживая клавишу мыши, переместить указатель в область, где должна размещаться панель. При этом серыми пунктирными линиями будет отображено предлагаемое положение перетаскиваемого окна (один из четырех вариантов: слева, справа, снизу

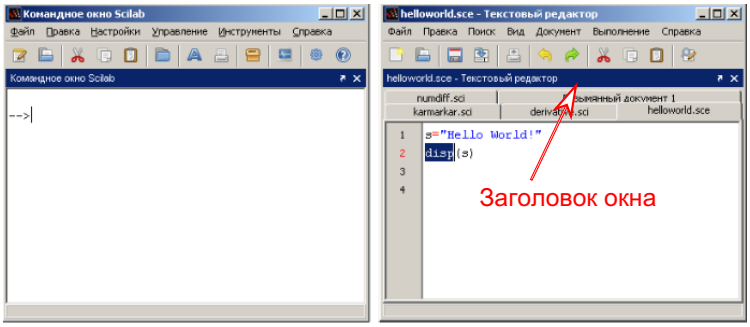

Рис. 7. Строка заголовка перемещаемого окна. Для перемещения выбранного окна, перетащите его в нужную область, удерживая указатель на строке заголовка.

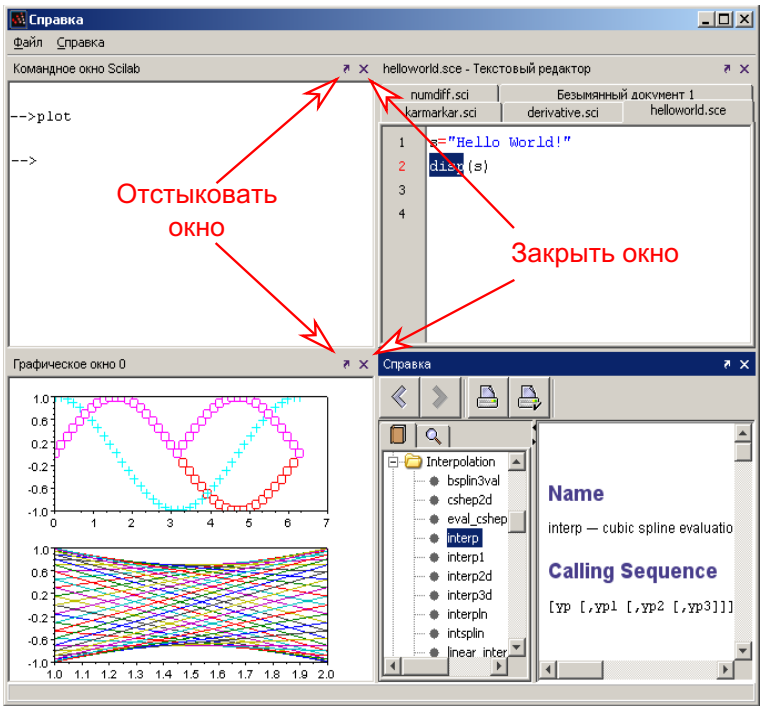

Рис. 8. Управляющие элементы заголовка окна. Закругленная стрелка позволяет отстыковать окно из текущего положения, а крестик служит для закрытия окна.

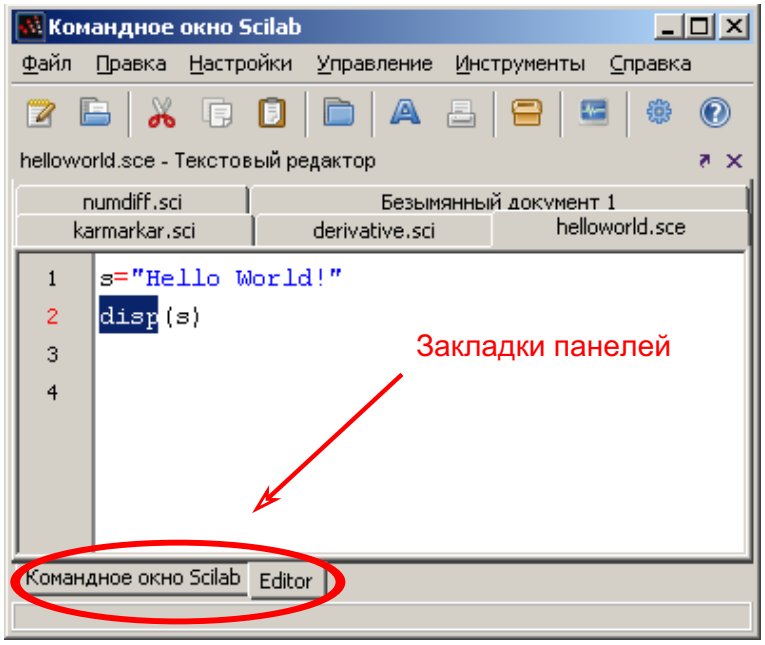

Рис. 9. Вкладки в главном окне Scilab.

или сверху). Выбрав один из вариантов, отпустите левую клавишу мыши, после чего панель займет свое положение (как на рис. 7).

Если отпустить клавишу мыши над центром области, панель будет добавлена в качестве вкладки, как показано на рис. 9.

2.4 Использование команды `exec`

Последовательность из нескольких команд Scilab, предназначенную для многократного выполнения, удобно записать в файл, используя редактор скриптов Scilab. Для того чтобы впоследствии запустить данный скрипт на выполнение из консоли, применяется команда `exec`, после которой указывается имя файла. В зависимости от содержимого файла расширением будет либо `.sce`, либо `.sci`:

- файлы с расширением `.sci` содержат определения функций Scilab - выполнение таких файлов приводит к загрузке данных функций в рабочее окружение пакета, после чего их можно использовать точно так же, как встроенные функции Scilab;

- файлы с расширением `.sce` могут содержать как определения функций, так и исполняемые инструкции - обычно в результате выполнения такого файла вычисляются значения переменных, происходит вывод результатов в консоль, отображение графиков, чтение или запись файлов и т.д.

Представим, что файл `myscript.sce` содержит всего одну строку:

```
disp("Hello World!")
```

Используя функцию `exec` в консоли, мы запускаем данный скрипт на выполнение:

-e инструкция	выполнить *инструкцию*
-f файл	выполнить скрипт, записанный в *файле*
-l язык	выбрать *язык* интерфейса: `fr` для французского, `en` для английского, `ru` для русского. По умолчанию языком интерфейса является английский. Для установки языка по умолчанию, используемого при загрузке Scilab без задания параметра *-l*, применяется команда `setdefaultlanguage`. После выполнения этой команды программу необходимо перезапустить. Команда `getlanguage` позволяет узнать, какой язык используется программой в данный момент.
-mem N	установить начальный размер стека равным *N* байт
-ns	отключить выполнение скрипта `scilab.start` при загрузке Scilab
-nb	отключить вывод приветствия Scilab
-nouserstartup	отключить выполнение пользовательских стартовых скриптов `SCIHOME/.scilab` (в Linux и Mac OS) или `SCIHOME/scilab.ini` (в Windows)
-nw	запустить Scilab в режиме командной строки с подключением дополнительных возможностей (например отображения графиков)
-nwni	запустить Scilab в режиме командной строки без дополнительных возможностей
-version	отобразить текущую версию Scilab и завершить работу

Таблица 1. Параметры командной строки Scilab.

```
-->exec("myscript.sce")
-->disp("Hello World!")
 Hello World!
```

На практике при отладке сложных алгоритмов в интерактивном режиме чаще всего используется комбинация вызовов функций `exec` и `disp`.

2.5 Пакетная обработка

Иным способом работы с пакетом Scilab является его запуск из командной строки операционной системы без отображения оконного интерфейса. Для этого используются параметры командной строки, приведенные в табл. 1.

Вне зависимости от используемой операционной системы, исполняемые файлы размещаются в каталоге `scilab-<version>/bin`, где *<version>* представляет номер установленной на компьютере версии Scilab (например, '5.2.1'). Аргументы командной строки должны следовать за именем исполняемого файла, как будет показано ниже. Параметр *-nw* позволяет отключить отображение окна консоли, при этом ввод команд и вывод результатов будет происходить в текстовом режиме в терминале операционной системы. Опция *-nwni* позволяет

запустить Scilab без графических библиотек. В этом случае функции отображения (например вывод графиков) будут заблокированы, а попытка их использовать приведет к ошибке.

Для удобства пользователя в подкаталоге `bin` основного каталога Scilab присутствуют файлы, предназначенные для запуска Scilab в различных режимах без использования параметров командной строки:

- Для операционных систем семейства Windows в дистрибутивах Scilab поставляются два исполняемых файла. Первый из них, `WScilex.exe`, запускает интерактивную графическую (оконную) консоль Scilab, описанную выше. Именно на этот файл указывает ярлык, создаваемый программой-установщиком Scilab на рабочем столе Windows. Второй исполнимый файл, `Scilex.exe`, служит для запуска консоли в текстовом режиме, использующем стандартные средства терминала Windows, как при использовании опции *-nw*. Указав при запуске `Scilex.exe` параметр *-nwni*, можно полностью отключить загрузку графических библиотек, после чего вывод диаграмм и другие подобные возможности станут недоступны.

- В Linux поведение оболочки Scilab можно контролировать, используя параметры стартового скрипта. По умолчанию, Scilab запускается в графическом режиме, однако аргументы *-nw* и *-nwni* можно использовать для запуска текстового интерфейса. Каталог `bin` в версии Scilab для Linux также содержит два исполняемых файла `scilab-cli` и `scilab-adv-cli`, первый из которых запускает Scilab без оконного интерфейса (эквивалент параметра *-nw*), а второй - без вспомогательных библиотек (эквивалент *-nwni*).

- Поведение версии Scilab для Mac OS аналогично версии для Linux.

В следующем примере для Windows мы запустим файл `Scilex.exe` с параметром *-nwni*. Далее попытаемся выполнить команду `plot` и убедимся, что графические возможности Scilab отключены:

```
D:\Programs\scilab-5.2.0\bin>Scilex.exe -nwni
        -------------------------------------------
                      scilab-5.2.0
                Consortium Scilab (DIGITEO)
              Copyright (c) 1989-2009 (INRIA)
              Copyright (c) 1989-2007 (ENPC)
        -------------------------------------------
Startup execution:
  loading initial environment
-->plot()
       !--error 4
Undefined variable: plot
```

Наиболее полезным параметром командной строки является *-f*, позволяющий выполнить команды, содержащиеся в некотором файле. Такой метод называется *пакетной* обработкой. Предположим, что файл `myscript2.sce` содержит следующие строки (команда `quit` применяется для выхода из Scilab):

```
disp("Hello World!")
quit()
```

По умолчанию в интерактивном режиме Scilab ожидает от пользователя ввода следующей команды. Команды `quit` используется для завершения сессии и выхода из Scilab. Для дальнейшей демонстрации предположим, что мы создали на диске `C:` каталог `scripts` и поместили в него файл скрипта `myscript2.sce`. Следующий пример показывает результат выполнения данного скрипта с использованием опции *-f*. Заметьте, что в данном случае требуется указать полный путь к исполняемому файлу `Scilex.exe`.

```
C:\>D:\Programs\scilab-5.2.0\bin\Scilex.exe -f myscript2.sce
        -------------------------------------------
                      scilab-5.2.0
                Consortium Scilab (DIGITEO)
               Copyright (c) 1989-2009 (INRIA)
               Copyright (c) 1989-2007 (ENPC)
        -------------------------------------------
Startup execution:
  loading initial environment
 Hello World !
```

Любая строка, начинающаяся символами "//", считается комментарием и игнорируется интерпретатором Scilab. Для того чтобы проверить, что по умолчанию Scilab остается в интерактивном режиме, мы исключим команду `quit`, используя символы комментария:

```
disp("Hello World!")
//quit()
```

Если теперь набрать в терминале операционной системы инструкцию `scilex -f myscript2.sce`, Scilab отобразит строку "Hello World!" и перейдет в режим ожидания. Теперь, чтобы выйти, необходимо ввести команду `quit` вручную.

2.6 Упражнения

Упражнение 2.1 (*Использование консоли*) Наберите следующее слово в консоли Scilab и нажмите клавишу <Tab>:

```
atoms
```

Рассмотрите появившееся окно. Нажмите клавишу "I" и затем снова <Tab>. Как изменилось содержимое окна?

Упражнение 2.2 (*Использование функции `exec`*) При создании скриптов функция `exec` часто используется в сочетании с функцией `ls`, отображающей содержимое текущего каталога. Предопределенная константа `SCI` содержит путь к каталогу, в который установлен пакет Scilab, и удобна для записи пути к тому или иному скрипту, поставляемому в составе пакета. Введите следующие команды в консоли Scilab и рассмотрите результаты их выполнения:

```
pwd
SCI
ls(SCI + "/modules")
ls(SCI + "/modules/graphics/demos")
dname = SCI+"/modules/graphics/demos/2d_3d_plots"
filename = fullfile(dname ,"contourf.dem.sce");
exec(filename)
exec(filename);
```

3 Основные элементы языка Scilab

Как уже отмечалось, Scilab относится к числу языков программирования высокого уровня и позволяет эффективно манипулировать сложными структурами данных. В данной главе будут рассмотрены основные возможности языка, а именно создание вещественных матриц, использование элементарных математических функций и т.д. Если бы Scilab ограничивался только этими возможностями, он был бы не более чем продвинутым настольным калькулятором. Конечно, возможности Scilab намного шире, и в последующих разделах мы рассмотрим работу с другими типами данных (логическими переменными, комплексными и целыми числами, а также строками).

Следует с самого начала усвоить, что большинство объектов в Scilab являются матрицами. Все вещественные, комплексные, целочисленные и логические переменные, строки и полиномы представлены в виде матриц. Не являются матрицами списки и другие составные структуры - эти типы данных в настоящем руководстве не рассматриваются.

Матрица представляет набор элементов *одного* типа в виде таблицы, содержащей некоторое число строк и столбцов. Тип элементов определяет набор операций, в которых матрица может участвовать. В данной главе мы рассмотрим работу с матрицами различных типов, начиная с наиболее часто используемого типа - вещественной матрицы.

3.1 Определение вещественных переменных

Scilab предоставляет возможности для работы как с вещественными, так и с комплексными числами. Это может приводить к путанице, если не вполне ясен контекст. Комплексные переменные будут рассмотрены в разделе 3.7 как обобщение вещественных переменных. В большинстве случаев вещественные и комплексные переменные ведут себя одинаково, хотя иногда обработка комплексных величин требует отдельного внимания. Для простоты будем далее рассматривать вещественные переменные, делая необходимые оговорки в том случае, если поведение комплексных переменных имеет особенности применительно к обсуждаемому вопросу.

В данном разделе мы научимся создавать вещественные переменные и выполнять с ними простые манипуляции.

В Scilab, поскольку он является интерпретируемым языком, нет необходимости объявлять переменную до ее использования. Переменная создается в тот момент, когда ей впервые присваивается значение.

В следующем примере мы создаем переменную `x`, которой присваиваем значение 1, после чего выполняем умножение на 2. Оператор "=" в Scilab используется для установки значения переменной (в отличие от оператора "==", который применяется для проверки на равенство).

```
-->x = 1
 x  =
    1.
-->x = x * 2
 x  =
```

+	сложение
-	вычитание
*	умножение
/	деление справа, т.е. $x/y = xy^{-1}$
\	деление слева, т.е. $x\backslash y = x^{-1}y$
^	возведение в степень, т.е. x^y
**	возведение в степень (эквивалентно ^)
'	эрмитово сопряжение (комплексное сопряжение и транспонирование)

Таблица 2. Элементарные математические операторы Scilab.

```
 2.
```

Значение переменной отображается после выполнения каждой инструкции. Если это нежелательно, после инструкции ставится символ ";", как показано в следующем фрагменте:

```
-->y = 1;
-->y = y * 2;
```

Все привычные алгебраические операторы доступны в Scilab (см. табл. 2). Стоит отметить, что оператор возведения в степень представляется символом карет "^", поэтому вычисление x^2 в Scilab выполняется посредством выражения `x^2` или эквивалентного ему выражения `x**2`. Оператор эрмитова сопряжения "'" будет более подробно рассмотрен в разделе 3.7, где мы коснемся работы с комплексными числами. Также оператор "'" является предметом обсуждения в разделе 4.12.

3.2 Имена переменных

Имена переменных в Scilab могут иметь произвольную длину, однако лишь первые 24 символа имени являются значимыми, поэтому во избежание ошибок следует использовать имена длиной до 24 символов. Допустимыми символами в именах переменных являются латинские буквы, цифры, а также символы "%", "_", "#", "!", "$", "?". Следует отметить, что некоторые переменные, имена которых начинаются символом "%", имеют особый смысл в Scilab: как будет показано в разделе 3.5, такие переменные являются предопределенными математическими константами.

Scilab чувствителен к регистру символов, поэтому в следующем примере интерпретатор считает переменные `A` и `a` различными:

```
-->A = 2
 A  =
    2.
-->a = 1
 a  =
    1.
-->A
 A  =
    2.
-->a
```

acos	acosd	acosh	acoshm	acosm	acot	acotd	acoth
acsc	acscd	acsch	asec	asecd	asech	asin	asind
asinh	asinhm	asinm	atan	atand	atanh	atanhm	atanm
cos	cosd	cosh	coshm	cosm	cotd	cotg	coth
cothm	csc	cscd	csch	sec	secd	sech	sin
sinc	sind	sinh	sinhm	sinm	tan	tand	tanh
tanhm	tanm						

Таблица 3. Элементарные математические функции: тригонометрические.

exp	expm	log	log10	log1p	log2	logm	max
maxi	min	mini	modulo	pmodulo	sign	signm	sqrt
sqrtm							

Таблица 4. Элементарные математические функции: прочие.

```
a  =
   1.
```

3.3 Комментарии и продолжение строки

Для оформления комментариев в Scilab применяется синтаксис, заимствованный из языка C++: строки, начинающиеся двумя последовательными символами косой черты "//", считаются комментариями и игнорируются при выполнении. В отличие от C и C++, в Scilab *отсутствуют* многострочные комментарии (ограниченные символами "/*" и "*/").

Если инструкция слишком длинна и не помещается в одну строку, ее можно записать в несколько строк с использованием символа продолжения строки (две точки ".."). Всякая строка, завершающаяся двумя последовательными точками, считается продолженной на следующую строку. Вычисление выражения, записанного в несколько строк, происходит после ввода последней строки.

В следующем фрагменте кода приводится пример комментария и продолжения строки:

```
-->// Это комментарий.
-->x = 1..
-->+ 2..
-->+ 3..
-->+ 4
 x  =
    10.
```

3.4 Элементарные математические функции

В табл. 3 и 4 представлен список элементарных математических функций, доступных в Scilab. Большинство функций в этом списке принимают один аргумент на вход и возвращают единственное значение.

`%i`	мнимая единица i
`%e`	основание натурального логарифма e
`%pi`	число π

Таблица 5. Предопределенные математические константы.

Все элементарные функции *векторизованы* в том смысле, что могут быть применены к матрицам, при этом в результате также получаем матрицы. Это свойство позволяет обрабатывать данные более эффективно, без использования циклов.

В следующем примере мы воспользуемся стандартными функциями `sin` и `cos` для проверки равенства $\cos^2 x + \sin^2 x = 1$:

```
-->x = cos(2)
 x  =
  - 0.4161468
-->y = sin(2)
 y  =
    0.9092974
-->x ^ 2 + y ^ 2
 ans  =
    1.
```

3.5 Предопределенные математические константы

В Scilab имена математических констант начинаются символом "`%`". Примеры приведены в табл. 5.

В следующем примере переменная `%pi` используется для проверки равенства $\cos^2 x + \sin^2 x = 1$:

```
-->c = cos(%pi)
 c  =
  - 1.
-->s = sin(%pi)
 s  =
    1.225D-16
-->c ^ 2 + s ^ 2
 ans  =
    1.
```

Тот факт, что вычисленное значение $\sin \pi$ не равно в точности 0, является следствием ограниченной разрядности двоичного представления чисел с плавающей точкой в памяти компьютера.

3.6 Логический тип

Переменная логического типа может хранить значения *истина* или *ложь*. В Scilab *истина* представляется литералами `%t` или `%T`, а *ложь* - `%f` или `%F`.

В табл. 6 перечислены логические операторы и операторы сравнения, которые используются в пакете. Операторы сравнения принимают на вход данные любого из основных типов (вещественные, комплексные и целые числа, строки)

`a & b`	логическое 'И' (конъюнкция)
`a | b`	логическое 'ИЛИ' (дизъюнкция)
`~a`	логическое отрицание
`a == b`	истина, если `a` равно `b`
`a ~= b` или `a <> b`	истина, если `a` и `b` различаются
`a < b`	истина, если `a` меньше `b`
`a > b`	истина, если `a` больше `b`
`a <= b`	истина, если `a` меньше либо равно `b`
`a >= b`	истина, если `a` больше либо равно `b`

Таблица 6. Логические операторы и операторы сравнения.

`real`	возвращает действительную часть комплексного числа
`imag`	возвращает мнимую часть комплексного числа
`imult`	умножает число на мнимую единицу
`isreal`	проверяет отсутствие мнимой части

Таблица 7. Функции Scilab для работы с комплексными числами.

и возвращают логическое значение. Операторы сравнения также рассматриваются в разделе 4.14, посвященном сравнению матриц.

Следующий пример иллюстрирует выполнение операций с логическими типами:

```
-->a = %T
 a  =
  T
-->b = ( 0 == 1 )
 b  =
  F
-->a & b
 ans  =
  F
```

3.7 Комплексные числа

Комплексные числа в Scilab представляются в виде пары вещественных чисел. Предопределенная константа `%i` содержит значение мнимой единицы i, удовлетворяющее равенству $i^2 = -1$.

Все элементарные функции, рассмотренные выше, работают с комплексными числами. В этом случае возвращаемое значение также будет комплексным. В табл. 7 приведены наиболее часто используемые функции для работы с комплексными числами.

Для примера присвоим переменной `x` значение $1 + i$ и выполним над этой переменной несколько простых операций, в том числе выделим действительную и мнимую части. Напоминаем, что одинарная кавычка "'" обозначает в Scilab оператор эрмитова сопряжения, который при воздействии на скаляр эквивалентен обычному комплексному сопряжению.

int8	int16	int32
uint8	uint16	uint32

Таблица 8. Функции Scilab для создания целочисленных переменных.

```
-->x = 1 + %i
 x  =
    1. + i
-->isreal(x)
 ans  =
  F
-->x'
 ans  =
    1. - i
-->y = 1 - %i
 y  =
    1. - i
-->real(y)
 ans  =
    1.
-->imag(y)
 ans  =
  - 1.
```

Наконец, проверим выполнение равенства $(1+i)(1-i) = 1 - i^2 = 2$:

```
-->x * y
 ans  =
    2.
```

3.8 Целые числа

В Scilab определены шесть типов целочисленных переменных, для создания которых используются функции из табл. 8.

В этом разделе мы рассмотрим основные свойства целых чисел и диапазоны их значений, обратимся к преобразованию целых типов, а также уделим внимание проблемам, связанным с выходом значения за пределы диапазона, и вопросам переносимости программного кода, использующего целые числа.

3.8.1 Обзор целых чисел

Тип целочисленной переменной определяет количество бит, которое используется для представления значения переменной в памяти. В свою очередь, количество используемых бит задает диапазон допустимых значений переменной:

- целое число со знаком, для представления которого используется n бит, может принимать значения из диапазона $[-2^{n-1}, 2^{n-1} - 1]$,
- беззнаковое целое число, занимающее n бит в памяти, может принимать значения из диапазона $[0, 2^n - 1]$.

`y=int8(x)`	8-битовое число со знаком	$[-2^7, 2^7 - 1] = [-128, 127]$
`y=uint8(x)`	8-битовое число без знака	$[0, 2^8 - 1] = [0, 255]$
`y=int16(x)`	16-битовое число со знаком	$[-2^{15}, 2^{15} - 1] = [-32768, 32767]$
`y=uint16(x)`	16-битовое число без знака	$[0, 2^{16} - 1] = [0, 65535]$
`y=int32(x)`	32-битовое число со знаком	$[-2^{31}, 2^{31} - 1]$
`y=uint32(x)`	32-битовое число без знака	$[0, 2^{32} - 1] = [0, 4294967295]$

Таблица 9. Диапазоны значений целых чисел в Scilab.

iconvert	преобразование к целочисленному представлению
inttype	определение типа целого числа

Таблица 10. Функции преобразования целочисленных типов в Scilab.

Например, 8-битовое целое число со знаком (`int8`) может принимать значение в пределах $[-128, 127]$. Таблица 9 отражает соответствие между типом целого числа и диапазоном его значений.

Следующий фрагмент позволяет убедиться, что значение 32-битового целого числа без знака заключено в диапазоне $[0, 2^{32} - 1]$, т.е от 0 до 4294967295:

```
-->format(25)
-->n = 32
 n  =
    32.
-->2 ^ n - 1
 ans  =
    4294967295.
-->i = uint32(0)
 i  =
  0
-->j = i - 1
 j  =
  4294967295
-->k = j + 1
 k  =
  0
```

3.8.2 Преобразование целых типов

Scilab предоставляет две функции для работы с типами целочисленных переменных (таблица 10).

Функция `inttype` позволяет узнать тип переменной, содержащей целое число, возвращая одно из значений, перечисленных в таблице 11.

При выполнении арифметических операций над целыми числами типом результирующего значения становится наиболее широкий из типов операндов. В следующем фрагменте показан пример сложения 8-битового числа `i` (для которого `inttype` возвращает `1`) и 16-битового значения `j` (значение `inttype` равно `2`). Полученное значение `k` имеет тип 16-битового целого числа.

```
-->i = int8(1)
 i  =
```

inttype(x)	Тип переменной
1	8-битовое число со знаком
2	16-битовое число со знаком
4	32-битовое число со знаком
11	8-битовое число без знака
12	16-битовое число без знака
14	32-битовое число без знака

Таблица 11. Коды типов, возвращаемые функцией `inttype`.

```
  1
-->inttype(i)
 ans  =
    1.
-->j = int16(2)
 j  =
  2
-->inttype(j)
 ans  =
    2.
-->k = i + j
 k  =
  3
-->inttype(k)
 ans  =
    2.
```

3.8.3 Выход за пределы диапазона и проблемы переносимости

Поведение целых чисел при выходе за границы допустимого диапазона заслуживает особого внимания, поскольку часто различается в пакетах математических вычислений.

В Scilab значения целых чисел изменяются *циклически*, поэтому при увеличении на 1 за максимальным будет следовать минимальное значение диапазона. Приведенный фрагмент позволяет в этом убедиться:

```
-->uint8(0 + (-4 : 4))
 ans  =
  252  253  254  255  0  1  2  3  4
-->uint8(2 ^ 8 + (-4 : 4))
 ans  =
  252  253  254  255  0  1  2  3  4
-->int8(2 ^ 7 + (-4 : 4))
 ans  =
  124  125  126  127 -128 -127 -126 -125 -124
```

Такое поведение отличает Scilab от ряда математических пакетов, в частности Octave и Matlab, в которых при превышении верхней границы диапазона число остается равным значению верхней границы.

Например, следующая выдержка из рабочей сессии пакета Octave демонстрирует указанные различия при выполнении тех же действий, что и предыдущий фрагмент в Scilab:

```
octave-3.2.4.exe:1> uint8(0 + (-4 : 4))
ans =
  0  0  0  0  0  1  2  3  4
octave-3.2.4.exe:5> uint8(2 ^ 8 + (-4 : 4))
ans =
  252  253  254  255  255  255  255  255  255
octave-3.2.4.exe:2> int8(2 ^ 7 + (-4 : 4))
ans =
  124  125  126  127  127  127  127  127  127
```

Циклическое поведение целых чисел в Scilab способствует большей гибкости в работе, поскольку позволяет избежать избыточных проверок `if`. Вместе с тем, алгоритмы, допускающие выход за пределы диапазона, должны проверяться с особой тщательностью, в особенности при переносе алгоритма из одной вычислительной среды в другую.

3.9 Целые числа и числа с плавающей точкой

По умолчанию числовая переменная в Scilab имеет вещественный тип с плавающей точкой. Для представления значений этого типа в памяти компьютера используется 64 бита. Даже если при создании переменной ей присваивается целое значение, переменная будет иметь тип с плавающей точкой. Целые числа, хранимые таким образом, получили название "flint" (сокращение от "floating point integer") [7]. На практике целые значения в диапазоне $[-2^{52}, 2^{52}]$ можно без особых опасений хранить в вещественных переменных, так как Scilab в этом случае гарантирует точность выполнения операций. Например, в следующем фрагменте мы производим точное сложение двух больших целых чисел, лежащих в пределах диапазона $[-2^{52}, 2^{52}]$ и убеждаемся, что потери точности не происходит:

```
-->a = 2 ^ 40 - 12
 a  =
    1099511627764.
-->b = 2 ^ 45 + 3
 b  =
    35184372088835.
-->c = a + b
 c  =
    36283883716599.
```

Если же в ходе вычислений значение выходит за пределы указанного диапазона, результаты могут быть весьма неожиданными. Например, как демонстрирует следующий фрагмент, при работе с величинами, превышающими 2^{53}, полученное значение оказывается четным независимо от значений операндов:

```
-->(2 ^ 53 + (1 : 10))'
 ans  =
    9007199254740992.
    9007199254740994.
    9007199254740996.
    9007199254740996.
    9007199254740996.
    9007199254740998.
```

```
    9007199254741000.
    9007199254741000.
    9007199254741000.
    9007199254741002.
```

В следующем примере мы вычислим значение 2^{52} с использованием арифметики с плавающей точкой и 16-битовых целых чисел. В первом случае переполнения не происходит, даже несмотря на то, что значение находится на границе диапазона 64-битовых чисел с плавающей точкой. Напротив, во втором случае результат оказывается неверным, так как значение 2^{52} не представимо в рамках 16 бит.

```
-->2 ^ 52
 ans  =
    4503599627370496.
-->uint16(2 ^ 52)
 ans  =
  0
```

Вопросы доступа к элементам матрицы посредством вещественных индексов будут рассмотрены в разделе 4.15.

3.10 Переменная `ans`

Если в инструкции Scilab не указано, какой переменной следует присвоить результат вычисления, результат заносится в специальную переменную `ans`. Переменная `ans`, будучи инициализированной, может использоваться так же, как любая другая переменная в Scilab.

В следующем фрагменте мы вычисляем значение `exp(3)`, не указывая, куда поместить результат, и затем убеждаемся, что значение сохранено в переменной `ans`:

```
-->exp(3)
 ans  =
    20.085536923187667923 68
-->t = log(ans)
 t  =
    3.
```

Переменная `ans`, как правило, используется только в интерактивном режиме для вычисления некоторого численного результата без создания новой переменной. Недостатком такого способа является возможность потери ранее полученных значений в результате перезаписи переменной `ans`. При разработке скриптов полагаться на использование `ans` не следует - вместо этого необходимо применять обычные переменные.

3.11 Строки

Переменной в Scilab можно присвоить строковое значение, заключив его в двойные кавычки ”"”. Конкатенация (слияние) строк осуществляется с помощью оператора ”+”. В следующем примере мы объявляем две строки и склеиваем их при помощи оператора ”+”:

```
-->x = "foo"
 x  =
 foo
-->y = "bar"
 y  =
 bar
-->x + y
 ans  =
 foobar
```

Для обработки строк Scilab предоставляет широкий набор возможностей, включая функции для работы с регулярными выражениями, однако в данном документе эти функции рассматриваться не будут.

3.12 Динамическая типизация переменных

Тип переменной в Scilab может динамически меняться в зависимости от присвоенного ей значения. Таким образом, возможно, к примеру, создать переменную, содержащую вещественное число, а затем присвоить этой переменной строковое значение:

```
-->x = 1
 x  =
    1.
-->x + 1
 ans  =
    2.
-->x = "foo"
 x  =
 foo
-->x + "bar"
 ans  =
 foobar
```

Необходимо еще раз подчеркнуть, что Scilab является нетипизированным языком, поэтому нет необходимости указывать тип переменной до присвоения ей значения, и более того, тип может изменяться в течение времени жизни переменной.

3.13 Упражнения

Упражнение 3.1 (*Приоритет операторов*) Каков результат вычисления следующих выражений (проверьте свои ответы с помощью Scilab)?

```
2 * 3 + 4
2 + 3 * 4
2 / 3 + 4
2 + 3 / 4
```

Упражнение 3.2 (*Скобки*) Каков результат вычисления следующих выражений (проверьте свои ответы с помощью Scilab)?

```
2 * (3 + 4)
(2 + 3) * 4
(2 + 3) / 4
3 / (2 + 4)
```

Упражнение 3.3 (*Экспоненциальная запись чисел*) Каков результат вычисления следующих выражений (проверьте свои ответы с помощью Scilab)?

```
1.23456789d10
1.23456789e10
1.23456789e-5
```

Упражнение 3.4 (*Функции*) Каков результат вычисления следующих выражений (проверьте свои ответы с помощью Scilab)?

```
sqrt(4)
sqrt(9)
sqrt(-1)
sqrt(-2)
exp(1)
log(exp(2))
exp(log(2))
10^2
log10(10^2)
10^log10(2)
sign(2)
sign(-2)
sign(0)
```

Упражнение 3.5 (*Тригонометрические функции*) Каков результат вычисления следующих выражений (проверьте свои ответы с помощью Scilab)?

```
cos(0)
sin(0)
cos(%pi)
sin(%pi)
cos(%pi/4) - sin(%pi/4)
```

4 Матрицы

Матрицы играют ключевую роль в Scilab. В данном разделе будет рассмотрено создание матрицы, изменение ее свойств, доступ к элементам матрицы и высокоуровневые операции, работающие с группами элементов.

4.1 Обзор

Базовой структурой данных в Scilab является матрица. Всякая матрица характеризуется своим размером (т.е. числом строк и столбцов) и типом содержащихся в ней значений. Элементами матрицы могут являться вещественные, комплексные или целые числа, логические значения, строки или полиномы. Если две матрицы имеют совпадающее число строк и столбцов, говорят, что они имеют одинаковый *размер*.

Частным случаем матриц являются векторы, в которых число строк либо столбцов равно 1. Собственно скалярные величины в Scilab отсутствуют - скалярное значение представляется матрицей 1×1. Поэтому в данной главе мы рассматриваем работу с матрицами, подразумевая, что аналогичные действия

применимы и к векторам (т.е. матрицам $n \times 1$ или $1 \times n$) и скалярам (матрицам 1×1).

Необходимо отметить, что Scilab создавался в первую очередь для работы с матрицами вещественных значений, и поэтому содержит большое число функций для работы с вещественнозначными матрицами.

В числе задач проектирования Scilab также стояла оптимизация скорости выполнения таких операций. Для этого было разработано специальное внутреннее представление матриц, позволяющее интерпретатору эффективно манипулировать ими. Основные операции линейной алгебры, такие как сложение, вычитание, транспонирование и скалярное произведение, выполняются оптимизированными встроенными функциями. Эти операции обозначаются в Scilab символами "+","-", "'" и "*".

При использовании высокоуровневых операторов и функций практически отпадает необходимость в реализации циклов, которые, помимо прочего, выполняются существенно медленнее (от 10 до 100 раз), нежели встроенные функции. Данное свойство Scilab носит название *векторизации*. Для написания максимально эффективных алгоритмов в Scilab необходимо всегда пользоваться имеющимися высокоуровневыми возможностями, так чтобы каждой командой по возможности обрабатывалась целая матрица, а не один ее элемент.

Более сложные задачи линейной алгебры, такие как решение систем линейных уравнений $Ax = b$, различные виды разложений (например, гауссово разложение с перестановками $PA = LU$), поиск собственных значений и векторов, также выполняются встроенными оптимизированными функциями. Пользователю эти возможности доступны посредством операторов Scilab "/" и "\", а также специальных функций (например, функции `spec`, вычисляющей собственные значения и векторы для заданной матрицы).

4.2 Создание вещественных матриц

Определение вещественной матрицы в Scilab выглядит следующим образом:

```
A = [a11, a12, ..., a1n; ... ; an1, an2, ..., ann].
```

где:

- квадратные скобки "[" и "]" обозначают начало и конец перечисления элементов матрицы,
- запятой "," отделяются элементы матрицы, находящиеся в одной строке,
- точка с запятой ";" разделяет строки матрицы.

Пробелы при отделении значений запятыми не являются обязательными, но улучшают читаемость. Символы "..." обозначают пропущенные значения. В следующем примере мы создаем матрицу 2×3, содержащую действительные значения:

```
-->A = [1 , 2 , 3 ; 4 , 5 , 6]
 A  =
    1.    2.    3.
    4.    5.    6.
```

`eye`	единичная матрица
`linspace`	вектор равноотстоящих значений
`ones`	матрица, все элементы которой равны 1
`zeros`	нулевая матрица
`testmatrix`	специальные типы матриц (Гильберта, Франка и пр.)
`rand`	генератор случайных чисел
`grand`	генератор случайных чисел с возможностью выбора распределения

Таблица 12. Функции для создания матриц.

Можно воспользоваться упрощенным синтаксисом, не требующим применения символов ”,” и ”;”. В этом случае матрица записывается построчно, а значения в пределах одной строки разделяются пробелом, как показано ниже:

```
A = [a11 a12 ... a1n
a21 a22 ... a2n
...
an1 an2 ... ann]
```

Это позволяет существенно облегчить ввод матриц и повысить его наглядность:

```
-->A = [1 2 3
-->4 5 6]
 A  =
    1.    2.    3.
    4.    5.    6.
```

Многострочная запись матриц полезна при выводе матрицы в файл, поскольку упрощает просмотр и проверку значений, а также обратное считывание матрицы в Scilab.

Несколько функций Scilab позволяют создавать матрицы заданного размера, принимая в качестве своих параметров значения числа строк и столбцов. Среди таких функций (см. табл. 12) наиболее часто используются `eye`, `zeros` и `ones`:

```
-->A = ones(2, 3)
 A  =
    1.    1.    1.
    1.    1.    1.
```

4.3 Пустая матрица []

Пустую матрицу (размером 0×0) можно создать используя пустые квадратные скобки, как показано в следующем примере:

```
-->A = []
 A  =
     []
```

Этот синтаксис также позволяет очистить содержимое ранее созданной матрицы, чтобы освободить занимаемую память:

```
-->A = ones(100, 100);
-->A = []
 A  =
     []
```

4.4 Определение размера матрицы

Функции, приведенные в табл. 13, позволяют проверить и изменить размер матрицы.

Вызов функции `size` для некоторой матрицы возвращает два параметра `nr` и `nc`, значения которых равны числу строк и столбцов в данной матрице:

```
-->A = ones(2, 3)
 A  =
    1.    1.    1.
    1.    1.    1.
-->[nr, nc] = size(A)
 nc  =
    3.
 nr  =
    2.
```

Функция `size` может быть полезна при определении собственных функций в тех случаях, когда обработка аргументов зависит от размерностей аргументов. В качестве примера можно привести функцию, вычисляющую норму, которая будет различным образом действовать при получении вектора и матрицы.

Функция `size` также имеет альтернативный синтаксис:

```
nr = size( A , sel )
```

Возвращаемое значение в этом случае определяется вторым параметром `sel` функции:

- при `sel=1` или `sel="r"` возвращается число строк,
- при `sel=2` или `sel="c"` возвращается число столбцов,
- при `sel="*"` возвращается общее число элементов в матрице, равное числу строк, умноженному на число столбцов.

В приведенном ниже фрагменте с использованием функции `size` подсчитывается общее число элементов в матрице:

```
-->A = ones(2, 3)
 A  =
    1.    1.    1.
    1.    1.    1.
-->size(A, "*")
 ans  =
    6.
```

`size`	определить размер матрицы
`matrix`	изменить размер матрицы
`resize_matrix`	создать новую матрицу заданного размера и скопировать в нее элементы из исходной матрицы

Таблица 13. Функции для проверки и изменения свойств матрицы.

4.5 Работа с элементами матрицы

Scilab предоставляет несколько способов доступа к элементам матрицы `A`:

- используя имя матрицы `A`, можно оперировать матрицей как целым,
- для поэлементных манипуляций применяется запись `A(i,j)`,
- для доступа к группе элементов, индексы которых лежат в некотором диапазоне, служит оператор ":" - он будет рассмотрен в следующем разделе.

Для операций с матрицами используются имена содержащих эти матрицы переменных. Все элементарные алгебраические операции применимы к матрицам одинакового размера, как это показано ниже на примере вычисления суммы:

```
-->A = ones(2, 3)
 A  =
    1.    1.    1.
    1.    1.    1.
-->B = 2 * ones(2, 3)
 B  =
    2.    2.    2.
    2.    2.    2.
-->A + B
 ans  =
    3.    3.    3.
    3.    3.    3.
```

Получить доступ к отдельному элементу матрицы можно с помощью записи `A(i,j)`, при условии что `i` и `j` представляют собой допустимые значения индексов. В Scilab первый элемент имеет индекс 1, в отличие от многих языков программирования, где индексы элементов массива начинаются с 0. Допустим, к примеру, что `A` представляет собой матрицу $nr \times nc$, где nr есть число строк, а nc - число столбцов. Запись `A(i,j)` имеет смысл только при $1 \leq i \leq nr$ и $1 \leq j \leq nc$. Если значение хотя бы одного из двух индексов не является допустимым, выдается сообщение об ошибке:

```
-->A = ones(2, 3)
 A  =
    1.    1.    1.
    1.    1.    1.
-->A(1, 1)
 ans  =
    1.
-->A(12, 1)
         !--error 21
```

```
Invalid index.
-->A(0, 1)
        !--error 21
Invalid index.
```

Прямой доступ к элементам матрицы с использованием синтаксиса `A(i,j)` следует использовать только в том случае, если отсутствует соответствующая высокоуровневая функция Scilab. В большинстве случаев такая функция может быть найдена. Scilab предоставляет богатый набор простых и эффективных команд, основанных на возможностях векторизации. Одна из таких возможностей заключена в использовании оператора ":", который оказывается чрезвычайно полезным на практике.

4.6 Оператор ":"

Простейшая запись инструкции с оператором ":" выглядит следующим образом:

```
v = i : j
```

где `i` - это минимальное, а `j` - максимальное значение индекса, причем `i`≤`j`. Результатом является вектор $v = (i; i+1; ...; j)$. Следующий пример демонстрирует генерацию вектора, элементы которого пробегают значения от 2 до 4:

```
-->v = 2 : 4
 v  =
    2.    3.    4.
```

Расширенный вариант записи оператора ":" позволяет указать приращение индекса, называемое *шагом*:

```
v = i : s : j
```

где `i` и `j` имеют тот же смысл, что и ранее, а `s` представляет собой шаг. Данная команда создает вектор $v = (i; i+s; i+2s; ...; i+ns)$, где n - максимальное целое число, такое что $i + ns \leq j$. Если $(j - i)$ делится нацело на s, то максимальное значение индекса совпадает с j, в остальных случаях $i + ns < j$. Величина шага `s` может быть как положительной, так и отрицательной.

В следующем примере создается вектор со значениями от 3 до 10 с шагом 2:

```
-->v = 3 : 2 : 10
 v  =
    3.    5.    7.    9.
```

Последнее значение в данном случае равно 9, т.е. меньше `j`=10.

Следующий фрагмент иллюстрирует результат действия оператора ":" при отрицательном значении шага. В первом случае мы создаем вектор значений, убывающих от 10 до 4. Во втором случае результатом выполнения оператора становится пустая матрица, поскольку начальное значение меньше конечного.

```
-->v = 10 : -2 : 3
 v  =
    10.    8.    6.    4.
-->v = 3 : -2 : 10
```

```
 v  =
     []
```

Используя вектор значений индекса, можно обращаться к элементам матрицы в определенных диапазонах строк и столбцов:

```
A(i : j , k : l)
```

где `i`, `j`, `k` ,`l` - начальные и конечные значения индексов. Полный синтаксис `A(i:s:j,k:t:l)` включает также задание шагов `s` и `t`.

Предположим, что `A` является матрицей размера 4×5. Пусть необходимо обратиться к элементам $a_{i,j}$, причем $i = 1, 2$, а $j = 3, 4$. В Scilab для этого потребуется всего одна инструкция, `A(1:2,3:4)`, как показано в следующем фрагменте:

```
-->A = testmatrix("hilb", 5)
 A  =

    25.     - 300.       1050.     - 1400.       630.
  - 300.      4800.    - 18900.      26880.    - 12600.
    1050.   - 18900.     79380.    - 117600.     56700.
  - 1400.     26880.   - 117600.     179200.   - 88200.
    630.    - 12600.     56700.    - 88200.      44100.
-->A(1 : 2, 3 : 4)
 ans  =
    1050.    - 1400.
  - 18900.     26880.
```

В некоторых случаях значения индексов должны вычисляться динамически (например в цикле). Полученные в результате векторы могут использоваться для адресации элементов матрицы точно так же, как числовые литералы:

```
A(vi, vj),
```

Данная запись выделяет элементы матрицы `A`, индексы которых принадлежат векторам `vi` и `vj`. Следующий пример иллюстрирует эту возможность:

```
-->A = testmatrix("hilb", 5)
 A  =

    25.     - 300.       1050.     - 1400.       630.
  - 300.      4800.    - 18900.      26880.    - 12600.
    1050.   - 18900.     79380.    - 117600.     56700.
  - 1400.     26880.   - 117600.     179200.   - 88200.
    630.    - 12600.     56700.    - 88200.      44100.
-->vi = 1 : 2
 vi  =
    1.    2.
-->vj = 3 : 4
 vj  =
    3.    4.
-->A(vi, vj)
 ans  =
    1050.    - 1400.
  - 18900.     26880.
-->vi = vi + 1
 vi  =
```

`A`	матрица целиком
`A(:,:)`	матрица целиком
`A(i:j,k)`	элементы матрицы в `k`-ом столбце с `i`-ой по `j`-ую строку
`A(i,j:k)`	элементы матрицы в `i`-ой строке с `j`-ого по `k`-ый столбец
`A(i,:)`	`i`-ая строка матрицы
`A(:,j)`	`j`-ый столбец матрицы

Таблица 14. Различные варианты использования оператора ":" для доступа к элементам матрицы.

```
    2.    3.
-->vj = vj + 1
 vj  =
    4.    5.
-->A(vi, vj)
 ans  =
    26880.   - 12600.
  - 117600.    56700.
```

На основе оператора ":" может быть построено большое число комбинаций. Наиболее часто используемые из них представлены в табл. 14.

Например, следующий фрагмент демонстрирует использование оператора ":" для перестановки строк матрицы:

```
-->A = testmatrix("hilb", 3)
 A  =
    9.   - 36.     30.
  - 36.    192.  - 180.
    30.  - 180.    180.
-->A([1 2], :) = A([2 1], :)
 A  =
  - 36.    192.  - 180.
    9.   - 36.     30.
    30.  - 180.    180.
```

Обменять местами столбцы матрицы `A` можно при помощи инструкции `A(:, [3 1 2])`.

В этом разделе нами были рассмотрены несколько важных с практической точки зрения случаев использования оператора ":". Оператор ":" повсеместно используется при написании алгоритмов, для которых скорость выполнения имеет решающее значение. Именно использованием оператора ":" достигается *векторизация* алгоритмов - одна из ключевых возможностей Scilab, которой уделяется особое внимание в данном руководстве.

4.7 Генерация единичной матрицы. Функция `eye`

Функция `eye` позволяет создать единичную матрицу необходимого размера. Название этой функции в английском произношении созвучно названию буквы I, используемой в англоязычной литературе для обозначения единичной матрицы, и было выбрано таким с целью избежать путаницы с традиционным обозначе-

нием индексной переменной `i` или мнимой единицы[1].

Следующий фрагмент демонстрирует, как прибавить число 3 к диагональным элементам матрицы `A`:

```
-->A = ones(3, 3)
 A  =
    1.    1.    1.
    1.    1.    1.
    1.    1.    1.
-->B = A + 3 * eye()
 B  =
    4.    1.    1.
    1.    4.    1.
    1.    1.    4.
```

Далее мы создаем единичную матрицу `B` с использованием функции `eye`, причем размер создаваемой матрицы будет равен размеру матрицы `A`, переданной функции `eye` в качестве параметра. Сама матрица `A` при этом не изменяется:

```
-->A = ones(2, 2)
 A  =
    1.    1.
    1.    1.
-->B = eye(A)
 B  =
    1.    0.
    0.    1.
```

Наконец, используя синтаксис `eye(m,n)`, можно явно указать количество строк `m` и столбцов `n` в генерируемой матрице.

4.8 Динамическое изменение размера матрицы

Матрицы могут динамически изменять свой размер в процессе выполнения программы, что позволяет им приспосабливаться к хранимым данным.

Для иллюстрации создадим матрицу 2×3:

```
-->A = [1 2 3; 4 5 6]
 A  =
    1.    2.    3.
    4.    5.    6.
```

Далее мы добавим в эту матрицу элемент с индексами $(3, 1)$, равный 7. При этом в матрице будет создана третья строка, и первому элементу в ней будет присвоено указанное значение, а остальные элементы строки примут значение 0.

```
-->A(3, 1) = 7
 A  =
    1.    2.    3.
    4.    5.    6.
    7.    0.    0.
```

[1] В Scilab мнимая единица является предопределенной константой и обозначается `%i`.

`A(i,$)`	элемент на пересечении `i`-ой строки и последнего (`nc`-ого) столбца
`A($,j)`	элемент на пересечении последней (`nr`-ой) строки и `j`-ого столбца
`A($-i,$-j)`	элемент на пересечении строки `nr-i` и столбца `nc-j`

Таблица 15. Использование оператора "`$`" для доступа к элементам матрицы `A` размера $nr \times nc$.

Таким образом, размер матрицы может увеличиваться динамически. Далее мы покажем, что возможно и его динамическое уменьшение. Для этого, используя обозначение пустой матрицы `[]`, удалим третий столбец матрицы `A`:

```
-->A(:, 3) = []
 A  =
    1.    2.
    4.    5.
    7.    0.
```

Также можно полностью изменить размер матрицы посредством функции `matrix`. Эта функция трансформирует исходную матрицу в матрицу другого размера, копируя элементы столбец за столбцом. В следующем примере матрица `A` размером 3×2 преобразуется в вектор-строку с 6 элементами:

```
-->B = matrix(A, 1, 6)
 B  =
    1.    4.    7.    2.    5.    0.
```

4.9 Оператор "$"

Обычно при обращении к элементам матрицы отсчет индексов ведется от первой строки и первого столбца. Оператор "`$`" позволяет адресовать элементы матрицы, отсчитывая индексы от последней строки или столбца в зависимости от контекста.

Различные варианты использования оператора "`$`" приведены в табл. 15.

Следующий фрагмент демонстрирует использование оператора "`$`" для доступа к элементу `A(2,1) = A(nr-1,nc-2) = A($-1,$-2)` матрицы 3×3:

```
-->A=testmatrix("hilb", 3)
 A  =
    9.    - 36.     30.
  - 36.     192.  - 180.
    30.   - 180.    180.
-->A($ - 1, $ - 2)
 ans  =
  - 36.
```

Оператор "`$`" также позволяет динамически увеличить размер матрицы. В следующем фрагменте в Гильбертову матрицу добавляется новая строка:

```
-->A($ + 1, :) = [1 2 3]
 A  =
    9.    - 36.     30.
```

```
 - 36.     192.   - 180.
   30.   - 180.     180.
   1.      2.       3.
```

Чаще всего оператор "$" используется в составе выражения $+1, которое позволяет добавлять в матрицу строки или столбцы. Такой способ оказывается удобным, поскольку избавляет от необходимости постоянно отслеживать текущее число строк и столбцов. Однако пользоваться им необходимо аккуратно - лишь в тех случаях, когда размер матрицы неизвестен заранее. Поскольку при каждом добавлении строки или столбца происходит выделение памяти под матрицу большего размера и копирование всех элементов из исходной матрицы, злоупотребление этой возможностью может привести к существенному снижению производительности программы.

4.10 Арифметические операции

Все арифметические операторы, такие как "+", "-", "*" и "/", работают с вещественными матрицами. Ниже мы рассмотрим действие каждого оператора в отдельности, дабы устранить любую причину для путаницы.

Операции сложения "+" и вычитания "-" выполняются в соответствии с обычными правилами линейной алгебры. В следующем примере показано сложение двух матриц размерности 2×2:

```
-->A = [1 2
-->3 4]
 A  =
    1.     2.
    3.     4.
-->B = [5 6
-->7 8]
 B  =
    5.     6.
    7.     8.
-->A + B
 ans  =
    6.     8.
    10.    12.
```

Сложение матриц возможно только в случае одинаковых размерностей обоих операндов. Приведенный ниже фрагмент демонстрирует попытку сложения матриц размерностей 2×2 и 2×3, приводящую к ошибке:

```
-->A = [1 2
-->3 4]
 A  =
    1.     2.
    3.     4.
-->B = [1 2 3
-->4 5 6]
 B  =
    1.     2.     3.
    4.     5.     6.
-->A + B
     !--error 8
```

+	сложение	.+	поэлементное сложение
-	вычитание	.-	поэлементное вычитание
*	умножение	.*	поэлементное умножение
/	деление справа	./	поэлементное деление справа
\	деление слева	.\	поэлементное деление слева
^ или **	возведение в степень	.^	поэлементное возведение в степень
'	эрмитово сопряжение (комплексное сопряжение и транспонирование)	.'	транспонирование без сопряжения

Таблица 16. Элементарные матричные операции и их поэлементные варианты.

```
Inconsistent addition.
```

Исключение составляет ситуация, когда одно из слагаемых является скаляром (т.е. матрицей 1×1). В этом случае значение скаляра будет прибавлено к каждому элементу второго операнда:

```
-->A = [1 2
-->3 4]
 A  =
    1.    2.
    3.    4.
-->A + 1
 ans  =
    2.    3.
    4.    5.
```

Элементарные операторы для работы с матрицами представлены в табл. 16.

Оператор деления в Scilab реализован в двух вариантах: деление справа, обозначаемое символом "/", и деление слева, которому соответствует символ "\". Результат деления справа $X = A/B = AB^{-1}$ представляет собой решение уравнения $XB = A$. Результат деления слева $X = A\backslash B = A^{-1}B$ является решением уравнения $AX = B$. В случае когда A не является квадратной матрицей, результат деления слева $A\backslash B$ дает решение соответствующей задачи о наименьших квадратах.

В табл. 16 представлены также поэлементные операторы, работающие с отдельными элементами матриц. Эти операторы будут подробнее рассмотрены в следующем разделе.

4.11 Поэлементные операции

Точка "." перед соответствующим арифметическим оператором обозначает, что операция производится поэлементно. Например, при использовании обычного оператора умножения "*", элементы матрицы-произведения `C=A*B` вычисляются по известной формуле $c_{ij} = \sum_{k=1,n} a_{ik}b_{kj}$, при этом каждый элемент представляет собой сумму произведений элементов строки первой матрицы и столбца второй матрицы. В случае поэлементного умножения `C=A.*B`, элемент c_{ij} пред-

ставляет собой простое произведение соответствующих элементов a_{ij} и b_{ij}. Следующий фрагмент демонстрирует эти различия для двух конкретных матриц:

```
-->A = ones(2, 2)
 A  =
    1.    1.
    1.    1.
-->B = 2 * ones(2, 2)
 B  =
    2.    2.
    2.    2.
-->A * B
 ans  =
    4.    4.
    4.    4.
-->A .* B
 ans  =
    2.    2.
    2.    2.
```

Особый смысл придает точка ”.” оператору эрмитова сопряжения. Об этом пойдет речь в следующем разделе.

4.12 Эрмитово сопряжение и транспонирование

Может возникнуть некоторая путаница между операторами ” ’ ” и ” .’ ”, первый из которых сопрягает и транспонирует матрицу, а второй - только транспонирует ее. Для вещественнозначных матриц действие обоих операторов эквивалентно и сводится к транспонированию. Различия начинают проявляться, когда эти операторы применяются в отношении матриц, содержащих комплексные значения.

В случае комплекснозначных матриц эрмитово сопряжение матрицы `Z` посредством оператора ” ’ ” дает матрицу `A=Z’`, значения элементов которой определяются выражением $A_{jk} = X_{kj} - iY_{kj}$, где X и Y - матрицы действительных и вещественных частей Z соответственно, а i - мнимая единица. При транспонировании без сопряжения, выполняемом с использованием оператора ” .’ ”, значения элементов результирующей матрицы $A_{jk} = X_{kj} + iY_{kj}$ отличаются от полученных выше знаками мнимых частей.

В следующем примере операторы ” ’ ” и ” .’ ” применяются к асимметричной комплекснозначной матрице, так что различия в их действии становятся очевидны:

```
-->A = [1 2; 3 4] + %i * [5 6; 7 8]
 A  =
    1. + 5.i    2. + 6.i
    3. + 7.i    4. + 8.i
-->A'
 ans  =
    1. - 5.i    3. - 7.i
    2. - 6.i    4. - 8.i
-->A.'
 ans  =
    1. + 5.i    3. + 7.i
```

```
    2. + 6.i    4. + 8.i
```

Далее демонстрируется применение обоих операторов в случае вещественнозначной матрицы - результаты одинаковы:

```
-->B = [1 2; 3 4]
 B  =
    1.    2.
    3.    4.
-->B'
 ans  =
    1.    3.
    2.    4.
-->B.'
 ans  =
    1.    3.
    2.    4.
```

Поскольку указанные различия часто становятся источником ошибок, необходимо четко усвоить назначение каждого из операторов и применять оператор " .' " в случае, когда требуется только транспонирование (будь то вещественная или комплексная матрица), и оператор " ' " - когда необходимо выполнить транспонирование и сопряжение.

4.13 Умножение векторов

Допустим, $\mathbf{u} \in \mathbb{R}^n$ является вектором-столбцом, а $\mathbf{v}^T \in \mathbb{R}^n$ - вектором-строкой. Тогда элементы матрицы $A = \mathbf{u}\mathbf{v}^T$ представляют собой произведения $A_{ij} = u_i v_j$. Следующий фрагмент демонстрирует вычисление матрицы `A`:

```
-->u = [1
-->2
-->3]
 u  =
    1.
    2.
    3.
-->v = [4 5 6]
 v  =
    4.    5.    6.
-->u * v
 ans  =
    4.     5.     6.
    8.     10.    12.
    12.    15.    18.
```

Обычно в курсах линейной алгебры рассматриваются только векторы-столбцы, обозначаемые наподобие $\mathbf{u} \in \mathbb{R}^n$. В этом случае соответствующий вектор-строка будет записан как $\mathbf{u}^T$. В Scilab переменная может содержать непосредственно вектор-строку, а значит при умножении транспонировать этот вектор не понадобится.

Также источником ошибок может служить и обратное предположение, что в переменной хранится вектор-строка, в то время как на самом деле переменная содержит вектор-столбец. Поэтому всякий алгоритм, работающий только

`and(A,''r'')`	постолбцовое "И"
`and(A,''c'')`	построчное "И"
`or(A,''r'')`	постолбцовое "ИЛИ"
`or(A,''c'')`	построчное "ИЛИ"

Таблица 17. Функции построчного и постолбцового сравнения элементов матрицы.

с определенным типом матриц, должен проверять размерности входных аргументов и генерировать сообщение об ошибке в случае несоответствия.

4.14 Сравнение вещественных матриц

Сравнение двух матриц возможно при условии одинаковых размеров. При выполнении этого условия операторы, представленные в табл. 6, могут применяться и к матричным операндам. В результате сравнения двух матриц образуется матрица логических значений, где каждый элемент представляет результат сравнения соответствующих элементов исходных матриц. Логические операторы "`&`", "`|`" и др. также применимы к матрицам. Кроме того, матрицы логических значений могут выступать в качестве параметров функций `and` и `or`, смысл которых раскрывает табл. 17.

В следующем примере мы определяем матрицу `A` и сравниваем ее с числом 3 (при этом каждый элемент матрицы сравнивается с этим значением). Затем мы создаем вторую матрицу `B` и сравниваем ее с первой. Наконец, используя функцию `or`, мы выполняем построчное сравнение, в результате которого получаем вектор логических значений, указывающий, какие из столбцов матрицы `A` содержат элементы, превосходящие соответствующие элементы матрицы `B`.

```
-->A = [1 2 7
-->6 9 8]
 A  =
    1.    2.    7.
    6.    9.    8.
-->A > 3
 ans  =
  F F T
  T T T
-->B = [4 5 6
-->7 8 9]
 B  =
    4.    5.    6.
    7.    8.    9.
-->A > B
 ans  =
  F F T
  F T F
-->or(A > B, "r")
 ans  =
  F T T
```

4.15 Числа с плавающей точкой в качестве индексов

Предположим, что матрица `A` имеет размер 2×2. Для того чтобы обратиться к элементам матрицы, могут применяться как жестко заданные в тексте программы индексы, так и значения, вычисляемые динамически в ходе выполнения, в том числе нецелые, как показано далее:

```
-->A = testmatrix("hilb", 2)
 A  =
    4.   - 6.
  - 6.     12.
-->A( 2 , [1.0 1.1 1.5 1.9] )
 ans  =
  - 6.   - 6.   - 6.   - 6.
```

Этот пример демонстрирует, что все значения 1.0, 1.1, 1.5 и 1.9 округляются до 1, как при использовании функции `int`, усекающей дробную часть числа. Например, результатом каждого из следующих вызовов: `int(1.0)`, `int(1.1)`, `int(1.5)` и `int(1.9)` - будет значение 1, а выражения `int(-1.0)`, `int(-1.1)`, `int(-1.5)` и `int(-1.9)` возвращают число -1[1].

Заметим, что округление происходит в соответствии с особенностями работы функции `int`, а не `floor`, которая в отличие от `int` отрицательные значения округляет в меньшую сторону. В самом деле, предположим, что `A` представляет матрицу 4×4, например, созданную в результате вызова

```
A = testmatrix("hilb", 4)
```

Функция `triu(A,k)` возвращает верхнюю треугольную часть переданной ей матрицы `A`, лежащую выше `k`-ой диагонали. Команды `triu(A, -1)`, `triu(A, int(-1.5))` и `triu(A, -1.5)` возвращают один и тот же результат, в то время как инструкция `triu(A,floor(-1.5))` эквивалентна `triu(A,-2)`.

Такое поведение может показаться странным, но способствует единообразию языка Scilab. Действительно, при обращении к элементам матрицы с использованием переменных в качестве индексов, типом переменной чаще всего будет являться число с плавающей точкой. В этом случае преобразование значения происходит посредством функции `int`. Сказанное иллюстрирует следующий фрагмент:

```
-->i = 2
 i  =
    2.
-->j = 1
 j  =
    1.
-->A(i, j)
 ans  =
  - 6.
```

Обратите внимание, что в этом примере переменные `i` и `j` имеют тип с плавающей точкой.

[1] С точки зрения внутреннего устройства Scilab это объясняется использованием функции `int` на уровне вызовов C или Fortran внутри шлюзов Scilab для преобразования чисел с плавающей точкой в целые.

Иногда использование чисел с плавающей точкой в качестве индексов может приводить к неожиданным результатам, например:

```
-->ones(1, 1)
 ans  =
    1.
-->ones(1, (1 - 0.9) * 10)
 ans  =
     []
```

При отсутствии ошибок округления, обусловленных конечной разрядностью представления чисел в памяти компьютера, значение выражения $(1 - 0.9) * 10$ равнялось бы 1, и в приведенном примере мы получили бы матрицу 1×1. Однако в действительности команда `ones(1,(1-0.9)*10)` возвращает пустую матрицу, так как в результате округления числа $(1 - 0.9) * 10$ с использованием функции `int` получаем 0:

```
-->int((1 - 0.9) * 10)
 ans  =
    0.
```

Причиной тому являются особенности двоичного представления дробных чисел. В результате двоичное представление числа $1 - 0.9$ несколько меньше 0.1, а будучи умноженным на 10, дает число, которое меньше 1, в чем позволяет убедиться следующий фрагмент:

```
-->format(25)
-->1 - 0.9
 ans  =
    0.0999999999999999777955
-->(1 - 0.9) * 10
 ans  =
    0.9999999999999997779554
```

На практике при вычислении индексов не следует применять числа, которые не имеют точного представления в форме с плавающей точкой. Точные результаты дают обычные арифметические операции с целыми числами в пределах диапазона $[-2^{52}, 2^{52}]$.

4.16 Еще об элементарных функциях

В данном разделе рассматриваются тригонометрические функции, принимающие входные значения в градусах, логарифмические функции, а также специальные матричные операции.

Тригонометрические функции, такие как `sin` и `cos`, принимают на вход значения в радианах. Их аналоги, предназначенные для работы с аргументами, выраженными в градусах, отличаются наличием буквы "`d`" в конце имени (например, `sind`, `cosd`). Математически пары таких функций связаны соотношениями вида $\sin d\, x = \sin \frac{x\pi}{180}$. Однако ввиду особенностей компьютерной арифметики результаты выполнения функций для математически эквивалентных аргументов могут различаться. Например, следующий фрагмент демонстрирует вычисление значений $\sin \pi$ и $\sin 180^\circ$:

```
-->sin(%pi)
 ans  =
    1.225D-16
-->sind(180)
 ans  =
    0.
```

Тот факт, что вычисленное значение $\sin \pi$ не равно в точности 0, связан с ограниченной точностью представления действительных чисел в памяти компьютера. Действительно, количество разрядов в двоичном представлении числа π в памяти компьютера ограничено. Следовательно, и результат вычисления функции оказывается приближенным. В то же время число 180 представляется точно, поскольку является целым[1]. Поэтому значение `sind(180)` вычисляется точно.

Функция `log` вычисляет натуральный логарифм числа - функцию, обратную экспоненте `exp`$= e^x$, где e называется основанием натурального логарифма или константой Эйлера. Для вычисления логарифма по основаниям 10 и 2 можно использовать функции `log10` и `log2` соответственно. В следующем примере мы вычислим значения `log`, `log10` и `log2` для набора заданных значений `x`:

```
-->x = [exp(1) exp(2) 1 10 2^1 2^10]
 x  =
    2.7182818    7.3890561    1.    10.    2.    1024.
-->[x' log(x') log10(x') log2(x')]
 ans  =
    2.7182818    1.           0.4342945    1.442695
    7.3890561    2.           0.8685890    2.8853901
    1.           0.           0.           0.
    10.          2.3025851    1.           3.3219281
    2.           0.6931472    0.30103      1.
    1024.        6.9314718    3.0103       10.
```

В первом столбце отображены исходные значения x, второй столбец содержит значения `log(x)`, третий и четвертый, соответственно, значения `log10(x)` и `log2(x)`.

Большинство функций являются поэлементными, т.е., получив на вход матрицу, воздействуют на каждый ее элемент независимо. Вместе с тем, некоторые функции имеют специальное значение в линейной алгебре. Так, например, матричная экспонента определена как $e^X = \sum_{k=0,\infty} \frac{1}{k!} X^k$. Для расчета матричной экспоненты в Scilab применяется функция `expm`. Вполне очевидно, что применение обычной функции `exp` дает совершенно иной результат. Имена функций, имеющих специальное значение применительно к матрицам, в Scilab оканчиваются буквой "`m`", например, `expm`, `sinm` и др. Следующий фрагмент демонстрирует воздействие функций `sin` и `sinm` на матрицу `A` размером 2×2, содержащую кратные $\pi/2$ значения:

```
-->A = [%pi/2 %pi; 2*%pi 3*%pi/2]
 A  =
    1.5707963    3.1415927
    6.2831853    4.712389
-->sin(A)
```

[1] И лежит в диапазоне $[-2^{52}, 2^{52}]$ - см. раздел 3.9.

`chol`	разложение Холецкого
`companion`	сопровождающая матрица
`cond`	число обусловленности
`det`	определитель матрицы
`inv`	обратная матрица
`linsolve`	решение систем линейных уравнений
`lsq`	метод наименьших квадратов
`lu`	LU-разложение с выбором опорного элемента
`qr`	QR-разложение
`rcond`	обратное число обусловленности
`spec`	собственные значения и векторы
`svd`	разложение по сингулярным числам матрицы
`testmatrix`	генерация специальных матриц (Гильберта, Франка и др.)
`trace`	след матрицы

Таблица 18. Некоторые функции линейной алгебры.

```
 ans  =
    1.            1.225D-16
  - 2.449D-16   - 1.
-->sinm(A)
 ans  =
  - 0.3333333     0.6666667
    1.3333333     0.3333333
```

4.17 Высшая алгебра и другие возможности Scilab

Scilab предлагает широкий набор функций для выполнения всех распространенных операций линейной алгебры, позволяющих оперировать как разреженными, так и плотными матрицами. Описание всех доступных алгоритмов потребовало бы отдельной книги, поэтому ограничимся перечислением наиболее часто используемых функций в табл. 18.

4.18 Упражнения

Упражнение 4.1 (*Плюс 1*) С помощью одной инструкции получите вектор $(x_1 + 1, x_2 + 1, x_3 + 1, x_4 + 1)$, если x равен

```
x = 1 : 4;
```

Упражнение 4.2 (*Векторизованное умножение*) Получите вектор $(x_1y_1, x_2y_2, x_3y_3, x_4y_4)$, если x и y равны

```
x = 1 : 4;
y = 5 : 8;
```

Упражнение 4.3 (*Вектор обратных величин*) Получите вектор $\left(\frac{1}{x_1}, \frac{1}{x_2}, \frac{1}{x_3}, \frac{1}{x_4}\right)$, если x равен

```
x = 1 : 4;
```

Упражнение 4.4 (*Векторизованное деление*) Получите вектор $\left(\frac{x_1}{y_1}, \frac{x_2}{y_2}, \frac{x_3}{y_3}, \frac{x_4}{y_4}\right)$, если x и y равны

```
x = 12 * (6 : 9);
y = 1 : 4;
```

Упражнение 4.5 (*Векторизованное возведение в степень*) Получите вектор $(x_1^2, x_2^2, x_3^2, x_4^2)$, если $x = 1, 2, 3, 4$.

Упражнение 4.6 (*Применение функции к вектору*) Получите вектор $(sin(x_1), sin(x_2), \ldots, sin(x_{10}))$ для 10 значений, равномерно распределенных на отрезке $[0, \pi]$.

Упражнение 4.7 (*Векторизованные функции*) Вычислите значения $y = f(x)$ функции f, определяемой уравнением

$$f(x) = \log_{10}(r/10^x + 10^x) \tag{1}$$

если $r = 2.220.10^{-16}$, для 100 равноотстоящих значений из отрезка $[-16, 0]$.

5 Операторы ветвления и цикла

В этом разделе рассматриваются конструкции структурного программирования, доступные в Scilab, такие как условный оператор `if`, оператор выбора `select`, циклы `for` и `while`, а также инструкции `break` и `continue`, предназначенные для управления выполнением цикла.

5.1 Оператор `if`

Оператор `if` позволяет выполнить некоторый блок инструкций в случае истинности заданного условия. В качестве условия может выступать переменная логического типа или любое выражение, результатом вычисления которого является логическое значение. Блок выполняемых инструкций завершается ключевым словом `end`. В следующем примере мы отображаем строку "Привет!", если условие `%t` выполняется. Выражение `%t`, как мы знаем, представляет собой константу, обозначающую *истину*, поэтому данное условие всегда истинно и строка будет отображена в любом случае.

```
if ( %t ) then
  disp("Привет!")
end
```

В результате выполнения этого примера в консоль будет выведено

```
   Привет
 !
```

Если условие ложно, выполняется ветвь `else`, как показано в следующем фрагменте:

```
if ( %f ) then
  disp("Привет!")
else
  disp("До встречи!")
end
```

В данном случае в консоли появится строка

```
 Довстречи
!
```

В качестве условия может использоваться любое выражение, результатом вычисления которого является логическое значение, например, содержащее операторы сравнения "==", ">" и т.д. или функции, возвращающие логическое значение. В следующем примере мы используем оператор "==" для проверки условия и выводим сообщение "Привет!", если условие истинно, и "До встречи!", если оно ложно:

```
i = 2
if ( i == 2 ) then
  disp("Привет!")
else
  disp("До встречи!")
end
```

Подчеркнем, что для проверки на равенство используется оператор "==", а не "=". Использование последнего приводит к предупреждению, как показано в следующем примере:

```
-->i = 2
 i  =
    2.
-->if ( i = 2 ) then
Warning: obsolete use of '=' instead of '=='.
         !
-->  disp("Привет!")

 Hello !
-->else
-->  disp("До встречи!")
-->end
```

При наличии нескольких условий, которые должны быть проверены последовательно, полезной оказывается конструкция `elseif`. Следующий фрагмент демонстрирует, каким образом конструкция `elseif` может применяться для обработки различных значений переменной `i`:

```
i = 2
if ( i == 1 ) then
  disp("Привет!")
elseif ( i == 2 ) then
  disp("До встречи!")
elseif ( i == 3 ) then
  disp("Чао!")
else
  disp("Оревуар!")
end
```

Количество блоков `elseif` не ограничено, что позволяет создавать ветвление произвольной сложности. Однако большое число следующих друг за другом блоков `elseif` часто говорит о необходимости использования оператора `select`, речь о котором пойдет в следующем разделе.

5.2 Оператор `select`

Оператор `select` предназначен для сокращенной записи нескольких последовательных проверок переменной на равенство одному из ряда значений, которые в противном случае необходимо было бы оформить как блоки `elseif`. В зависимости от значения переменной оператор `select` выполняет один из блоков `case`. Количество таких блоков не ограничено.

Ниже показано, как отобразить одну из нескольких возможных строк в соответствии со значением переменной `i`:

```
i = 2
select i
case 1
  disp("Один")
case 2
  disp("Два")
case 3
  disp("Три")
else
  disp("Иное значение")
end
```

В результате, как и следовало ожидать, в консоли будет отображена строка "Два".

Блок `else` выполняется в том случае, если значение переменной не соответствует ни одному из перечисленных вариантов.

Наличие блока `else` не является обязательным, но считается хорошим тоном. Действительно, даже если программист полагает, что соответствующее этому блоку событие в нормальных условиях никогда не может произойти, ошибка в логике выполнения программы может привести к непредсказуемым последствиям. При отсутствии проверки скрипт продолжит выполняться и в наихудшем случае завершится без сообщений об ошибках, вернув неверный результат. Отладка такого скрипта представляет крайне сложную задачу, поскольку неясно, какая инструкция повлекла нарушение работы. Блок `else` призван воспрепятствовать распространению ошибки и может помочь в том, чтобы точнее определить место ее возникновения.

Таким образом, блок `else` должен присутствовать в большинстве конструкций `select`. Для обработки непредвиденных ситуаций в составе блока `else` часто применяется функция `error`. Функция `error` отображает сообщение об ошибке, содержащее указанный в качестве параметра текст. При этом выполнение алгоритма прерывается, интерпретатор Scilab покидает все вызванные функции и возвращает управление консоли.

Модифицируем предыдущий пример. Теперь в случае отрицательного значения переменной `i` будем выводить сообщение об ошибке:

```
i = -5;
select i
case 1
  disp("Один")
case 2
  disp("Два")
case 3
```

```
  disp("Три")
else
  error ( "Непредусмотренное значениепараметра  i" )
end
```

В результате будет отображено:

```
-->i = -5;
-->select i
-->case 1
-->  disp("Один")
-->case 2
-->  disp("Два")
-->case 3
-->  disp("Три")
-->else
-->  error ( "Непредусмотренное значениепараметра  i" )

Непредусмотренное значение параметра i
```

Чаще всего, когда конструкция `select` не содержит блока `else`, следует задаться вопросом, является ли такая ситуация результатом тщательного анализа либо, наоборот, банального упущения. Предположение о том, что блок `else` никогда не будет выполнен, в большинстве случаев не оправдывается.

5.3 Оператор `for`

Оператор `for` применяется для повторения некоторого действия заданное число раз. Чаще всего используется числовой счетчик, пробегающий ряд значений. В конце этого раздела мы увидим, что цикл `for` является гораздо более универсальным и позволяет проводить итерации по элементам матрицы произвольного типа.

Следующий фрагмент печатает значения `i` от 1 до 5:

```
for i = 1 : 5
  disp(i)
end
```

В результате в консоль будет выведено:

```
 1.
 2.
 3.
 4.
 5.
```

Использование оператора ":" для генерации вектора `[1 2 3 4 5]` было рассмотрено в разделе 4.6. Запись `1:5` в приведенном примере создает матрицу, содержащую пять вещественных значений:

```
-->i = 1 : 5
 i  =
    1.    2.    3.    4.    5.
```

Стоит еще раз подчеркнуть, что в данном случае матрица `1:5` содержит *вещественные* значения. Поэтому переменная-счетчик `i` также является вещественной. К этому вопросу мы вернемся позже в данном разделе, рассматривая общую форму записи цикла `for`.

Используя более общую форму записи оператора ":", можно вывести только нечетные числа в интервале от 1 до 5. Для этого, очевидно, в качестве шага необходимо задать значение 2:

```
for i = 1 : 2 : 5
  disp(i)
end
```

В результате выполнения этого алгоритма в консоли будут отображены строки:

```
 1.
 3.
 5.
```

Оператор ":" можно также использовать для перебора значений счетчика в порядке убывания. Следующий фрагмент отображает числа от 5 до 1 в порядке убывания:

```
for i = 5 : -1 : 1
  disp(i)
end
```

В консоль, как и следовало ожидать, будет выведено:

```
 5.
 4.
 3.
 2.
 1.
```

Действительно, инструкция `5:-1:1` возвращает вектор значений от 5 до 1 в порядке убывания:

```
-->i = 5 : -1 : 1
 i  =
    5.    4.    3.    2.    1.
```

Как уже говорилось, цикл `for` является гораздо более универсальным, позволяя перебирать значения различных типов, в том числе матриц и списков. В качестве элементов матриц могут выступать вещественные и целые числа, строки и полиномы.

В следующем примере цикл `for` используется для перебора элементов вектора-строки, содержащего вещественные значения $(1.5, e, \pi)$:

```
v = [1.5 exp(1) %pi];
for x = v
  disp(x)
end
```

В результате в консоли будет отображено:

```
 1.5
 2.7182818
 3.1415927
```

Хотелось бы еще раз обратить внимание читателей на нежелательность использования циклов в Scilab. Всякий раз, когда возникает такое намерение, следует выяснить, не существует ли подходящей векторизованной функции, выполняющей требуемые действия. Различие в скорости выполнения пользовательских циклов и встроенных функций может достигать 10-100 раз, поэтому при наличии такой функции предпочтение должно быть отдано ей. Цикл `for` следует использовать лишь в тех редких случаях, когда подходящей векторизованной функции найти не удается.

5.4 Оператор `while`

Оператор `while` предназначен для повторения некоторого блока инструкций до тех пор, пока условие цикла остается истинным. Проверка условия выполняется перед каждой (в том числе первой) итерацией. В определенный момент условие повторения обращается в *ложь* и цикл завершается. Заметим, что для корректного завершения цикла необходимо, чтобы в теле цикла каким-либо образом изменялись переменные, входящие в условие продолжения, так чтобы в какой-то момент значение этого выражения изменилось бы с истинного на ложное[1].

Следующий фрагмент демонстрирует применение цикла `while` для подсчета суммы чисел от 1 до 10:

```
s = 0
i = 1
while ( i <= 10 )
  s = s + i
  i = i + 1
end
```

Значения переменных по завершении выполнения этого фрагмента равны:

```
 s  =
    55.
 i  =
    11.
```

Заметим, что приведенный пример служит исключительно для иллюстрации работы цикла `while`. На практике, если потребуется вычислить сумму чисел от 1 до 10, следует воспользоваться функцией `sum`:

```
-->sum(1 : 10)
 ans  =
    55.
```

Все сказанное в предыдущем разделе о недостатках цикла `for` в равной степени относится и к `while`. Поэтому векторизованные вычисления также предпочтительны по сравнению с написанием собственных алгоритмов, использующих `while`.

[1] Исключение составляют ситуации с использованием инструкции `break`, речь о которой пойдет в разделе 5.5.

5.5 Инструкции `break` и `continue`

Инструкция `break` позволяет прервать выполнение цикла. Обычно она применяется для выхода из цикла при достижении определенного условия, делающего его продолжение бессмысленным.

Следующий фрагмент демонстрирует использование инструкции `break` для вычисления суммы чисел от 1 до 10. При достижении переменной `i` значения, превышающего 10, цикл завершается:

```
s = 0
i = 1
while ( %t )
  if ( i > 10 ) then
    break
  end
  s = s + i
  i = i + 1
end
```

После выполнения данного алгоритма значения переменных `s` и `i` равны:

```
 s  =
    55.
 i  =
    11.
```

Инструкция `continue` позволяет немедленно перейти к выполнению следующей итерации, пропустив команды, следующие после `continue` в теле цикла. Встретив команду `continue`, интерпретатор Scilab переходит к заголовку цикла, проверяет условие продолжения, и, если оно истинно, делает следующую итерацию.

Следующий пример демонстрирует вычисление суммы $s = 1+3+5+7+9 = 25$. Используемая здесь функция `modulo(i,2)` возвращает 0 при условии, что `i` четно. В данном случае скрипт наращивает значение `i` и использует инструкцию `continue` для перехода к следующей итерации:

```
s = 0
i = 0
while ( i < 10 )
  i = i + 1
  if ( modulo ( i , 2 ) == 0 ) then
    continue
  end
  s = s + i
end
```

Значения переменных `s` и `i` после выполнения данного скрипта равны

```
 s  =
    25.
 i  =
    11.
```

Тот же результат можно получить, используя единственную команду `sum` в сочетании с оператором ":", что является примером векторизованных вычислений в Scilab:

```
s = sum(1 : 2 : 10);
```

Использование высокоуровневой функции (в данном случае `sum`) имеет ряд преимуществ перед эквивалентным ей с точки зрения результата циклом на основе `while`:

1. Высокоуровневая запись короче, а значит проще для понимания человеком.

2. Для матриц значительной размерности высокоуровневые операции выполняются намного быстрее, чем алгоритмы на основе циклов.

Поэтому необходимо внимательно изучить доступные в Scilab функции перед тем, как писать собственный алгоритм с использованием `while`.

6 Функции

Данный раздел посвящен функциям в Scilab. Здесь мы рассмотрим вопросы определения собственных функций и их загрузки в Scilab, научимся создавать *библиотеки*, представляющие собой наборы функций. Поскольку большая часть возможностей Scilab реализована в виде функций, мы увидим, как исследовать свойства той или иной функции. Также будет рассмотрена работа со входными и выходными аргументами. Наконец, мы обсудим возможности отладки функций с использованием инструкции `pause`.

6.1 Обзор

Выделение последовательности инструкций в отдельную функцию, пригодную для повторного использования, является одной из наиболее распространенных задач при работе со Scilab.

Простейший синтаксис вызова функции выглядит следующим образом:

```
outvar = myfunction ( invar )
```

Значение каждого из трех элементов вызова функции приведено в списке:

- `myfunction` представляет собой наименование вызываемой функции,
- `invar` обозначает входные аргументы,
- `outvar` соответствует выходным аргументам.

Значения переменных, указанных при вызове в качестве фактических параметров, функция изменить не может.

Мы уже имели дело со многими функциями на протяжении данного руководства. Например, функция `sin` в составе команды `y=sin(x)` принимает входной аргумент `x` и помещает результат вычисления в переменную `y`. В соответствии с терминологией Scilab входные аргументы называются *правосторонними*, а выходные - *левосторонними*.

Количество входных и выходных аргументов функции не ограничено. Синтаксис вызова функции с фиксированным числом аргументов таков:

```
[o1, ..., on] = myfunction ( i1, ..., in )
```

Список входных аргументов ограничивается *круглыми* скобками, а выходных - *квадратными*. Названия переменных в списках отделяются друг от друга запятыми ”,”.

Следующий фрагмент демонстрирует выполнение LU-разложения матрицы Гильберта. Для начала матрица генерируется с использованием функции `testmatrix`, принимающей два входных аргумента: тип матрицы и ее порядок. Созданную матрицу мы передаем функции `lu`, которая возвращает две или три матрицы в зависимости от заданного пользователем количества выходных аргументов. Если аргументов три, в качестве последнего возвращается матрица перестановок `P`:

```
-->A = testmatrix("hilb", 2)
 A  =
    4.   - 6.
  - 6.     12.
-->[L, U] = lu(A)
 U  =
  - 6.     12.
    0.     2.
 L  =
  - 0.6666667     1.
    1.            0.
-->[L, U, P] = lu(A)
 P  =
    0.     1.
    1.     0.
 U  =
  - 6.     12.
    0.     2.
 L  =
    1.            0.
  - 0.6666667     1.
```

Как мы могли убедиться, поведение функции `lu` зависит от числа выходных аргументов: во втором случае строки матрицы `L` меняются местами. Говоря точнее, при двух выходных аргументах выполняется разложение $A = LU$ (команда `A-L*U` позволяет проверить корректность результата), а при трех - разложение $PA = LU$ с матрицей перестановок P (в чем можно убедиться, выполнив команду `P*A-L*U`). Таким образом, функция `lu` выбирает соответствующий алгоритм в зависимости от количества переданных ей параметров. Scilab предоставляет также возможность определения функций с переменным числом аргументов, однако этот вопрос выходит за рамки данного руководства.

Инструкции Scilab, предназначенные для работы с функциями, приведены в табл. 19. В последующих разделах наиболее часто используемые из них будут рассмотрены подробно.

6.2 Создание собственной функции

Для определения новой функции используются ключевые слова `function` и `endfunction`. В следующем примере мы создаем функцию `myfunction`, которая

`function`	открывает определение функции
`endfunction`	завершает определение функции
`argn`	количество входных и выходных аргументов в данном вызове функции
`varargin`	вектор, представляющий переменное число входных аргументов функции
`varargout`	вектор, представляющий переменное число выходных аргументов функции
`fun2string`	генерирует текстовое определение (исходный код) функции
`get_function_path`	возвращает путь к файлу исходного кода функции
`getd`	отображает полный список функций, определения которых хранятся в заданном каталоге файловой системы
`head_comments`	отображает комментарии к функции
`listfunctions`	отображает свойства функций, которые вызывались ранее в данной сессии
`macrovar`	возвращает списки входных и выходных параметров функции, а также используемых в теле функции внешних переменных, вызовов других функций и локальных переменных

Таблица 19. Ключевые слова и инструкции Scilab, использующиеся при работе с функциями.

принимает единственный параметр `x`, умножает его значение на 2 и возвращает результат в качестве выходного параметра `y`:

```
function y = myfunction ( x )
  y = 2 * x
endfunction
```

Строка `function y = myfunction ( x )` представляет собой *заголовок* функции, в то время как *тело* функции содержит единственную инструкцию `y = 2 * x`. В общем случае тело функции может включать произвольное число команд.

В Scilab существует, по меньшей мере, три способа определить такую функцию:

- Во-первых, можно ввести тело функции непосредственно в консоли Scilab, инструкция за инструкцией. Встретив запись наподобие `function y = myfunction ( x )`, интерпретатор переходит в режим ожидания тела функции. Завершается ввод командой `endfunction`, после чего Scilab возвращается в обычный режим.

- Более удобным вариантом является определение функции в отдельном файле. Этот способ применяется в большинстве случаев. Для того чтобы загрузить заданную таким образом функцию, можно скопировать содержимое файла в консоль (удобно если определение функции содержит всего несколько строк) либо воспользоваться командой *Загрузить в Scilab* (*Load into Scilab*) в меню Scilab.

- Также для загрузки функции можно использовать команду `exec`. Предположим, что определение функции размещается в файле `C:\myscripts\examples-functions.sce`. Для загрузки применяется команда `exec`, как показано в следующем фрагменте:

  ```
  -->exec("C:\myscripts\examples-functions.sce")
  -->function y = myfunction ( x )
  -->  y = 2 * x
  -->endfunction
  ```

  Функция `exec` предназначена для исполнения инструкций, содержащихся в некотором файле, так, как если бы они вводились непосредственно в консоли Scilab. При этом в консоли отображается каждая строка алгоритма. Если файл содержит большое число команд, отображение каждой из них может оказаться нежелательным. Чтобы этого избежать, после инструкции `exec` необходимо поставить "`;`":

  ```
  -->exec("C:\myscripts\examples-functions.sce" );
  ```

  Того же результата можно добиться, выбрав пункт меню *Выполнить файл в Scilab* (*Execute file into Scilab*).

После того как функция создана, ее можно использовать подобно любой другой команде Scilab:

```
-->exec("C:\myscripts\examples-functions.sce");
-->y = myfunction ( 3 )
 y  =
    6.
```

Заметим, что присвоение выходному аргументу `y` значения (в данном случае `y=2*x`) является обязательным. Для того чтобы убедиться в этом, рассмотрим следующий пример, где значение присваивается переменной `z`, а не выходному параметру `y`:

```
function y = myfunction ( x )
  z = 2 * x
endfunction
```

Попытаемся теперь вызвать эту функцию, передав ей значение 1:

```
-->myfunction ( 1 )
 !--error 4
Undefined variable: y
at line       4 of function myfunction called by :
myfunction ( 1 )
```

Интерпретатор Scilab сообщает нам об ошибке, поскольку переменная `y` не была инициализирована в теле функции.

При решении некоторой задачи часто возникает потребность в определении более чем одной функции. Например, при выполнении оптимизации с использованием функции `optim` нам, во-первых, требуется задать целевую функцию в соответствии с форматом, которого ожидает `optim`, а во-вторых - определить функцию-исполнитель, которая будет вызывать `optim` с требуемыми параметрами. В данном случае необходимы всего две функции, однако на практике для решения задачи может потребоваться и несколько десятков функций. В этом случае будет разумным объединить функции в библиотеку, как показано в следующем разделе.

6.3 Библиотеки функций

Библиотека представляет собой набор функций, написанных на языке Scilab и хранящихся в отдельных файлах.

Если набор функций невелик и не содержит файлов справки или исходных текстов на компилируемых языках (таких как C/C++ или Fortran), объединение их в библиотеку является весьма удачным вариантом, в противном случае следует задуматься о создании *модуля*. Разработка собственного модуля не представляет трудностей, однако требует более детального знакомства с внутренним устройством пакета Scilab. Кроме того, модули также имеют в своей основе библиотеки, поэтому для создания первых требуется понимание работы последних. Тем не менее, построение собственного модуля выходит за рамки настоящего руководства. Во многих практических ситуациях создание библиотеки является достаточным для организации набора функций.

В этом разделе рассматривается создание простой библиотеки функций Scilab, а также способы ее автоматической загрузки при запуске пакета.

`genlib`	создает библиотеку из функций, определения которых расположены в заданном каталоге
`lib`	загружает библиотеку

Таблица 20. Инструкции Scilab для работы с библиотеками функций.

Предположим, что имеется несколько файлов `.sci`, содержащих определения функций на языке Scilab. Последовательность действий в этом случае такова:

1. Создать бинарные версии функций, используя команду `genlib`. Функция `genlib` помимо прочего генерирует индексные файлы.

2. Загрузить библиотеку в Scilab, для чего используется функция `lib`.

Перед тем как приступить к рассмотрению примера, необходимо обозначить основные правила создания библиотек функций в Scilab:

- Файлы, содержащие определения функций, должны иметь расширение `.sci`. Строго говоря, это требование не является обязательным, но помогает при поиске скриптов Scilab на жестком диске компьютера.

- В одном файле `.sci` могут быть определены несколько функций Scilab, однако только первая из них будет доступна извне. Иными словами, только первая функция, определенная в файле, считается общедоступной, в то время как остальные неявно полагаются закрытыми (служебными) функциями.

- Имя файла `.sci` должно совпадать с именем общедоступной функции в этом файле. Например, если имя функции `myfun`, то файл, содержащий ее, должен иметь название `myfun.sci`. Это требование является обязательным, в противном случае функция `genlib` не будет работать корректно.

Инструкции Scilab, используемые при работе с библиотеками функций, представлены в табл. 20.

Теперь перейдем к примеру создания конкретной библиотеки. Предположим, что мы работаем на компьютере под управлением ОС Windows. Пусть в каталоге `samplelib` размещаются два файла:

- `C:/samplelib/function1.sci`:

```
function y = function1 ( x )
  y = 1 * function1_support ( x )
endfunction
function y = function1_support ( x )
  y = 3 * x
endfunction
```

- `C:/samplelib/function2.sci`:

```
function y = function2 ( x )
  y = 2 * x
endfunction
```

Для получения бинарных версий этих функций используем инструкцию `genlib`. Первый аргумент `genlib` представляет название будущей библиотеки, а второй указывает каталог, где размещены файлы функций. Заметим, что в данном случае только функции `function1` и `function2` являются общедоступными, а функция `function1_support` может использоваться только внутри библиотеки, но не вне ее.

```
-->genlib("mylibrary", "C:/samplelib")
-->mylibrary
 mylibrary  =
Functions files location : C:\samplelib\.
 function1           function2
```

Функция `genlib` генерирует и помещает в каталог `C:/samplelib` следующие файлы:

- `function1.bin`: бинарная версия файла `function1.sci`,
- `function2.bin`: бинарная версия файла `function2.sci`,
- `lib`: бинарная версия библиотеки,
- `names`: текстовый файл, содержащий имена всех функций в библиотеке.

Полученные файлы `*.bin` и файл `lib` являются кроссплатформенными в том смысле, что могут без изменений использоваться версиями Scilab для Windows, Linux и Mac OS.

Сразу же после вызова `genlib`, две новые функции становятся доступны окружению Scilab и могут быть вызваны, как показано ниже:

```
-->function1(3)
 ans  =
    9.
-->function2(3)
 ans  =
    6.
```

Конечно, каждый раз генерировать библиотеку заново не требуется. Готовую библиотеку можно загрузить посредством команды `lib`, единственный аргумент которой указывает местоположение загружаемой библиотеки в файловой системе. Следующий фрагмент иллюстрирует загрузку созданной нами библиотеки:

```
-->mylibrary = lib("C:\samplelib\")
 ans  =
Functions files location : C:\samplelib\.
 function1           function2
```

При большом числе загружаемых библиотек, удобно поместить вызовы `lib` в стартовый скрипт Scilab, который автоматически исполняется при запуске

пакета. В этом случае все указанные библиотеки будут доступны сразу же после запуска Scilab. Файл стартового скрипта размещается в основном каталоге Scilab, путь к которому можно узнать, проверив значение переменной SCIHOME:

```
-->SCIHOME
 SCIHOME  =
 C:\Users\username\AppData\Roaming\Scilab\scilab-5.2.0
```

Стартовый скрипт носит имя .scilab и является обычным скриптом Scilab, в том числе может содержать комментарии. Для загрузки созданной ранее библиотеки добавим в файл .scilab следующие строки:

```
// Load my favorite library.
mylibrary = lib("C:/samplelib/")
```

В результате библиотека mylibrary будет загружаться всякий раз при запуске Scilab.

6.4 Управление выходными переменными

Каждая функция Scilab может иметь один или несколько входных и выходных аргументов. В простейшем случае число входных и выходных аргументов фиксировано, поэтому использовать такую функцию не составляет труда. Однако, как мы увидим далее, даже простейшие случаи допускают вариации.

Предположим, что функция simplef определена с двумя входными и двумя выходными аргументами:

```
function [y1 , y2] = simplef ( x1, x2 )
  y1 = 2 * x1
  y2 = 3 * x2
endfunction
```

При вызове функции можно указать два, один либо ни одного выходного аргумента. Если не указано ни одного выходного аргумента, значение, которое должно было быть присвоено первому из них, помещается в переменную ans. Можно указать один выходной аргумент, который получит значение y1. Наконец, можно предоставить функции два аргумента, как это предусмотрено ее определением. В следующем примере рассматриваются все три варианта вызова функции simplef с различным числом выходных переменных:

```
-->simplef ( 1 , 2 )
 ans  =
    2.
-->y1 = simplef ( 1 , 2 )
 y1  =
    2.
-->[y1,y2] = simplef ( 1 , 2 )
 y2  =
    6.
 y1  =
    2.
```

Таким образом, даже простейшее определение функции допускает некоторую свободу относительно числа выходных аргументов. Более гибким способом

`whereami`	отображает текущее дерево вызовов
`where`	представляет текущее дерево вызовов в виде матрицы

Таблица 21. Команды Scilab для работы со стеком вызовов.

задания переменного числа входных и выходных аргументов является использование ключевых слов `argn`, `varargin` и `varargout`. Эти возможности не рассматриваются в настоящем руководстве, однако для построения действительно гибких функций следует иметь их в виду.

6.5 Уровни стека вызовов

Как и в других языках программирования, в Scilab вызовы функций могут быть вложенными, т.е. функция `f` может вызывать функцию `g`, а та, в свою очередь, обращаться к функции `h` и т.д. Команды, введенные пользователем в консоли Scilab, соответствуют нулевому уровню стека вызовов. Инструкции, находящиеся в теле запущенной из консоли функции, составляют первый уровень. Каждый вложенный вызов увеличивает глубину стека на 1, так что текущая глубина равна длине цепочки вложенных вызовов. Функции, представленные в табл. 21, используются для получения информации о текущем состоянии стека.

Для иллюстрации мы определим три функции `fmain`, `flevel1` и `flevel2`, вызывающие друг друга, и используем в `flevel2` инструкцию `whereami`, которая отображает текущее состояние стека вызовов:

```
function y = fmain ( x )
  y = 2 * flevel1 ( x )
endfunction
function y = flevel1 ( x )
  y = 2 * flevel2 ( x )
endfunction
function y = flevel2 ( x )
  y = 2 * x
  whereami()
endfunction
```

При вызове функции `fmain` в консоль будет выведено:

```
-->fmain(1)
whereami  called at line 3 of macro flevel2
flevel2   called at line 2 of macro flevel1
flevel1   called at line 2 of macro fmain
 ans  =
    8.
```

Как можно видеть, отображается три уровня стека вызовов, соответствующие вызванным функциям.

Значения уровней стека вызовов в приведенном примере таковы:

- уровень 0 - глобальный (команды, вводимые в консоли Scilab),
- уровень -1 - в теле функции `fmain`,
- уровень -2 - в теле функции `flevel1`,

- уровень -3 - в теле функции `flevel2`.

Эти уровни вызова отображаются в составе приглашения Scilab при интерактивной отладке функций с использованием инструкции `pause` и точек останова.

6.6 Инструкция `return`

Инструкция `return` позволяет завершить выполнение функции и вернуть управление вызывающему коду. Это бывает полезно, если в силу каких-то причин исполнение оставшейся части алгоритма потеряло смысл.

Представленная ниже функция вычисляет сумму целых чисел от `istart` до `iend`. При корректных значениях параметров для этого вызывается функция `sum`, однако если значение переменной `istart` отрицательно либо если условие `istart<=iend` не выполняется, значение переменной `y` устанавливается в 0 и функция завершает выполнение.

```
function y = mysum ( istart , iend )
  if ( istart < 0 ) then
    y = 0
    return
  end
  if ( iend < istart ) then
    y = 0
    return
  end
  y = sum ( istart : iend )
endfunction
```

Следующий фрагмент позволяет убедиться, что функция `mysum` работает именно так, как задумано:

```
-->mysum ( 1 , 5 )
 ans  =
    15.
-->mysum ( -1 , 5 )
 ans  =
    0.
-->mysum ( 2 , 1 )
 ans  =
    0.
```

Присутствие в теле одной функции многочисленных инструкций `return` считается плохим стилем программирования, поскольку серьезно усложняет анализ исходного кода такой функции. В этой связи рекомендуется ограничиться использованием `return` лишь в тех ситуациях, где в противном случае потребовалось бы написание большого объема дополнительного кода. Как правило же, функция должна возвращать управление при достижении последней строки.

6.7 Отладка функций. Инструкция `pause`

В данном разделе мы рассмотрим основной метод отладки, позволяющий устранить большую часть простых ошибок. Этот метод основан на использовании инструкций `pause`, `resume` и `abort`, назначение которых раскрывает табл. 22.

`pause`	приостановить выполнение функции и ожидать команд пользователя
`resume`	продолжить выполнение функции
`abort`	завершить выполнение функции

Таблица 22. Отладочные инструкции Scilab.

Работа с пакетом Scilab обычно состоит в определении функций, реализующих алгоритмы для решения поставленной задачи. Однако работоспособность той или иной функции нередко оказывается под вопросом ввиду наличия синтаксических ошибок в ее коде.

Пусть требуется вычислить сумму целых чисел от `istart` до `iend`. Для этого определим функцию `mysum`, текст которой (приведен ниже) содержит ошибки. В частности, второй аргумент, `"foo"`, передаваемый функции `sum`, избыточен.

```
function y = mysum ( istart , iend )
  y = sum ( iend : istart , "foo" )
endfunction
```

Следующий фрагмент отражает результат выполнения функции `mysum`:

```
-->mysum ( 1 , 10 )
 !--error 44
Wrong argument 2.
at line       2 of function mysum called by :
mysum ( 1 , 10 )
```

Для того чтобы обнаружить проблему, поместим инструкцию `pause` в тело функции `mysum`:

```
function y = mysum ( istart , iend )
  pause
  y = sum ( iend : istart , "foo" )
endfunction
```

Снова обратимся к функции `mysum`, передав ей те же самые аргументы:

```
-->mysum ( 1 , 10 )
Type 'resume' or 'abort' to return to standard level prompt.
-1->
```

В данный момент мы находимся *внутри* тела функции `mysum`. Приглашение "`-1->`" указывает, что текущему положению соответствует уровень -1 стека вызовов. Теперь мы можем проверить значения переменных `istart` и `iend`, введя их имена в консоли:

```
-1->istart
 istart  =
    1.
-1->iend
 iend  =
    10.
```

Чтобы увидеть, к какому результату приведет выполнение той или иной инструкции при текущих значениях переменных, скопируем эту инструкцию в консоль:

```
-1->y = sum ( iend : istart , "foo" )
y = sum ( iend : istart , "foo" )
                                    !--error 44
Wrong argument 2.
```

Теперь легко видеть, что источником проблемы является именно введенная строка. Наличие второго аргумента функции `sum` является в данном случае ошибкой, поэтому удалим его:

```
-1->y = sum ( iend : istart )
 y  =
    0.
```

После этой модификации выполнение функции `mysum` больше не приводит к ошибкам, однако результат по-прежнему неверен. Можно заметить, что переменные `istart` и `iend` перепутаны местами. Проверим результат выполнения более корректного, на наш взгляд, вызова и убедимся, что он совпадает с ожидаемым:

```
-1->y = sum ( istart : iend )
 y  =
    55.
```

Для выхода из режима отладки служит инструкция `abort`, которая прерывает всю цепочку вызовов и возвращает управление командной консоли Scilab.

```
-1->abort
-->
```

Вид приглашения "`-->`" указывает, что теперь мы находимся на нулевом (глобальном) уровне стека вызовов.

Исправим определение функции в соответствии с выводами, сделанными в ходе отладки:

```
function y = mysum ( istart , iend )
  pause
  y = sum ( istart : iend )
endfunction
```

Вызовем функцию снова:

```
-->mysum ( 1 , 10 )
Type 'resume' or 'abort' to return to standard level prompt.
-1->
```

Поскольку теперь мы уверены в корректности кода, введем команду `resume`, которая указывает Scilab продолжить исполнение кода:

```
-->mysum ( 1 , 10 )
-1->resume
 ans  =
    55.
```

Теперь, когда ошибки устранены, можно удалить инструкцию `pause` из тела функции:

```
function y = mysum ( istart , iend )
  y = sum ( istart : iend )
endfunction
```

`plot`	двухмерный график
`surf`	трехмерный график
`contour`	контурный график
`pie`	круговая диаграмма
`histplot`	гистограмма
`bar`	столбиковая диаграмма
`barh`	горизонтальная столбиковая диаграмма
`hist3d`	трехмерная гистограмма
`polarplot`	график в полярных координатах
`Matplot`	цветной двухмерный график матрицы
`Sgrayplot`	сглаженный контурный график с использованием цвета
`grayplot`	несглаженный контурный график с использованием цвета

Таблица 23. Функции Scilab для отображения графиков.

В данном разделе мы могли убедиться в эффективности интерактивной отладки функций с помощью команд `pause`, `resume` и `abort`. Приведенный пример, конечно, существенно упрощен и вряд ли требует отладки для обнаружения вполне очевидных ошибок. Тем не менее, использование `pause` оказывается удобным и эффективным способом выявления неполадок и в гораздо более сложных ситуациях.

7 Построение графиков

Отображение графиков и других изображений является распространенной задачей при анализе данных и создании отчетов. Scilab предоставляет широкие возможности для создания и настройки различных типов графиков и диаграмм. В данном разделе мы рассмотрим создание двухмерных и контурных графиков, затем отобразим на графике название и легенду и, наконец, увидим, как сохранить результаты отображения в файл для дальнейшего использования.

7.1 Обзор графических возможностей Scilab

Scilab предоставляет возможности для создания различных типов графиков, среди которых двухмерные, контурные и трехмерные графики, гистограммы, столбиковые и круговые диаграммы и др. Наиболее часто используемые функции для отображения графиков представлены в табл. 23.

Для того чтобы получить пример трехмерного графика, достаточно набрать в консоли Scilab команду `surf()`:

```
-->surf()
```

При создании графиков в данном разделе используются вспомогательные функции, приведенные в табл. 24.

`linspace`	генерирует вектор из заданного числа равноотстоящих значений
`feval`	вычисляет значения функции в точках сетки
`legend`	задает легенду текущего графика
`title`	отображает название на текущем графике
`xtitle`	отображает название и подписи к осям на текущем графике

Таблица 24. Вспомогательные функции, используемые при построении графиков.

7.2 Отображение двухмерных графиков

В этом разделе мы увидим, как отобразить простой двухмерный график функции, уделяя особое внимание возможностям векторизации, позволяющим создать матрицу исходных данных одной командой.

Для начала определим функцию, график которой мы собираемся строить. Наша функция `myquadratic` будет возводить свой аргумент `x` в квадрат, используя оператор "`^`":

```
function f = myquadratic ( x )
  f = x ^ 2
endfunction
```

При помощи функции `linspace` создадим вектор из 50 равноотстоящих значений на отрезке $[1, 10]$:

```
xdata = linspace ( 1 , 10 , 50 );
```

Вектор `xdata` мы передаем функции `myquadratic`, которая рассчитывает значения в каждой точке:

```
ydata = myquadratic ( xdata );
```

Теперь в нашем распоряжении есть вектор-строка `ydata`, содержащий 50 элементов, который наряду с `xdata` мы используем в качестве параметра функции `plot` для отображения графика:

```
plot ( xdata , ydata )
```

Полученный график показан на рис. 10.

Отметим, что тот же график можно было построить без явного вычисления вектора значений `ydata`, передав функции `plot` в качестве второго параметра саму функцию `myfunction`[1]:

```
plot ( xdata , myquadratic )
```

Использование функции в качестве параметра позволяет существенно снизить затраты памяти при построении графика в случае большого числа точек.

7.3 Контурные графики

В этом разделе рассматриваются контурные графики функций двух переменных, для построения которых используется функция `contour`. Контурные гра-

[1] В Scilab функции являются полноценными объектами, поэтому могут использоваться в качестве входных и выходных аргументов других функций.

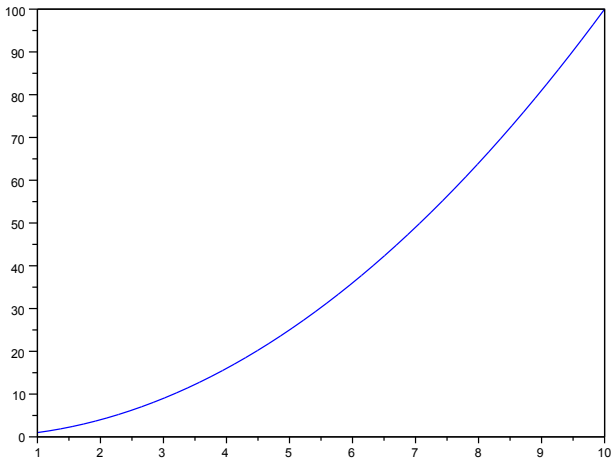

Рис. 10. График функции x^2.

фики находят активное применение при оптимизации функций, позволяя отобразить рельеф функции двух переменных так, что местонахождение оптимума становится очевидным.

Допустим, определена функция f от n переменных $f(\mathbf{x}) = f(x_1, \ldots, x_n)$ и $\mathbf{x} \in \mathbb{R}^n$. Для заданного $\alpha \in \mathbb{R}$ уравнение

$$f(\mathbf{x}) = \alpha \tag{2}$$

определяет поверхность в $(n+1)$-мерном пространстве $\mathbb{R}^{n+1}$.

При $n = 2$ точки $z = f(x_1, x_2)$ образуют поверхность в трехмерном пространстве $(x_1, x_2, z) \in \mathbb{R}^3$, что позволяет отобразить *контурный* график целевой функции. При $n > 3$ столь удобного решения не существует - в этом случае можно выбрать две наиболее значимые переменные и построить график, варьируя только их.

Функция `contour`, позволяющая построить контурный график, имеет следующий синтаксис:

```
contour( x , y , z , nz )
```

где

- `x` и `y` - векторы-строки значений x и y, с числом элементов `n1` и `n2` соответственно;
- `z` - вещественнозначная матрица размером `(n1,n2)`, содержащая значения рассматриваемой функции, либо объект-функция Scilab, определяющая поверхность `z=f(x,y)`,
- `nz` - значения уровней либо их количество.

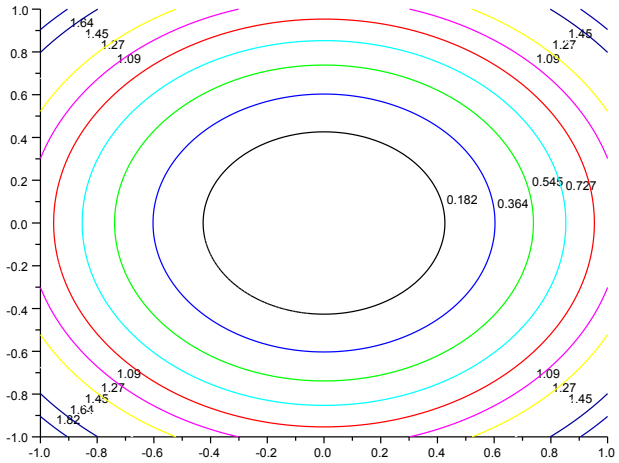

Рис. 11. Контурный график функции $f(x_1, x_2) = x_1^2 + x_2^2$.

В следующем фрагменте мы используем простую форму функции `contour`, которой в качестве параметра передается функция `myquadratic`. Функция `myquadratic` принимает два аргумента `x1` и `x2` и возвращает значение $f(x_1, x_2) = x_1^2 + x_2^2$. Для генерации значений переменных, образующих сетку, используется функция `linspace`:

```
function f = myquadratic2arg ( x1 , x2 )
  f = x1 ** 2 + x2 ** 2;
endfunction
xdata = linspace ( -1 , 1 , 100 );
ydata = linspace ( -1 , 1 , 100 );
contour ( xdata , ydata , myquadratic2arg , 10)
```

Полученный в результате график представлен на рис. 11.

На практике функция, график которой необходимо отобразить, часто принимает единственный аргумент `x`, представляющий собой вектор-строку, в то время как функция `contour` требует наличия двух аргументов. Можно предложить следующие варианты решения данной проблемы:

- определить новую функцию, которая будет вызывать исходную,
- передать функции `contour` массив данных вместо объекта-функции.

Оба этих подхода рассмотрены далее, так что читатель может выбирать наиболее подходящий вариант.

Для начала обратимся ко второму способу, предполагающему генерацию массива значений функции. Пусть функция `myquadratic1arg` принимает на вход вектор из двух элементов. Для вычисления матрицы `zdata`, содержащей значения функции, выполняются два вложенных цикла. Для каждой комбинации $(x(i), y(j)) \in \mathbb{R}^2$ при $i = 1, 2, \ldots, n_x$ и $j = 1, 2, \ldots, n_y$, где n_x и n_y - это

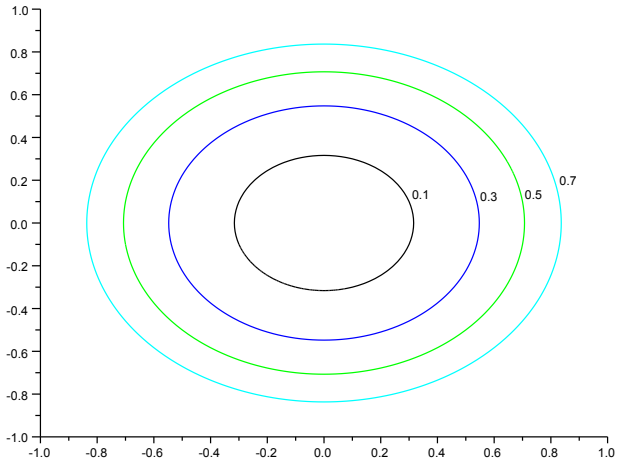

Рис. 12. Контурный график функции $f(x_1, x_2) = x_1^2 + x_2^2$ с явным указанием уровней.

количество точек по осям x и y соответственно, в матрицу `zdata` заносится соответствующее значение. Наконец, для построения графика, мы воспользуемся функцией `contour`, передав ей список уровней (а не их число, как в предыдущем фрагменте). Это позволяет явно задать требуемые уровни вместо того, чтобы предоставлять Scilab их автоматическое вычисление.

```
function f = myquadratic1arg ( x )
  f = x(1) ** 2 + x(2) ** 2;
endfunction
xdata = linspace ( -1 , 1 , 100 );
ydata = linspace ( -1 , 1 , 100 );
// Внимание! Применения двух вложенных циклов следует избегать.
for i = 1 : length(xdata)
    for j = 1 : length(ydata)
      x = [xdata(i) ydata(j)].';
      zdata ( i , j ) = myquadratic1arg ( x );
    end
end
contour ( xdata , ydata , zdata , [0.1 0.3 0.5 0.7])
```

Полученный график показан на рис. 12.

Рассмотренный фрагмент выполняет поставленную задачу, однако работает неэффективно из-за использования циклов. Для повышения скорости выполнения использованию циклов следует предпочесть применение встроенных функций и векторизованных операций, рассмотренных ранее. В частности, для вычисления значений функции на сетке можно использовать функцию `feval`.

Предположим, что модифицировать функцию `myquadratic1arg` невозможно, поэтому определим промежуточную функцию `myquadratic3`, принимающую два входных аргумента и вызывающую `myquadratic1arg`. Теперь, используя встроенную функцию `feval`, можем получить матрицу значений функции

zdata:

```
function f = myquadratic1arg ( x )
  f = x(1) ** 2 + x(2) ** 2;
endfunction
function f = myquadratic3 ( x1 , x2 )
  f = myquadratic1arg ( [x1 x2] )
endfunction
xdata = linspace ( -1 , 1 , 100 );
ydata = linspace ( -1 , 1 , 100 );
zdata = feval ( xdata , ydata , myquadratic3 );
contour ( xdata , ydata , zdata , [0.1 0.3 0.5 0.7])
```

Результатом является тот же контурный график, что и ранее (рис. 12).

Наконец, построить график функции `myquadratic3` также можно, непосредственно передав эту функцию в качестве аргумента `contour`:

```
function f = myquadratic1arg ( x )
  f = x(1) ** 2 + x(2) ** 2;
endfunction
function f = myquadratic3 ( x1 , x2 )
  f = myquadratic1arg ( [x1 x2] )
endfunction
xdata = linspace ( -1 , 1 , 100 );
ydata = linspace ( -1 , 1 , 100 );
contour ( xdata , ydata , myquadratic3 , [0.1 0.3 0.5 0.7])
```

Полученный в результате график, конечно, будет в точности совпадать с предыдущими (рис. 12). Преимуществом же этого способа является экономия памяти, так как в данном случае нет нужды хранить матрицу значений `zdata`.

Таким образом, мы вкратце рассмотрели построение возможности, которые Scilab предоставляет для отображение графиков. В следующем разделе мы обратимся к возможностям настройки таких элементов графика, как заголовок, названия осей и легенда.

7.4 Подписи на графиках

Для придания графику законченного вида необходимо отобразить название графика, подписи осей и легенду.

Вернемся к примеру построения графика функции x^2, рассмотренному в разделе 7.2:

```
function f = myquadratic ( x )
  f = x .^ 2
endfunction
xdata = linspace ( 1 , 10 , 50 );
ydata = myquadratic ( xdata );
plot ( xdata , ydata )
```

Результат выполнения данного фрагмента был показан на рис. 10.

Графические возможности Scilab основаны на использовании *графических дескрипторов*. Графические дескрипторы предоставляют объектно-ориентированный доступ к свойствам графического объекта. Каждая графическая область состоит из примитивов, таких как линии, образующие кривые, оси, название графика, его легенда и т.п. Всякому графическому примитиву соответствует

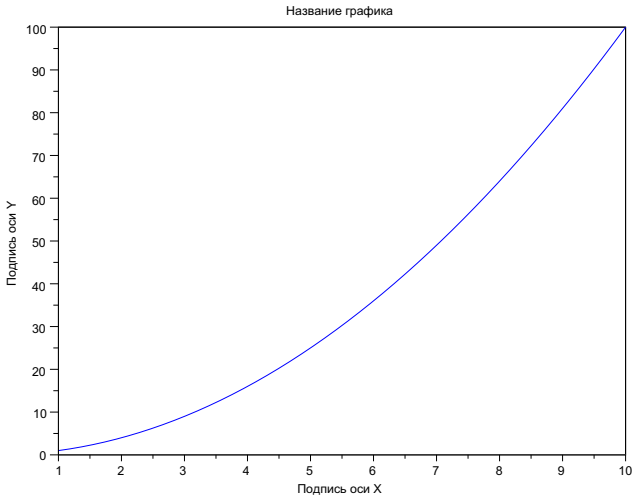

Рис. 13. График функции x^2 с названием и подписями осей.

определенный набор свойств, например, толщина, цвет линий и т.п. Доступ к этим свойствам осуществляется так же, как и к любым другим переменным Scilab. Управление графическими дескрипторами, таким образом, является достаточно гибким и удобным механизмом работы с графическими объектами.

Простейшее оформление графиков может выполняться посредством встроенных функций Scilab без явного обращения к дескрипторам - в данном руководстве мы ограничимся рассмотрением только этих базовых возможностей. Например, функция `title` используется для того, чтобы задать название графика:

```
title ( "Название графика" );
```

Для того чтобы отобразить на графике подписи осей, используем функцию `xtitle`:

```
xtitle ("Название графика", "Подпись оси X", "Подпись оси Y");
```

На рис. 13 представлен результат выполнения этой команды.

Часто возникает необходимость отобразить совместно графики двух функций. Ниже мы определяем функции $f(x) = x^2$ и $f(x) = 2x^2$ и отображаем их на одном графике. Для того чтобы различать кривые, соответствующие каждой из функций, мы используем третий параметр функции `plot`. Значения `"+-"` и `"o-"` определяют способ отображения точек (в данном случае они будут отображаться символами "`+`" и "`o`") и соединяющих линий на графике (сплошная линия).

```
function f = myquadratic ( x )
  f = x ^ 2
endfunction
function f = myquadratic2 ( x )
  f = 2 * x ^ 2
endfunction
```

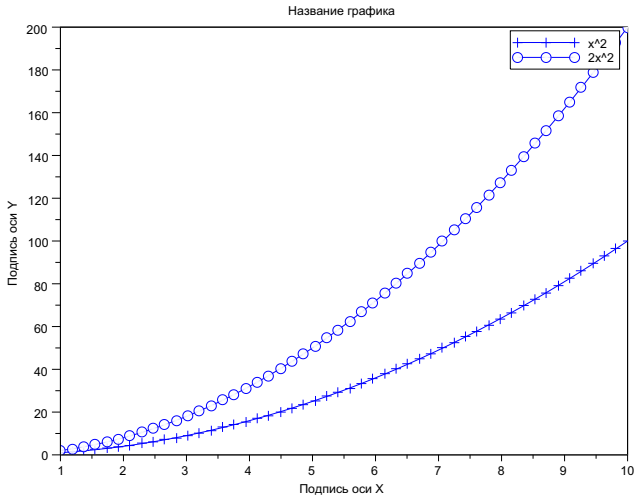

Рис. 14. График двух функций $f_1(x) = x^2$ и $f_2(x) = 2x^2$. Для соотнесения кривых и функций используются стили отображения точек и линий, а также легенда.

```
xdata = linspace ( 1 , 10 , 50 );
ydata = myquadratic ( xdata );
plot ( xdata , ydata , "+-" )
ydata2 = myquadratic2 ( xdata );
plot ( xdata , ydata2 , "o-" )
xtitle ("Название графика", "Подпись оси X", "Подпись оси Y");
```

Помимо этого необходимо отобразить легенду, указав, какая из кривых соответствует каждой из функций, для чего используется инструкция `legend`:

```
legend ( "x^2" , "2x^2" );
```

Рис. 14 демонстрирует получившийся в итоге график.

Теперь мы знаем, как создать график и настроить параметры его отображения. Если график окажется достаточно интересным, его можно сохранить в файл для последующего использования. Экспорту графиков посвящен заключительный раздел данной главы.

7.5 Экспорт изображений

В данном разделе рассматриваются способы вывода изображения в графический файл с использованием оконного меню или встроенных функций Scilab.

Экспортировать выведенное на экран изображение можно, выбрав в меню пункт *Файл* > *Экспортировать...* (*File* > *Export to...*). Появляющееся после этого диалоговое окно предложит указать имя и тип создаваемого файла. Scilab позволяет экспортировать изображения в файлы многих распространенных векторных и растровых форматов. Для того чтобы сохранить изображение, не прибегая к помощи меню, используются функции, указанные в табл. 25.

Векторные	
`xs2png`	экспорт в формат PNG
`xs2pdf`	экспорт в формат PDF
`xs2svg`	экспорт в формат SVG
`xs2eps`	экспорт в формат Encapsulated Postscript
`xs2ps`	экспорт в формат Postscript
`xs2emf`	экспорт в формат EMF (только для Windows)
Растровые	
`xs2fig`	экспорт в формат FIG
`xs2gif`	экспорт в формат GIF
`xs2jpg`	экспорт в формат JPG
`xs2bmp`	экспорт в формат BMP
`xs2ppm`	экспорт в формат PPM

Таблица 25. Функции экспорта изображений.

Все функции, перечисленные в табл. 25, принимают два параметра, первый из которых, `window_number`, представляет собой номер графического окна (отображается в заголовке), а второй, `filename`, задает желаемое имя файла, куда будет записано изображение:

```
xs2png ( window_number , filename )
```

Следующая команда, к примеру, выведет содержимое окна с номером 0 в файл `foo.png`:

```
xs2png ( 0 , "foo.png" )
```

Для получения качественных документов предпочтителен векторный формат изображений. Если предполагается использовать изображения в документах LaTeX, можно остановить свой выбор на формате PDF.

8 Заключение

В данном руководстве был затронут широкий спектр вопросов, что, как мы надеемся, послужит хорошей отправной точкой для более глубокого знакомства с пакетом Scilab. Вместе с тем, ряд вопросов остался за рамками обсуждения. Для их изучения читатель может обратиться к источникам, упомянутым в начале документа.

В разделе 1.3 была отмечена возможность самостоятельной компиляции Scilab из открытого исходного кода. Другой задачей может являться добавление собственных функций непосредственно в Scilab. Для этого была введена модульная организация пакета. Однако часто возникает потребность интегрировать в Scilab готовые функции, реализованные на языках C или Fortran. В этом случае необходимо изучить особенности внутреннего устройства пакета. Раздел 7 "Interfacage" в книге [4] и раздел 2.5, "Interfacing" в книге [3] могут оказаться полезными в этом случае.

Читателей, знающих французский язык, может заинтересовать книга [4], где освещаются темы создания и использования библиотек Scilab, решения диффе-

ренциальных уравнений, применения Scicos и др. Более глубокое представление о Scicos можно получить, прочитав книгу [3] (на английском языке).

Ссылки на литературу для дальнейшего изучения можно также найти на сайте Scilab [5] в разделе документации.

9 Благодарность

Выражаем благодарность Claude Gomez, Vincent Couvert, Allan Cornet и Serge Steer за ценные замечания, а также Julie Paul и Sylvestre Ledru, оказавшим помощь при подготовке данного документа.

10 Ответы к упражнениям

10.1 Ответы к упражнениям раздела 1.7

Ответ к упражнению 1.1 (*Установка Scilab*) Установите текущую версию Scilab на свой компьютер (на момент написания данного руководства текущей является версия Scilab 5.2). Установка Scilab не представляет труда, поскольку производится специальной программой-установщиком, которая выполняет за пользователя большую часть работы по развертыванию и настройке пакета. На рис. 15 представлены шаги мастера установки Scilab v5.2.2 в ОС Windows. □

Ответ к упражнению 1.2 (*Интерактивная справка:* `derivative`) В данном упражнении необходимо различными способами получить справку о функции `derivative`.

- *Способ 1.* Откройте окно справки, выбрав в меню пункт *Помощь* > *Содержание* (*?* > *Help Browser*). В панели слева выберите вкладку "Поиск", обозначенную пиктограммой лупы. В поле поиска введите слово `derivative` и нажмите <Enter>. Справочная система отобразит список всех страниц, содержащих слово *derivative* с указанием частоты появления этого слова на каждой. Первая из перечисленных страниц представляет справочную запись о функции `derivative`, а другие лишь содержат упоминания об этой функции. Выберите первую страницу, нажав на ее название левой клавишей мыши, и рассмотрите описание функции `derivative`.
- *Способ 2.* Для получения справки также можно использовать ресурсы, доступные на сайте Scilab:

  http://www.scilab.org/product/man

  Используя веб-браузер, выполните поиск по слову *derivative*. Страница справки размещается по следующем адресу:

  http://www.scilab.org/product/man/derivative.html

  Ради интереса можете найти тем же способом описание функций `diff`, `bsplin3val`, `derivat` и `dlgamma`.
- *Способ 3.* Наконец, справку о функции можно получить из консоли, набрав команду

```
help derivative
```

  Искомая страница справки представлена на рис. 16.

□

10.2 Ответы к упражнениям раздела 2.6

Ответ к упражнению 2.1 (*Использование консоли*) Введите следующие символы в консоли Scilab:

```
atoms
```

Нажмите клавишу <Tab>. Появившееся окно (рис. 17) отобразит список функций, имена которых начинаются с "atoms". Нажмите клавишу "I" и снова <Tab>. Теперь в окне подсказки (рис. 18) перечислены только функции, имена которых начинаются символами "atomsI". □

Ответ к упражнению 2.2 (*Использование функции* `exec`) Переменная `SCI` содержит путь к каталогу, в который установлен пакет Scilab. Команда `SCI + "/modules"` выводит строку, которая представляет собой конкатенацию этого пути и строки `"/modules"`:

```
-->SCI + "/modules"
 ans  =
 C:/PROGRA~1/SCILAB~1.0-B/modules
```

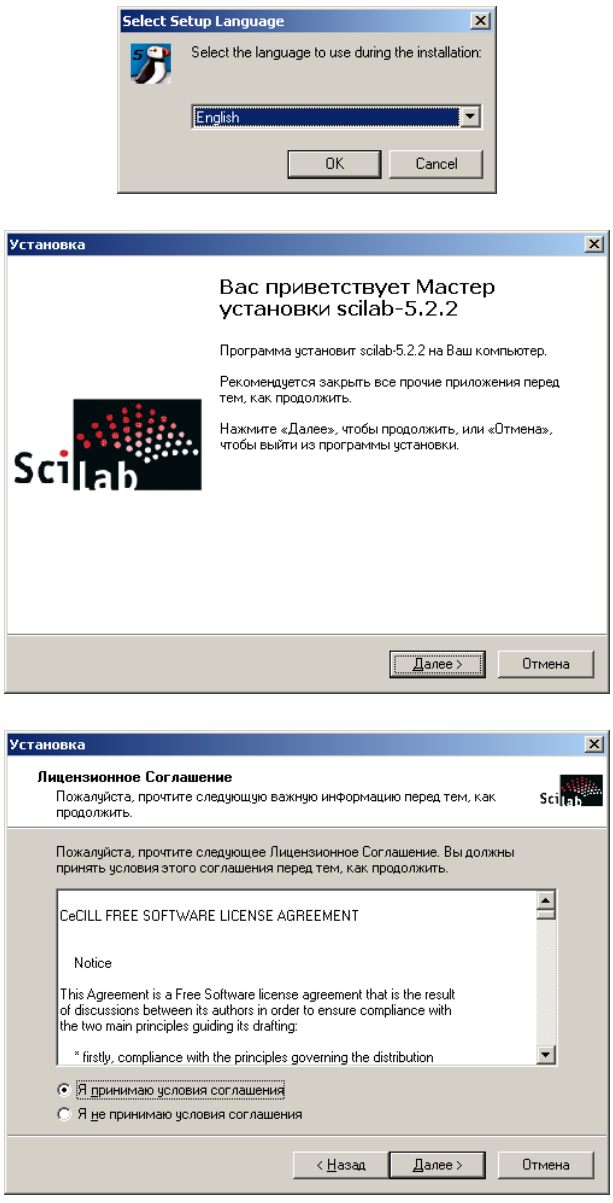

Рис. 15. Шаги мастера установки Scilab v5.2.2 в ОС Windows.

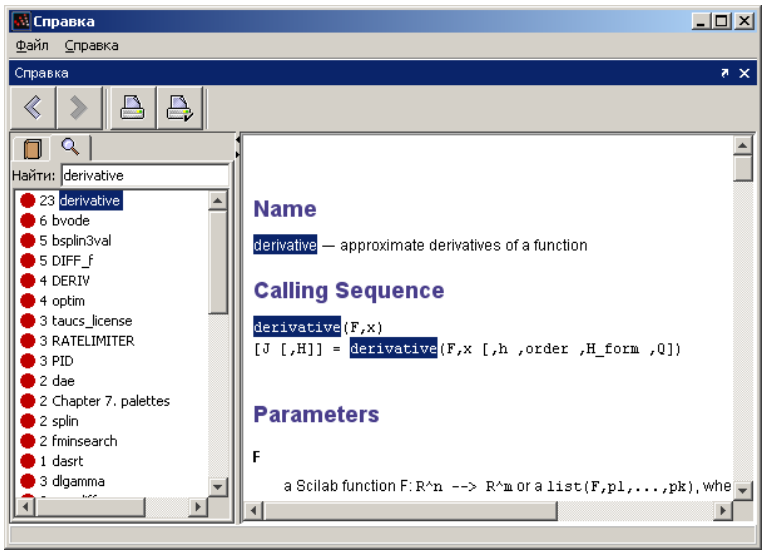

Рис. 16. Страница справки Scilab по функции `derivative`.

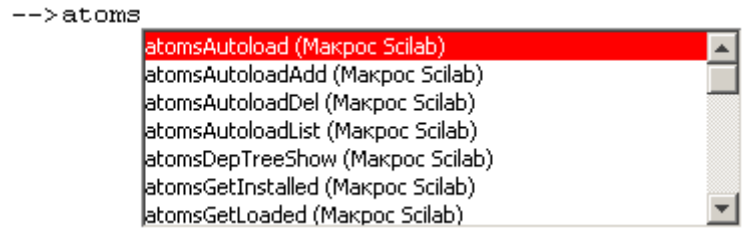

Рис. 17. Использование подсказки для просмотра функций модуля ATOMS.

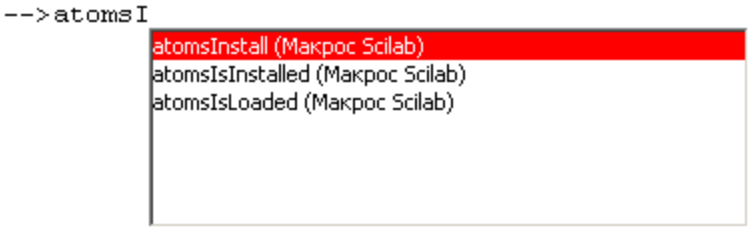

Рис. 18. Ввод каждого следующего символа сужает количество предлагаемых вариантов.

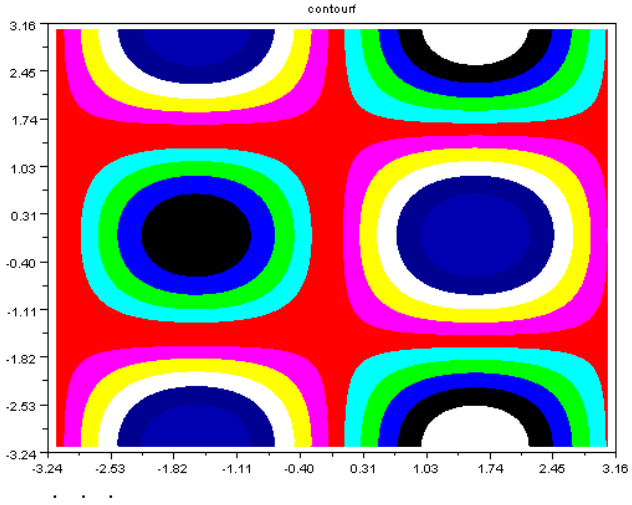

Рис. 19. Результат выполнения демонстрационного скрипта `contourf.dem.sce`.

Теперь с помощью команды `ls(SCI + "/modules")` мы можем просмотреть список файлов в подкаталоге *modules* основного каталога Scilab:

```
-->ls(SCI + "/modules")
 ans  =

!xpad                          !
!                              !
!xcos                          !
!                              !
!windows_tools                 !
!                              !
[...]
```

Исполнение демонстрационного скрипта `contourf.dem.sce` отображает график, представленный на рис. 19.

Различие между инструкциями

```
dname=SCI+"/modules/graphics/demos/2d_3d_plots";
exec(fullfile(dname,"contourf.dem.sce"))
exec(fullfile(dname,"contourf.dem.sce"));
```

состоит в использовании символа ";" после второй из них. При этом Scilab не отображает текст алгоритма в ходе его выполнения, что может быть удобно, если текст содержит большое число строк. □

10.3 Ответы к упражнениям раздела 3.13

Ответ к упражнению 3.1 (*Приоритет операторов*) Порядок выполнения операторов в выражении называется *приоритетом*. Например, выражение $2\times3+4$ эквивалентно выражению $(2\times3)+4$, поскольку операция умножения имеет более высокий приоритет по сравнению со сложением. В следующем примере мы убедимся, что Scilab следует тем же правилам, которые действуют в арифметике:

```
-->2 * 3 + 4
```

```
 ans  =
    10.
-->2 + 3 * 4
 ans  =
    14.
-->2 / 3 + 4
 ans  =
    4.6666667
-->2 + 3 / 4
 ans  =
    2.75
```

□

Ответ к упражнению 3.2 (*Скобки*) Для того чтобы явно указать последовательность выполнения операций, используются скобки "(" и ")":

```
-->2 * (3 + 4)
 ans  =
    14.
-->(2 + 3) * 4
 ans  =
    20.
-->(2 + 3) / 4
 ans  =
    1.25
-->3 / (2 + 4)
 ans  =
    0.5
```

□

Ответ к упражнению 3.3 (*Экспоненциальная запись чисел*) Для задания числовых значений со степенным множителем, например, $1.23456789 \cdot 10^{10}$, используется буква "d", которая разделяет мантиссу и экспоненту в записи числа, как показано ниже:

```
-->1.23456789d10
 ans  =
    1.235D+10
```

Для этой же цели может использоваться и буква "e", как в следующем фрагменте, где мы задаем константы $1.23456789 \cdot 10^{10}$ и $1.23456789 \cdot 10^{-5}$:

```
-->1.23456789e10
 ans  =
    1.235D+10
-->1.23456789e-5
 ans  =
    0.0000123
```

□

Ответ к упражнению 3.4 (*Функции*) Функция `sqrt` ведет себя в точном соответствии с математическим определением квадратного корня:

```
-->sqrt(4)
 ans  =
    2.
-->sqrt(9)
 ans  =
    3.
```

Для отрицательных значений аргумента `x` функция `sqrt` возвращает комплексное число `y`, являющееся решением уравнения $y^2 = x$:

```
-->sqrt(-1)
 ans  =
    i
-->sqrt(-2)
 ans  =
    1.4142136i
```

Функция `exp` обозначает экспоненту, а `log` - обратную ей функцию натурального логарифма:

```
-->exp(1)
 ans  =
    2.7182818
-->log(exp(2))
 ans  =
    2.
-->exp(log(2))
 ans  =
    2.
```

Функция `log10` служит для вычисления логарифма по основанию 10. Интересно отметить, что для целого x значение $\log_{10}(x)$ дает количество десятичных цифр в записи этого числа:

```
-->10 ^ 2
 ans  =
    100.
-->log10(10 ^ 2)
 ans  =
    2.
-->10 ^ log10(2)
 ans  =
    2.
```

Функция `sign` возвращает 1 при положительном значении аргумента, -1 при отрицательном и 0 в случае, когда аргумент равен 0:

```
-->sign(2)
 ans  =
    1.
-->sign(-2)
 ans  =
  - 1.
-->sign(0)
 ans  =
    0.
```

□

Ответ к упражнению 3.5 (*Тригонометрические функции*) Следующий фрагмент демонстрирует примеры использования функций `cos` и `sin`:

```
-->cos(0)
 ans  =
    1.
-->sin(0)
 ans  =
    0.
```

Вследствие ограниченной разрядности представления дробных чисел, результаты тригонометрических функций (как и любых других) округляются. Ниже мы проверяем известные

математические равенства $\cos\pi = -1$, $\sin\pi = 0$ и $\cos(\pi/4) = \sin(\pi/4)$ и убеждаемся, что в случае вычислений с плавающей точкой они выполняются лишь приближенно, хотя и с очень высокой точностью:

```
-->cos(%pi)
 ans  =
  - 1.
-->sin(%pi)
 ans  =
    1.225D-16
-->cos(%pi / 4) - sin(%pi / 4)
 ans  =
    1.110D-16
```

□

10.4 Ответы к упражнениям раздела 4.18

Ответ к упражнению 4.1 (*Плюс 1*) Получим вектор $(x_1+1, x_2+1, x_3+1, x_4+1)$ при заданном x. Используем обычный оператор сложения для того, чтобы увеличить значение каждого элемента на 1:

```
-->x = 1 : 4;
-->y = x + 1
 y  =
    2.    3.    4.    5.
```

□

Ответ к упражнению 4.2 (*Векторизованное умножение*) Вычислим вектор $(x_1y_1, x_2y_2, x_3y_3, x_4y_4)$ при заданных x и y, выполнив поэлементное умножение с использованием оператора ".*":

```
-->x = 1 : 4;
-->y = 5 : 8;
-->z = x .* y
 z  =
    5.    12.    21.    32.
```

□

Ответ к упражнению 4.3 (*Вектор обратных величин*) Получим вектор $\left(\frac{1}{x_1}, \frac{1}{x_2}, \frac{1}{x_3}, \frac{1}{x_4}\right)$, выполнив поэлементное деление с использованием оператора "./":

```
-->x = 1 : 4;
-->y = 1 ./ x
 y  =
    1.    0.5    0.3333333    0.25
```

Следующий пример демонстрирует, что использование обычного (а не поэлементного) оператора деления не дает требуемого результата, вместо этого возвращая решение y уравнения $xy = 1$:

```
-->y = 1 / x  // Неверный способ выполнения данного упражнения!
 y  =
    0.0333333
    0.0666667
    0.1
    0.1333333
```

□

Ответ к упражнению 4.4 (*Векторизованное деление*) Получим вектор $\left(\frac{x_1}{y_1}, \frac{x_2}{y_2}, \frac{x_3}{y_3}, \frac{x_4}{y_4}\right)$, выполнив поэлементное деление с использованием оператора "./":

```
-->x = 12 * (6 : 9);
-->y = 1 : 4;
-->z = x ./ y
 z  =
    72.    42.    32.    27.
```

□

Ответ к упражнению 4.5 (*Векторизованное возведение в степень*) Получим вектор $\left(x_1^2, x_2^2, x_3^2, x_4^2\right)$ при $x = (1, 2, 3, 4)$, выполнив поэлементное возведение в степень с использованием оператора ".^":

```
-->x = 1 : 4;
-->y = x .^ 2
 y  =
    1.     4.     9.     16.
```

□

Ответ к упражнению 4.6 (*Применение функции к вектору*) Получим вектор $(s_1, s_2, \ldots, s_{10})$ при $x \in [0, \pi]$, $s_i = \sin(x_i)$, применив функцию `sin` к вектору, полученному в результате вызова функции `linspace`:

```
-->x = linspace(0, %pi, 10);
-->y = sin(x)
 y  =
         column 1 to 6
    0.     0.3420201    0.6427876    0.8660254    0.9848078
 0.9848078
         column  7 to 10
    0.8660254    0.6427876    0.3420201    1.225D-16
```

□

Ответ к упражнению 4.7 (*Векторизованные функции*) Вычислим значения $y = f(x)$ функции f, заданной уравнением

$$f(x) = \log_{10}\left(r/10^x + 10^x\right) \tag{3}$$

при $r = 2.220.10^{-16}$ и $x \in [-16, 0]$. Следующий пример демонстрирует, как это можно сделать, используя операторы поэлементного деления "./" и возведения в степень ".^":

```
-->r = 2.220D-16;
-->x = linspace(-16, 0, 100);
-->y = log10(r ./ 10 .^ x + 10 .^ x);
```

Данная функция используется при вычислении оптимального шага численного дифференцирования, который равен `h=sqrt(%eps)`.

□

Список литературы

[1] Atlas - Automatically Tuned Linear Algebra Software. http://math-atlas.sourceforge.net.

[2] Cecill and free software. http://www.cecill.info.

[3] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos*. Springer, 2006.

[4] J.-P. Chancelier, F. Delebecque, C. Gomez, M. Goursat, R. Nikoukhah, and S. Steer. *Introduction à Scilab, Deuxième Édition*. Springer, 2007.

[5] The Scilab Consortium. Scilab. http://www.scilab.org.

[6] Intel. Intel math kernel library. http://software.intel.com/en-us/intel-mkl/.

[7] Cleve Moler. Numerical computing with MATLAB.

[8] Flexdock project. Flexdock project home. https://flexdock.dev.java.net/.

Предметный указатель