Пакеты прикладных программ

Н.Б. Шамрай

26 октября 2010 г.

Содержание

1 Знакомство с файловой системой Linux

Первое знакомство с операционной системы Linux стоит начать с освоения ее файловой системы. Напомним, что *файловая система* — это структура, с помощью которой ядро операционной системы предоставляет пользователям (в том числе процессам, приложениям и т.п.) ресурсы долговременной памяти (жесткие диски, CD-ROM, флеш-память и т.п.). Для пользователя файловая система повернута своей "видимой " стороной, которая выглядит как логическая структура каталогов и файлов, но существует и "невидимая" сторона, которая представляет собой достаточно сложный механизм, отвечающий, например, за запись файлов на различные носители, алгоритмы доступа к файлам, шифрование файлов и т.д. Нас будет интересовать только "видимая" сторона. Также отметим, что познакомиться предстоит с файловой системой типа ext2.

Пользователь обращается к файлу по имени. Имена файлов в Linux могут состоять из не более чем 255 символов, за исключением символа с кодом 0 и `/` (слэш). Не рекомендуется также включать в имена файлов следующие символы:

```
! @ # $ & ~ % * ( ) [ ]   ' '' \ : ; < > ` пробел
```

Отметим, что при создании имен файлов можно использовать один или несколько символов . (точка), например, *myfile_ create08.02.2007*. При этом теряет смысл такое понятие (принятое в DOS, Windows), как расширение имени файла, хотя все же зачастую последние части имени, отделенные точками, используют для обозначения файлов каких-то особых типов, например

**.tex* — для обозначения файлов, созданных в формате TEX или LATEX;
**.oc* — для обозначения файлов, содержащих код `octave`;
**.mp* — для обозначения файлов, содержащих код `metapost`.

Но если точка является первым символом имени, то данный файл является скрытым для некоторых команд, например, такие файлы не показываются при выполнении команды `ls`.

В Linux различаются символы верхнего и нижнего регистра в именах файлов, поэтому *Myfile.txt* и *myfile.txt* могут существовать одновременно и являться именами разных файлов.

С целью поддержания порядка в своем рабочем дисковом пространстве, файлы удобно группировать в каталоги. Каждый каталог может содержать как отдельные файлы, так и подкаталоги. В результате получается иерархическая структура каталогов.

В MS Windows или DOS каталоговая структура строиться отдельно для каждого физического носителя и корневой каталог каждой каталоговой структуры обозначается какой-либо буквой латинского алфавита. В Linux (и UNIX вообще) строится единая каталоговая структура для всех носителей, и единственный корневой каталог этой структуры обозначается символом `/`. В эту единую каталоговую структуру можно подключить любое число каталогов, физически расположенных на разных носителях (говорят "смонтировать носитель").

Имена каталогов строятся по тем же правилам, что и имена файлов. Любой каталог — это обычный файл, в котором перечислены все файлы и подкаталоги этого каталога.

Путем к файлу называется список имен вложенных друг в друга подкаталогов. Путь может начинаться с корневого каталога (полный путь), например,

/home/shamray/Document/TexFile/mydoc.tex

или позиционироваться относительно текущего каталога (относительный путь), например,

Document/TexFile/mydoc.tex

— если текущее местонахождение */home/shamray/*. Заканчивается путь собственно именем файла. Отметим, что текущий каталог всегда отображается в начале (в заголовке) командной строки оболочки shell. Кроме того, имеется специальная команда `pwd`, которая сообщает полный путь к текущему каталогу.

Для каждого зарегистрированного пользователя в системе Linux выделяется свой "домашний каталог (директория)". Обычно домашние каталоги размещаются в директории */home* и имеют имена, совпадающие с именем пользователя в системе, например, */home/shamray/* — домашний каталог пользователя shamray. Перейти в домашний каталог можно при помощи значка ~, например, пользователь shamray может обратиться к файлу, имеющему полный путь

/home/shamray/Document/TexFile/mydoc.tex

как к

~/Document/TexFile/mydoc.tex.

Когда пользователь входит в систему, текущим каталогом становится его домашняя директория.

А теперь начнем знакомство с командами, позволяющими работать с файлами и каталогами:

- `ls` — выводит имена файлов и подкаталогов текущего каталога. Если надо просмотреть содержимое какого-либо другого каталога, то команде `ls` необходимо указать полный или относительный путь к этому каталогу:

  `ls` */tmp*;

  `ls` *Document/TexFile/*.

  Команда `ls` может использоваться с указанием параметров. Например, если указать параметр `-l` (`ls -l`), то будут выведены не только имена файлов, но также данные о правах доступа, количество жестких ссылок или имен файла (для каталога число дополнительных блоков), имя владельца и группы, размер, дата последней модификации. При задании параметра `-t` (`ls -t`) при выводе файлы будут упорядочены по времени модификации. Параметр `-r` меняет порядок сортировки на обратный (используется вместе с параметрами `-l` и `-t`: `ls -lr, ls -ltr, ls -tr`).

- `mkdir` — позволяет создавать новые каталоги. В качестве аргумента необходимо указать имя создаваемого каталога:

  `mkdir` *MyDoc* — создать директорию *MyDoc* в текущем каталоге;

  `mkdir`*/Document/MyDoc* — создать директорию *MyDoc* в каталоге *Document* (при условии, что *Document* существует в текущем каталоге).

  Команда `mkdir` может использоваться с параметром `-p`, при этом будут создаваться все промежуточные каталоги, если они не существуют, например,

  `mkdir -p` *Document/TexFile/MyDoc* — создать в каталоге *Document* подкаталог *TexFile*, а затем в последнем директорию *MyDoc*.

- `cd` — служить для изменения текущего каталога. В качестве параметра необходимо указать полный или относительный (начиная с текущей директории) путь к нужному каталогу:

`cd` */home/shamray/Document/TexFile/MyDoc* — перейти в каталог *MyDoc* полный путь к которому */home/shamray/Document/TexFile/MyDoc*;

`cd` */Document/TexFile/MyDoc* — для пользователя shamray перейти в каталог *MyDoc*, который находится по пути *Document/TexFile/MyDoc* начиная с домашней директории;

`cd ..` — перейти на каталог выше;

`cd ~` — перейти в домашнюю директорию;

`cd` *../../Folder1/Folder2* — подняться на два "этажа" вверх и оттуда спуститься по пути *Folder1/Folder2*.

- `cat` — используется для создания файлов (как правило небольших), вывода содержимого одного или нескольких файлов на экран, копирования файлов, объединения нескольких файлов в один.

  `cat >` *newfile* — направить данные с клавиатуры в новый файл *newfile*, если *newfile* уже существует, то он будет переписан заново; после окончания ввода данных нажать комбинацию клавиш <Ctrl>+<D> или <Ctrl>+<C>;

  `cat` *filename* — вывести на экран содержимое файла *filename*;

  `cat` *file1* > *file2* — копировать *file1* в *file2*;

  `cat` *file1 file2 ... fileN* `>` *newfile* — объединить в указанной последовательности содержимое файлов *file1 file2 ... fileN* в новом файле *newfile*;

  `cat` *file1 file2 ... fileN* `>>` *oldfile* — дописать в указанной последовательности содержимое файлов *file1 file2 ... fileN* в конец файла *oldfile*.

- `cp` — служит для копирования файлов и директорий, общая форма вызова команды имеет вид

  `cp [options]` *source destination*

  и означает, что файл (каталог) *source* копируется в файл (каталог) *destination*. Для копирования надо иметь права на чтение копируемых файлов и права на запись в каталог, в который производится копирование. Если в качестве целевого указать существующий файл, то его содержимое будет затерто. Команда `cp` может использоваться как без дополнительных параметров (`options`) так и с параметрами, позволяющими получить дополнительные возможности при копировании. Вот некоторые, наиболее часто используемые опции:

  `-i` — если файл *destination* существует, то будет запрошено подтверждение на его перезапись;
  `-p` — сохраняет время модификации файла и максимально возможные полномочия. Без этой опции для нового файла задаются полномочия, соответствующие полномочиям запустившего команду пользователя;
  `-r` — если *source* — каталог, то копируется как он, так и все входящие в него подкаталоги, т.е. сохраняется исходная форма дерева;
  `-d` — эта опция при копировании символических ссылок сохраняет за файлом статус символической ссылки (иначе вместо ссылки копируется файл, на который дается ссылка);

`-f` — при копировании перезаписывать файлы (если такие уже есть) без дополнительных предупреждений.

- `mv` — служит для перемещения файлов и директорий в дереве каталогов, общая форма вызова

  `mv [option]` *source destination*

  и означает, что файл (каталог) *source* перемещается в файл (каталог) *destination*. Для перемещения необходимо иметь права на чтение перемещаемого файла и права на запись в каталог, в который производится перемещение. Дополнительные опции такие, как у команды `cp`. Команду `mv` также можно использовать для переименования (одного) файла:

  `mv` *oldfile newfile*

- `rm` — служит для удаления файлов, общая форма вызова

  `rm [option]` *filename*

  Для того, чтобы воспользоваться командой `rm` необходимо иметь право записи в каталоге, в котором расположен удаляемый файл *filename*. Полезно при вызове использовать опцию `-i`, чтобы получить дополнительный запрос на подтверждение операции. Удалять можно сразу несколько файлов

  `rm` *file1 file2 ... fileN*

  Если среди имен *file1 file2 ... fileN* есть каталоги, то их удаление не произойтет (система выдаст соответствующее сообщение). Для удаления каталога и его содержимого необходимо вызывать команду `rm` с опцией `-r`

  `rm -r` *dir*

  Фатальное удаление содержимого текущей директории делается командой `rm -r *`. Однако всегда следует помнить, что в Linux нет команды восстановления файлов после их удления.

2 Редактор `vi`

Многие великие (и не очень великие, и даже просто бесполезные) компьютерные творения начинались с создания обыкновенного текстового файла. Далеко за примерами ходить не надо, любой исходный код используемой Вами операционной системы, программного продукта, даже руководство, которое вы сейчас читаете зародилось на базе текстового файла. Поэтому смело можно утверждать, что первые шаги к вершинам мастерства Архитектора начинаются с освоения какого-либо текстового редактора. Наш (а следовательно и Ваш) выбор пал на экранный редактор `vi`.

`vi` называют экранным редактором, поскольку он использует в качестве рабочего поля весь экран терминала. Экран задействован большей частью для отображения редактируемого текста, а одну строку (последнюю) `vi` отводит для общения с пользователем. Практически во всех руководствах `vi` называют *" мощным средством для создания и редактирования файлов"*. И это правда, но человеку, который привык работать с так называемыми визуальными текстовыми процессами, где редактирование сводится к нажатию кнопки мыши на нужном пункте меню или нужной кнопки панели инструментов, первое знакомство с `vi` хочется забыть как страшный сон и вернуться к "мышиной возне" по так милым сердцу и глазу "кнопочкам" и "пунктикам". Чтобы подобные мысли не приходили к Вам в голову приведем несколько на Наш взгляд убедительных доводов в пользу освоения `vi`:

1) `vi` является стандартным редактором UNIX-систем и любой дистрибутив UNIX включает в себя инсталяционный файл этого программного продукта;

2) в `vi` заложены действительно широкие возможности по созданию/редактированию разного рода документов и текстов программ;

3) освоение команд `vi` тренирует память и заставляет "качать мозгу";

4) и наконец, чтобы выполнить задания по дисциплине и получить зачет, `vi` Вам просто необходим!

Итак, вы начали ощущать непреодолимую тягу к познанию таинств текстового редактора `vi` и Нам ничего не остается, как перейти от слов к делу (чтобы эта Ваша "тяга" не исчезла безвозвратно).

В `vi` предусмотрено **три режима работы**:

- командный режим — режим, в котором можно перемещаться по файлу и выполнять редактирующие команды над текстом. Команды вызываются обычными латинскими буквами. Переход в этот режим осуществляется путем нажатия клавиши `Esc`;

- режим ввода текста — режим, в котором нажатие на клавишу с печатным символом приводит к вставке этого символа в текст. Переход в этот режим осуществляется путем нажатия клавиш `a, A, i, I, c, C, o, O`, которые будут описаны ниже. Выход из этого состояния происходит при нажатии клавиши `Esc`;

- режим строчного редактора ED — режим, в котором выполняются более глобальные операции над текстом/файлом/редактором (например, сохранить файл, выйти из редактора, настроить редактор и т.д.). Переход в этот режим осуществляется из командного режима, путем нажатия клавиш `:` (двоеточие) или `/` (слэш), при этом курсор переводится на последнюю строку экрана и редактор ожидает ввода дальнейших команд. Вводимая команда отображается на экране и исполняется после нажатия клавиши `Enter`.

Работая с vi вы должны знать, что этот редактор работает не с файлом как таковым, а с его копией, помещаемой в буфер редактора. Поэтому все изменения, которые вы делаете в процессе редактирования, не отражаются в содержании реального файла до тех пор, пока вы не исполнили в явном виде команду записи-сохранения. Буфер редактора представляет собой на самом деле тоже файл, но в нем изменения отображаются почти немедленно. Это дает возможность с помощью специальных команд восстановить результаты редактирования после аварийных ситуаций, например, прерывание питания или неожиданная остановка работы системы.

Создание и открытие файла. Чтобы создать текстовый файл с именем *myfile* в командной строке необходимо набрать

vi *myfile*

и нажать клавишу Enter. Если файл с именем *myfile* уже существует, то открыть его в редакторе vi можно при помощи этой же последовательности команд. При этом файл откроется в командном режиме и курсор будет стоять в строке, где он (курсор) находился при последнем выходе из файла. Чтобы при открытии файла курсор встал на сроку с номером N, необходимо набрать команду vi +N *myfile*, при этом файл *mtfile* конечно должен существовать и его содержимое должно иметь строку N, например,

vi +100 *myfile* — откроет в редакторе vi файл *myfile* и поставит курсор в начало строки с номером 100.

Перемещение по тексту происходит с помощью клавиш "стрелочек", или команд

h — влево;
j — вниз;
k — вверх;
l — вправо;
0, Home — в начало текущей строки;
^ — перемещение на первый непробельный символ строки;
+ — первый символ следующей строки;
- — первый символ предыдущей строки;
n| — переместиться на n-ый символ текущей строки;
$, End — в конец текущей строки;
w, W — на слово вправо;
b, B — на слово влево;
e, E — переход на конец слова;

Перед каждой из перечисленных команд навигации (кроме 0 и ^) можно задать число повторений этой команды или в некоторых случаях номер строки, к которой команда должна быть применена, например,

5h — переместить курсор на 5 символов влево;
2$ — переход в конец строки, второй после текущей.

Отметим, что символ в верхнем регистре предполагает либо включения режима CapsLock, либо, при отключенном режиме CapsLock, одновременное нажатие клавиши Shift и необходимого символа.

Будучи текстовым редактором, vi понимает, что текст может состоять из предложений, заканчивающихся точкой (.) и абзацев, разделенных пустыми строками. Определенные команды vi позволяют перемещаться и позиционировать курсор относительно этих единиц текста.

`(` — начало текущего предложения;
`)` — начало следующего предложения;
`{` — начало текущего абзаца;
`}` — начало следующего абзаца;
`[` — начало текущего раздела;
`]` — начало следующего раздела.

Позиционирование может осуществляться также относительно строк, видимых в настоящий момент на экране терминала `vi`:

n`H` — перейти на *n*-ую видимую на экране строку сверху;
n`L` —- перейти на *n*-ую видимую на экране строку снизу;
n`G` — перейти на *n*-ую строку от начала файла;
`M` — перейти на среднюю строку экрана;
`z Enter` — сместить экран так, чтобы текущая строка стала первой;
`z.` — сместить экран так, чтобы текущая строка стала средней;
`z-` — сместить экран так, чтобы текущая строка стала последней.

Быстрый просмотр текста можно осуществить при помощи комбинаций клавиш

`Cntl- f, PageUp` — один экран вверх;
`Cntl- b, PageDown` — один экран вниз;
`Cntl- d` — полэкрана вверх;
`Cntl- u` — полэкрана вниз

перед нажатием которых, также можно указать число n повторений команды.

Если вам необходимо время от времени возвращаться к какой-то конкретной строке текста, то совсем необязательно запоминать ее номер (к тому же этот номер может меняться в процесссе редактирования файла), можно расставить метки и "прыгать" по ним в тексте. Метке надо дать имя — некоторый символ. Поставить метку можно двумя способами

`m` *символ*
или
`:ma` *символ*.

Перейти на метку с именем *символ* можно при помощи команды

`` ` `` *символ*.

Редактирование файла. Чтобы собственно наполнить содержимое файла символами, строками, абзацами, разделами и т.п. необходимо перевести редактор `vi` в режим ввода текста. В зависимости от того, в какой позиции вы хотите вставлять символы, переход в данный режим осуществляется из командного режима путем нажатия следующих символов:

`a` — вставка текста после курсора, может использоваться с аргументами:

`a/`z — вводить текст после первого обнаруженного символа z;

`A` — вставка текста в конце строки;
`i` — вставка текста перед курсором, аналогично команде `a` может использоваться с аргументами;
`I` — вставка текста перед первым непробельным символом в строке (начало строки);
`r` — замена символа в текущей позиции курсора;
`R` — замена текста, начиная с текущей позиции курсора;
`c` — замена текста, может используется с аргументами:

> `cw` — заменять текущее слово;
> `c/`z — заменить текст от текущего положения курсора до обнаружения первого символа z, сам z при этом не заменится;

`C` — замена текста до конца строки;
`o` — создание пустой строки после текущей;
`O` — создание пустой строки перед текущей.

Сохранение файла. Редактируя файл, не забывайте сохранять изменения. Сохранение проводится в режиме строчного редактора ED, перейти в который можно из командного режима, нажав двоеточие (:). При этом знак : и последующие команды буду отображаться в последней строке экрана. Возможны следующие варианты:

`:q!` — выход из `vi` без сохранения изменений;
`:w` — сохранение изменений не выходя из редактора;
`:w` *filename* — сохранение данных в файл и именем *filename*;
`:wq, :x` — сохранить изменения и выйти из редактора;
`:q` — выход из `vi`, если файл не был изменен, в противном случае `vi` выдаст предупреждение о несохраненных изменениях.

Итак, таперь вам известно как создать/открыть файл в редакторе `vi`, как просмотреть файл, как внести изменени (добавить текст, редактировать текст), как сохранить внесенные изменения. Продолжая освоение редактора, научимся быстро удалять, перемещать и копировать данные.

Удаление, перемещение, копирование. Переведем `vi` в командный режим. Удалить данные из текста помогут следующие коанды:

`dd, D` — удалить текущую строку;
n `dd` — удалить n строк, начиная с текущей (`5dd` — удалить 5 строк);
`d$` — удалить текст до конца строки;
`dw` — удалить текущее слово;
n `dw` — удалить n слов, начиная с текущего (`5dw` — удалить 5 слов);
`d)` — удалить текст до начала следующего предложения;
`d}` — удалить текст до начала следующего абзаца;
`d]` — удалить текст до начала следующего раздела;
`d/`*символ* — удалить весь текст, до первой встречи *символа* (`dd/`z — удалить все с текущей позиции до символа z);
`x, Delete` — удалить символ в позиции курсора;

Здесь надо сделать одно замечание, строкой в `vi` считается не экранная строка, а последовательность символов до перевода каретки (
n). Если строка больше 80 символов (значение по умолчанию), то она переносится на новую линию

(строку экрана). Команда `dd` удаляет всю строку вне зависимости от того, на скольких экранных линиях она размещается. Чтобы определить, где находится конец строки, нажмите клавишу `$`.

Последний удаленный текст `vi` автоматически помещает во внутренний буфур, таким образом его (текст) можно оттуда (из буфера) востановить на прежднее или какое-то другое место. В этом помогут команды

`p` — вставить текст до курсора;
`P` — вставить текст после курсора.

Просто скопировать текст в буфер можно при помощи команд:

`yy, Y` — скопировать текущую строку; n`yy` — скопировать n строк, начиная с текущей;
n,m`yy` — скопировать строки с n по m;
`yw` — скопировать слово;
n`yw` — скопировать n слов.

Также полезными могут оказаться команды:

`u` — отменить последнюю операцию (действие), при повторном применении этой комендды будет отменена только что сделанная отмена;

3 Издательская система LaTeX

LaTeX представляет собой пакет прикладных программ, в основе которого лежит система форматирования документов TeX. Родоначальником TeX является американский ученый Дональд Кнут, известный миру, в частности, своей многотомной монографией «Искусство программирования» и серией книг, посвященных методам вычислительной математики.

Будучи выдающимся математиком, Д. Кнут создал систему TeX для издания своих научных трудов, поскольку был совершенно недоволен тем, как проходит верстка его статей в процессе публикации. Впоследствии профессор Кнут предоставил TeX во всеобщее пользование при условии сохранения неизменным тщательно отлаженного им кода ядра. Поэтому, в отличие от многих других издательских систем, для которых несовместимость различных версий — частое явление, все построенные на основе TeX диалекты характеризуются полной совместимостью. Среди таких диалектов особое место и занимает LaTeX(разработчик Лесли Лампорт).

Система LaTeX вместо привычного многим способа набора текста в режиме «пишущей машинки» предоставляет широкий набор инструментов (команд и процедур) для создания документов с высочайшим качеством дизайна. Так, например, LaTeX содержит удобные средства генерации алфавитного указателя, списка литературы, рисунков и таблиц, развитые средства импортирования графики, обеспечивает автоматическую нумерацию формул, ссылок и других подобных объектов в сочетании с эффективным механизмом перекрестного цитирования. Особого совершенства LaTeX достиг в форматировании математических формул, поэтому LaTeX особенно популярен в научных кругах. Практически во всех издательствах научно-технической литературы при приеме работ для публикации предпочтение отдается статьям, подготовленным в LaTeX.

3.1 Входной файл

Исходными данными для системы LaTeX является обычный текстовый файл, в котором содержится вся информация о стиле, содержании и т.п. документа. Такой файл можно создать в любом текстовом редакторе. Как правило к имени файла с исходным текстом добавляют расширение `.tex`, например, `myreport.tex`. Верстка документа осуществляется при помощи специальных команд. В настоящее время разработано много редакторов по созданию входного файла для LaTeX помогающих быстро вводить нужные команды. Например, к числу таких редакторов относятся `Kile`, `LEd`, `WinEdt` и т.д.

Исходный текст условно можно разделить на две части. В первой — задаются настройки, относящиеся ко всему документу: объявляется класс документа, подключаются нужные пакеты, описываются стили и форматы, объявляются команды и т.д. Эту часть называют преамбулой. Во второй части — программируется структура и тело документа.

Открывает преамбулу команда

`\documentclass[`*options*`]{`*class*`}`

которая определяет основную структуру печатного документа.

По синтаксису LaTeX в квадратных скобках [ ] заключается необязательный аргумент команды, в фигурных скобках { } — обязательный. Необязательный аргумент вместе со скобками может отсутствовать.

Условимся, что любой текст, напечатанный `машинописным` шрифтом, фигурные или квадратные скобки следует вводить во входной файл в точности так, как они прописаны в данном документе. Напротив, текст набранный *курсивом* может меняться.

Аргумент *class* обязательный, он определяет класс печатного документа и может принимать одно из шести стандартных значений: `article, letter, report, book, proc, slides`. Класс `article`(статья) является наиболее универсальным и удовлетворяет большинству требований к документу. Для технического отчета больше подходит класс `report`(отчет), название класса `book` говорит само за себя. Если предстоит написать десяток писем нескольким адресантам, то следует выбрать класс `letter`(письмо). Класс `proc` предназначен для научных публикаций в трудах конференций — текст печатается в две колонки. Класс `slides`(слайды) используется для подготовки демонстрационных материалов. Перечисленные классы отличаются друг от друга набором допустимых команд, правда стоит отметить, что число команд, специфичных для каждого класса невелико. В основном различия заключаются в размерах шрифтов заголовков, способу нумерации глав, рисунков и таблиц. Для изучения возможностей LaTeX будем работать с классом `article`.

Как уже было отмечено, аргумент *options* может отсутствовать, тогда все опции документа будут использованы по умолчанию. Если необходимо изменить какие-то параметры, то нужные значения перечисляют через запятую. Так, например, команда

```
\documentclass[12pt,a4paper,twocolumn]{article}
```

устанавливает класс документа `article`, текст которого будет печататься 12 шрифтом (`12pt`) на странице размера A4 (`a4paper`) в две колонки (`twocolumn`).

После команды `\documentclass`, как правило, идет одна или несколько команд

```
\usepackage[options]{package},
```

с помощью которой по необходимости к документу присоединяются служебные файлы (пакеты) LaTeX.

Для начала работы с русскоязычными документами достаточно подключить два пакета `inputenc` и `babel`. Пакет `inputenc` позволяет пользователю установить внутреннюю кодировку шрифтов LaTeX. Так, например, чтобы откомпилировать исходный текст, подготовленный в среде Unix, достаточно включить в преамбулу команду

```
\usepackage[koi8-r]{inputenc}.
```

Если работа идет в среде Windows, то вместо опции `koi8-r` необходимо указать `cp1251`. Пакет `babel` предназначен для поддержки языков. Для русскоязычного документа необходимо указать опцию `russian`:

```
\usepackage[russian]{babel}.
```

В преамбуле входного файла также можно указать параметры страниц документа(отступы от границ листа, межстрочный интервал, размер и вид колонтитулов и т.д.), его название, авторов, дату создания. Все это делается при помощи специальных команд. Вот некоторые из них:

`\textwidth`	—	ширина текста;
`\textheight`	—	высота текста (без колонтитулов);
`\topmargin`	—	расстояние между верхним краем листа и верхним колонтитулом минус 1 дюйм;
`\oddsidemargin`	—	расстояние между левым краем листа и левым краем текста на правой странице минус 1 дюйм;
`\evensidemargin`	—	то же самое для левосторонней страницы.

Перечисленные величины удобно задавать в сантиметрах (`cm`) или миллиметрах (`mm`). Синтаксис при использовании следующий

```
\textwidth = 170mm
```

Название документа можно определить с помощью команды `\title{title}`. Авторов указывают посредством команды `\author{author}`. Дата выпуска печатного документа определяется командой `\date{date}`. Например,

```
\title{Пакеты прикладных программ}
\author{Н.Б. Шамрай}
\date{14 февраля 2008 г.}
```

Фамилии авторов разделяются командой `\and`, но можно также писать их через запятую или пробел. Сами по себе команды `\title, \author, \date` ничего не печатают. Титульную информацию формирует команда `\maketitle`, которая будет вызвана в теле документа. Если `\maketitle` действительно используется, то команды `\title, \author` являются обязательными, `\date` — необязательная. При отсутствии `\date` в титуле печатается текущая дата.

После преамбулы формируется собственно текст документа. Начало текста обозначается обязательной командой

```
\begin{document}
```

конец — обязательной командой

```
\end{document}
```

Все, что следует после `\end{document}`, LaTeX просто игнорирует.

При работе с LaTeXдокументом может оказаться необходимым закомментировать часть содержимого исходного текста. Для этого используют знак процента % — LaTeX игнорирует любой текст между символов % и концом текущей строки.

Структура и дизайн тела документа зависят от фантазий и потребностей его создателей. Как правило, тело начинается с головы — титула, которую формирует команда `\maketitle`. Потом может идти аннотация, границы которой задаются командами `\begin{abstract}`, `\end{abstract}`. Далее тело может делиться на разделы, секции, начиняться текстом, формулами, таблицами, рисунками и т.д. Все это детально будет рассмотрено далее.

Подводя итоги всего вышесказанного создадим простой документ LaTeX используя описанные выше команды. При помощи редактора `vi` откроем файл `first.tex`, в который внесем следующие команды

```
\documentclass[12pt]{article}     %  выбираем класс документа
\usepackage[koi8-r]{inputenc}     %  определяем внутреннюю кодировку
\usepackage[russian]{babel}       %  поддержка русского языка

\textwidth = 175mm                %  ширина печатного текста
\textheight = 250mm               %  высота печатного текста
\topmargin = -15mm                %  отступ от верхнего края листа
\oddsidemargin = -5mm             %  отступ от левого края листа

\title{Первый документ в \LaTeX}  %  заголовок документа
\author{П.П. Петров}              %  авторы
\date{29 февраля 2008 г.}         %  дата печати

\begin{document}                  %  конец преамбулы, начало текста документа
\maketitle                        %  печать заголовка
```

```
Освоив команды и синтаксис языка \LaTeX\ я смогу
создавать печатные документы с высоким качеством дизайна.
 \end{document}                    %  конец документа
```

Чтобы посмотреть, что получилось, созданный исходный файл `first.tex` необходимо обработать средствами пакета LaTeX. Первое, откомпилируем `first.tex` — для этого в командной строке консоли набираем

```
latex first.tex
```

Процесс компиляции будет отображен на экране. Если в исходном файле допущены ошибки, то компилятор `latex` сообщит об этом, как правило, указывается тип ошибки и номер строки, в которой она обнаружена. Вслед за строкой локализации ошибки `latex` выводит знак вопроса в начале новой строки и ждет реакции пользователя. В ответ можно

— либо нажать клавишу `Enter`, чтобы проигнорировать ошибку,

— либо ввести букву `h` потом `Enter`, чтобы получить рекомендацию относительно дальнейших действий (на английском языке),

— либо букву `x` потом `Enter`, чтобы немедленно прекратить компиляцию исходного файла,

— либо букву `q` потом `Enter`, чтобы LaTeX больше не останавливался, встретив новые ошибки.

В любом случае информация об ошибках будет записана в файл `first.log`, которые автоматически создается в текущей папке и содержит информацию о процессе компиляции. После устранения ошибок компиляция повторяется. При успешной обработке `first.tex` в текущей папке помимо `first.log` автоматически появятся и другие файлы, например, `first.dvi` — содержит смакетированный текст печатного документа в независимости от печатных устройств формате, `first.aux` — содержит список ключей, используемых для перекрестного цитирования формул, таблиц, рисунков и вообще любых объектов, имеющих номер.

Последние версии LaTeX в результате успешной компиляции сразу создают `pdf`-файл для последующего просмотра документа. Если этого не произошло, то можно преобразовать смакетированный документ `first.dvi` в формат PostScript при помощи команды

```
dvips first.dvi
```

и просмотреть результат программой GostView

```
gv first.ps
```

Возможности пакета LaTeX по созданию красивых документов, удовлетворяющих требованиям и желаниям авторов, практически неограничены. Если все же для реализации задуманной верстки в пакете не нашлось специальной команды, процедуры, декларации и пр., то обладая определенными навыками и знаниями можно самим запрограммировать так необходимые инструменты (создать свой стилевой файл), и успешно пользоваться ими в среде LaTeX.

В рамках данного курса будут описаны лишь некоторые элементы языка LaTeX, освоение которых вполне достаточно для создания читабельных работ радующих глаз (рефератов, курсовых и дипломных работ, отчетов).

Везде далее предполагается, что класс документа определен как `article`.

3.2 Основные конструкции LaTeX

Формирование документа осуществляется с помощью таких конструкций языка LaTeX как команды, декларации и процедуры.

Общий вид команды следующий: символ обратного слеша (\) начинает команду, далее идет имя команды — ключевое слово библиотеки LaTeX затем, согласно синтаксиса языка, в фигурных скобках ({ }) указываются обязательные аргументы, в квадратных ([ ]) — необязательные

```
\documentclass[12pt,a4paper]{article}
```

Команды не имеющие аргументов и предписывающие LaTeX какие-то действия называются декларациями. Область действия декларации можно выделить, группируя текст в блок с помощью фигурных скобок: левая скобка { открывает текущий блок, а правая } — его закрывает.

```
{\Large Устанавливаем большой шрифт}
```

Устанавливаем большой шрифт

Процедуры LaTeX объявляются с помощью так называемых командных скобок `\begin{`*proc*`}` и `\end{`*proc*`}`, где *proc* — это имя процедуры, зарезервированное слово библиотеки LaTeX. Между командными скобками находится тело процедуры. Команда `\begin` может иметь дополнительные обязательные и необязательные аргументы

```
\begin{tabular}{lcr}...\end{tabular}
```

Подробнее о командах, декларациях и процедурах будет написано далее, по мере изучения возможностей языка LaTeX.

3.3 Слова и предложения

Текст документа набирается в исходном файле. Формат ввода текста может быть произвольным, LaTeX игнорирует то, как набран исходный текст и фиксирует только концы слов, предложений и абзацев.

Слова отделяются друг от друга пробелами. Число следующих подряд пробелов несущественно, LaTeX все равно интерпретирует их как один. Предложения заканчиваются точкой (.), восклицательным (!) или вопросительным (?) знаками, за которыми следует пробел. После этих знаков LaTeX чуть увеличивает пробел. Пустая строка означает конец абзаца. Несколько подряд идущих пустых строк воспримутся LaTeX как одна, т.е. также конец абзаца.

Для изображения двойных левых (“) и правых (”) кавычек можно использовать команды `\textquotedblleft` и `\textquotedblright`, или набирать два символа ’ подряд: ’’ (но в последнем случае отображение будет зависеть от интерпретации пакетом `babel`). Русские кавычки (« ») можно набрать при помощи команд << и >>.

```
\textquotedblleft Учиться, учиться и еще
раз учиться \ldots\textquotedblright
```

“ Учиться, учиться и еще раз учиться . . . ”

```
<< Делу время --- потехе час.>>
```

«Делу время — потехе час.»

В пакете LaTeX используются три различных размера дефисов (тире): один дефис - используется в составных словах:

```
кто-то, как-нибудь, где-либо
```

кто-то, как-нибудь, где-либо

Для указания диапазона чисел лучше использовать удлиненное тире (–), в исходном файле для этого набирают два подряд дефиса --

```
1--9, 2008--2020
```

1–9, 2008–2020

Длинное тире (—), используемое в предложениях, обозначается тремя идущими подряд дефисами ---

```
Учение --- свет, неученье --- тьма.
```

Учение — свет, неученье — тьма.

Подстрочные примечания (сноску) печатает команда `\footnote{txtnote}`. Ее надо ставить в том месте исходного текста, где в печатном документе должен появиться маркер примечания. Аргументом команды является текст примечания.

```
Два брата\footnote{Петровы Василий и
Иван} --- акробата.
```

Два брата[1]— акробата.

[1]Петровы Василий и Иван

Первая строка каждого абзаца начинается с отступа фиксированной длины. Однако, первый абзац после заголовка отступа не имеет — это традиция, сложившаяся в английском языке. Для отмены отступа для абзаца служит команда `\noindent`, вставляет отступ команда `\indent` (но даже она бессильна перед магией первого абзаца, здесь необходимы другие инструменты).

LaTeX автоматически разбивает текст на строки, делает переносы слов, выравнивает текст по ширине листа. Если в тексте имеются логически связанные сочетания слов (Рис. 1, стр. 3–6), которые желательно располагать в одной и той же строке (не разрывать по пробелу), то используется символ тильда (~). Этот символ, вставленный между соседними словами, создаст пробел, на котором LaTeX никогда не разорвет строку

```
Рис.~3, стр.~3--6
```

Если LaTeX не нашел возможного способа переноса слова, то можно помочь ему, указав место переноса командой `\-`

```
бро\-не\-транс\-пор\-тер
```

По умолчанию текст документа выравнивается по ширине. Если необходимо центрировать какую-то часть текста, то используют процедуру `center`

```
\begin{center}
Этот текст центрируется.
\end {center}
А этот --- нет.
```

Этот текст центрируется.

А этот — нет.

Выравнивание текста по левой или правой границам осуществляется процедурами `flushleft` и `flushright` соответственно

```
\begin{flushleft}
Этот текст будет слева.
\end {flushleft}
\begin{flushright}
А этот --- справа.
\end {flushright}
```

Этот текст будет слева.

А этот — справа.

При форматировании текста печатного документа полезными могут оказаться следующие команды

`\\`	— принудительный переход на новую строку в печатном документе;
`\\[`*2cm*`]`	— переход на новую строку и увеличение отступа до следующей строки на 2 cm;
`\␣`	— принудительный отступ между словами (символ ␣ обозначает пробел);
`\vspace{`*20mm*`}`	— вертикальный отступ на 20мм;
`\hspace{`*10mm*`}`	— горизонтальный отступ на 10мм;
`\hfill`	— вставляет горизонтальный пробел бесконечно растяжимой длины;
`\vfill`	— вставляет вертикальный пробел бесконечно растяжимой длины;
`\hrulefill`	— заполняет свободный промежуток в строке горизонтальной линией;
`\dotfill`	— заполняет свободный промежуток в строке точками.

```
A\hfill и\hfill B
A\hrulefill и \hrulefill B
A\dotfill и \dotfill B
```

A и B
A и B
A и B

Если необходимо в печатном документе отобразить текст в точности так, как он набран в исходном файле, то используют процедуру `verbatim`

```
\begin{verbatim}
Что хочу --- ворочу:
{{%&_$${{
\end{ verbatim}
```

```
Что хочу --- ворочу:
{{%&_$${{
```

3.4 Шрифты

Выбор шрифта подчинен логической структуре текста. LaTeX различает форму, серию и семейство шрифта. В таблице 1 перечислены возможные варианты шрифта и соответствующие команды их определяющие.

По умолчанию используется шрифт формы upright, серии medium и семейства roman.

```
\textsc{Корабли \textit{лавировали,
\textbf{лавировали,
\textsf{да не вылавировали.} } }}
```

Корабли *лавировали,* ***лавировали,***
да не вылавировали.

Как видно из примера, вложенная команда `textit` отменяет форму капитель, объявленную командой `textsc`. Это же правило будет работать и для вложенных серий и семейств шрифтов, поскольку у текста может быть определена только одна форма, серия и семейство.

Таблица 1: Команды и декларации переключения начертания шрифтов

Форма шрифтов		
Upright shape (прямая)	`\textup{Upright shape}`	`{\upshape Upright shape}`
Italic shape (курсивная)	`\textit{Italic shape}`	`{\itshape Italic shape}`
Slanted shape (наклонная)	`\textsl{Slanted shape}`	`{\slshape Slanted shape}`
Small caps shape (капитель)	`\textsc{Small caps}`	`{\scshape Small caps}`
Серия шрифтов		
Medium series (средняя)	`\textmd{Medium series}`	`{\mdseries Medium series}`
Boldface series (полужирная)	`\textbf{Boldface series}`	`{\bfseries Boldface series}`
Семейство шрифтов		
Roman family (романский)	`\textrm{Roman family}`	`{\rmfamily Roman family}`
Sans serif family (рубленный)	`\textsf{Sans serif}`	`{\sffamily Sans serif}`
`Typewriter family (машинописный)`	`\texttt{Typewriter}`	`{\ttfamily Typewriter}`

Длинные имена команд переключения шрифтов вообще говоря не предназначены для частого использования в исходном тексте документа. Их следует использовать в определении других команд, формирующих логическую структуру текста. Но LaTeX позволяет переопределять имеющиеся и создавать новые команды, и этим при желании можно воспользоваться (см. раздел 3.9).

В таблице 2 приведен набор деклараций для изменения размера (кегля) шрифта.

Таблица 2: Декларации переключения размера шрифтов

`{\tiny крошечный}`	крошечный	`{\large большой}`	большой
`{\scriptsize индексный}`	индексный	`{\Large еще больше}`	еще больше
`{\footnotesize подстрочный}`	подстрочный	`{\LARGE еще больше}`	еще больше
`{\small маленький}`	маленький	`{\huge огромный}`	огромный
`{\normalsize стандартный}`	стандартный	`{\Huge огромный}`	огромный

По умолчанию действует декларация `normalsize`. Стандартный размер шрифта зависит от опции (`10pt, 11pt 12pt`), определенных в команде `\documentclass`. В таблице 3 приведены размеры шрифтов соответствующие разным опциям.

3.5 Команды секционирования

Документ большого содержания, как правило, разбивают на логически завершенные части — разделы (главы, параграфы, секции и т.д.). С точки зрения LaTeX разделы образуют иерархическую структуру, каждый элемент которой начинается с одной из команд секционирования. Для класса документа `article` имеется три уровня вложенности разделов

`\section{`*head*`}` `\subsection{`*head*`}` `\subsubsection{`*head*`}`

где обязательный аргумент *head* содержит название раздела.

Таблица 3: Абсолютные размеры шрифтов в стандартных классах

Размер	опция `10pt`	опция `11pt`	опция `12pt`
`\tiny`	5pt	6pt	6pt
`\scriptsize`	7pt	8pt	8pt
`\footnotesize`	8pt	9 pt	10 pt
`\small`	9pt	10pt	11pt
`\normalsize`	10pt	11pt	12pt
`\large`	12pt	12pt	14pt
`\Large`	14pt	14pt	17pt
`\LARGE`	17pt	17pt	20pt
`\huge`	20pt	20pt	25pt
`\Huge`	25pt	25pt	25pt

Нумерация разделов проходит автоматически. Но если к командам секционирования добавить знак звездочка (*)

`\section*{`*head*`}` `\subsection*{`*head*`}` `\subsubsection*{`*head*`}`

то разделы нумероваться не будет.

Если в печатном документе имеется приложение, то оно начинается с декларации

```
\appendix
```

и использует те же команды секционирования, что и основная часть текста. Декларация `\appendix` изменяет способ нумерации разделов. После нее начинается новый отсчет разделов, причем первая секция обозначается буквой А, вторая — Б и т.д.

Информацию о разделах LaTeX записывает в файл с расширением `toc`. Чтобы напечатать оглавление, в текст входного файла нужно вставить команду `\tableofcontents`. В оглавление автоматически помещаются только те разделы, которые были пронумерованы LaTeX. Если раздел в документе не нумеруется, но должен быть помещен в содержание (например, такие разделы как введение, предисловие, заключение), то информацию о нем прописывается в `toc`-файл при помощи команды

`\addcontentsline{toc}{`*unit*`}{`*entry*`}`

где *unit* — тип раздела (`section, subsection, subsubsection`), *entry* — имя раздела.

```
\section*{Введение}
\addcontentsline{toc}{section}{Введение}
```

Если каждый раздел необходимо начинать с новой страницы, то команде секционирования должна предшествовать команда

```
\newpage
```

название которой говорит само за себя.

3.6 Списки

LaTeX предлагает три процедуры для составления списков

`\begin{itemize}`	*item-list*	`\end{itemize}`	маркеруемый список
`\begin{enumerate}`	*item-list*	`\end{enumerate}`	нумеруемый список
`\begin{description}`	*item-list*	`\end{description}`	маркеруемый список

Тело процедуры *item-list* состоит из последовательности записей, начинающихся с команды

`\item [`*mark*`]`

Команда `item` помечает запись меткой *mark*. Если необязательный аргумент *mark* отсутствует, то LaTeX использует метку по умолчанию. Каждый из списков допускает четыре уровня вложенности.

```
\begin{itemize}
\item Первая запись первого уровня
\begin{itemize}
\item Первая запись второго уровня
\begin{itemize}
\item Третий уровень
\begin{itemize}
\item Четвертый уровень
\end{itemize}
\end{itemize}
\item Вторая запись второго уровня
\end{itemize}
\item Вторая запись первого уровня
\end{itemize}
```

- Первая запись первого уровня
  - Первая запись второго уровня
    - Третий уровень
      - Четвертый уровень
  - Вторая запись второго уровня
- Вторая запись первого уровня

В примере использовались метки по умолчанию. Один из способов изменить значения маркеров заключается в определении аргумента *mark*.

```
\begin{itemize}
\item [--] Первая запись первого уровня
\begin{itemize}
\item [>]Первая запись второго уровня
\begin{itemize}
\item [\textbullet] Третий уровень
\begin{itemize}
\item [!] Четвертый уровень
\end{itemize}
\end{itemize}
\item [>] Вторая запись второго уровня
\end{itemize}
\item [--]Вторая запись первого уровня
\end{itemize}
```

– Первая запись первого уровня
  > Первая запись второго уровня
    • Третий уровень
      ! Четвертый уровень
  > Вторая запись второго уровня
– Вторая запись первого уровня

Нумеруемый список по умолчанию выглядит так

```
\begin{enumerate}
\item Первая запись первого уровня
\begin{enumerate}
\item Первая запись второго уровня
\begin{enumerate}
\item Третий уровень
\begin{enumerate}
\item Четвертый уровень
\end{enumerate}
\end{enumerate}
\item Вторая запись второго уровня
\end{enumerate}
\item Вторая запись первого уровня
\end{enumerate}
```

1. Первая запись первого уровня
   (a) Первая запись второго уровня
      i. Третий уровень
         A. Четвертый уровень
   (b) Вторая запись второго уровня
2. Вторая запись первого уровня

Способы нумерации LaTeX по умолчанию можно менять при помощи специальных команд. Вид маркеров зависит от фантазии автора (символы, картинки и т.п.). Изучение этих возможностей оставляется студенту на самостоятельное образование.

В процедуре `description` метка по умолчанию отсутствует. Поэтому в команде `item` обязательно присутствует аргумент *mark*. По умолчанию метка печатается полужирным шрифтом, но это легко можно изменить используя команды раздела 3.4.

```
\begin{description}
\item [Утка] Водоплавающая птица.
\item [Корова] Парнокопытное животное.
\item [\itshape Единорог] Сказочное
животное.
\end{decsription}
```

Утка Водоплавающая птица.

Корова Парнокопытное животное.

Единорог Сказочное животное.

Для создания смешанных списков (например, внутри нумеруемого списка идет маркеруемый) процедуры `itemize, enumerate, description` вкладываются друг в друга.

3.7 Перекрестное цитирование

LaTeX автоматически нумерует разделы, страницы, формулы и многое другое. Схема организации печатного документа LaTeX позволяет делать ссылки вперед и назад по тексту документа на номера любых пронумерованных объектов. При изменении числа таких объектов ссылки автоматически перенумеруются.

Чтобы сослаться на какой-либо пронумерованный объект, необходимо в первую очередь его пометить с помощью команды

`\label{`*namekey*`}`

где *namekey* — это имя метки (произвольное слово прописанное автором). Имя метки для каждого нумеруемого объекта должно быть уникальным. Для печати ссылки в тексте документа используют команды

`\ref{`*namekey*`}` — печатает номер объекта, помеченного меткой *namekey*

`\pageref{`*namekey*`}` — печатает ссылку на номер страницы, на которую попадает метка *namekey*

```
\subsection{Перекрестное
цитирование}\label{ss_marks}
Текущий раздел имеет номер
\ref{ss_marks} и начинается на странице
\pageref{ss_marks}.
```

Текущий раздел имеет номер 3.7 и начинается на странице 22.

3.8 Математические формулы

3.8.1 Основные процедуры

Для создания математических формул в LaTeX необходимо включить специальный математический режим форматирования — перейти в математическую моду. В этом деле помогут три процедуры

`$ ... $` — размещает небольшие формулы внутри абзаца;

`\[ ... \]` — печатает формулу на отдельной строке;

`\begin{equation}` … `\end{equation}` — печатает формулу на отдельной строке и автоматически нумерует ее.

Формулу, созданную с помощью процедуры `enumerate` можно помечать, применяя команду `label`, чтобы организовать ссылку на нее в любом месте печатного документа.

```
Небольшие формулы типа $E=mc^2$
печатают внутри абзаца.

В отдельной строке
печатают сложные формулы, например,
\[
\sum_{n=1}^{\infty}
\frac{(-1)^n}{1+n}=\ln 2.
\]

Для последующего цитирования формулы
нумеруют, например,
\begin{equation}\label{math_int}
\int_{-\infty}^{+\infty}
e^{-x^2}dx=\sqrt{\pi}
\end{equation}
Уравнение (\ref{math_int}) приведено
на стр.~\pageref{math_int}.
```

Небольшие формулы типа $E = mc^2$ печатают внутри абзаца.
В отдельной строке печатают сложные формулы, например,

$$\sum_{n=1}^{\infty} \frac{(-1)^n}{1+n} = \ln 2.$$

Для последующего цитирования формулы нумеруют, например,

$$\int_{-\infty}^{+\infty} e^{-x^2} dx = \sqrt{\pi} \qquad (1)$$

Уравнение (1) приведено на стр. 23.

По умолчанию все включенные в документ формулы выравниваются по центру и нумеруются справа. Опция `fleqn` команды `\documentclass` устанавливает выравнивание слева, опция `leqno` изменяет правую нумерацию формул на левую.

Специальные математические символы производятся командами, которые перечислены в таблицах 9-18. приложения А. Все пробелы в исходном тексте математической формулы LaTeX игнорирует.

Символы, которые используются в формулах для логического выделения математического выражения называют разделителями (скобки, слэш, стрелки и т.п.). Чтобы подстроить размер разделителя под размер формулы необходимо дополнительно использовать команды `\left` и `\right`

```
\begin{equation}
\pi(n)=\sum_{k=2}^n
\left\lfloor
 \frac{\phi(k)}{k-1}
\right\rfloor
\end{equation}
```

$$\pi(n) = \sum_{k=2}^{n} \left\lfloor \frac{\phi(k)}{k-1} \right\rfloor \tag{2}$$

Если один из разделителей не нужен, то его достаточно заменить точкой: команды `\left.` и `\right.` производят невидимые разделители.

```
\[
\frac{a+1}{b} \left/
\frac{c+1}{d} \right.
\]
```

$$\frac{a+1}{b} \Big/ \frac{c+1}{d}$$

3.8.2 Основные структуры

Индексы. Верхний индекс вводится командой `^`, а нижний — командой `_`. Если аргумент этих команд состоит из более чем одного символа, то его необходимо заключить в круглые скобки. Порядок следования команд верхнего и нижнего индексов не имеет значения.

```
$x^2$, $x^{2y}$, $x^{2^y}$
$x_2$, $x_{y_2}$, $x_i^{2n}$
$x_1^2=x^2_1$, $y'_1+y''_2={g'}^2$
```

x^2, x^{2y}, x^{2^y}
x_2, x_{y_2}, x_i^{2n}
$x_1^2 = x_1^2$, $y'_1 + y''_2 = g'^2$

Индексы у некоторых символов переменного размера могут располагаться либо справа от изображения символа, либо под ним, либо под ним.

```
Нетрудно заметить разницу между
изображением формулы, вынесенной
на отдельную строку
\[
\int_a^b ydx=h\sum_{i=0}^{n-1}y_i
\]
и изображением той же формулы внутри абзаца
$\int_a^b ydx=h\sum_{i=0}^{n-1}y_i$
```

Нетрудно заметить разницу между изображением формулы, вынесенной на отдельную строку

$$\int_a^b ydx = h\sum_{i=0}^{n-1} y_i$$

и изображением той же формулы внутри абзаца $\int_a^b ydx = h\sum_{i=0}^{n-1} y_i$

Позиционированием индексов можно управлять при помощи команд

```
\limits
\nolimits
```

Команда `\limits` указывает, что индексы нужно позиционировать над и под символом, команда `\nolimits` имеет обратное действие.

```
\[ \int\limits_0^\infty
\sum\nolimits_{n=1}^m\]

$\lim\limits_{x\to 0}\sin(x)=0$
```

$$\int\limits_0^\infty \sum\nolimits_{n=1}^m$$

$$\lim_{x\to 0}\sin(x)=0$$

Дроби. В LaTeX дроби создаются либо при помощи символа /, либо при помощи команды

```
\frac{числитель}{знаменатель}
```

```
Деление на $n/2$ дает
$(m+n)/2$.

\[
x=\frac{y^2+z/3}{4+\frac{y}{z+a}}
\]
```

Деление на $n/2$ дает $(m+n)/2$.

$$x=\frac{y^2+z/3}{4+\frac{y}{z+a}}$$

Корни. Корень степени n из выражения *math* печатает команда

```
\sqrt[n]{math}
```

```
$\sqrt{2}$
$\sqrt[4]{2+x}$
$\sqrt[3]{x^2+\sqrt\alpha}$
```

$\sqrt{2}$
$\sqrt[4]{2+x}$
$\sqrt[3]{x^2+\sqrt{\alpha}}$

Размещение объектов друг над другом. Если математическое выражение состоит из объектов размещенных друг над другом, то сконструировать в LaTeX его можно при помощи команды

```
\stackrel{top}{bottom}
```

где первый аргумент *top* печатается над вторым аргументов *bottom*.

```
$\stackrel{\Leftrightarrow}{A}$
$A \stackrel{a'}{\to} D$
$\vec{v} \stackrel{\rm def}{\equiv}
(v_x, v_y, v_z)$
```

$\stackrel{\Leftrightarrow}{A}$
$A \stackrel{a'}{\to} D$
$\vec{v} \stackrel{\rm def}{\equiv} (v_x, v_y, v_z)$

Надчеркивание и подчеркивание математического выражения *math* реализуется командами

```
\overline{math}
\underline{math}
```

```
$\overline{x^2+\bar y}=\underline{5z}$
```
$\overline{x^2+\bar y}=\underline{5z}$

Горизонтальные фигурные скобки над и под выражением *math* вставляют команды

```
\overbrace{math}
\underbrace{math}
```

```
$\underbrace{a+\overbrace{b+c}+d}$

$\underbrace{a+\
overbrace{b+\cdots+c}^5+d}_7$
```
$\underbrace{a+\overbrace{b+c}+d}$

$\underbrace{a+\overbrace{b+\cdots+c}^{5}+d}_{7}$

Наконец, стрелки над объектами формируют команды

```
\overleftarrow{math}
\overrightarrow{math}
```

```
$\overleftarrow{ABC+\overrightarrow{abc}}$
```
$\overleftarrow{ABC+\overrightarrow{abc}}$

Стиль формулы. При печати формулы в документе символы, размещенные в дробях, под знаком корня или в индексе, уменьшаются. Существует четыре основных стиля форматирования математических выражений, определяемых декларациями

```
\displaystyle
\textstyle
\scriptstyle
\scriptscriptstyle
```

Стиль `\displaystyle` по умолчанию применяется в формулах, автоматически размещаемых на отдельных строках, стиль `\textstyle` — для формул внутри абзаца, стиль `\scriptstyle` — для индексов, стиль `\scriptscriptstyle` — для индексов в индексах. Чтобы изменить размер по умолчанию для некоторого элемента формулы, необходимо применить к нему нужную декларацию.

```
Вряд ли удовлетворит эстета
вид формулы
\[
x+\frac{1}{x+
   \frac{1}{x+
    \frac{1}{x+\frac{1}{x}}
   }
  }
\]
Небольшие усовершенствования приводят к
желаемому результату
\[
x+\frac{1}{x+
   \displaystyle\frac{1}{x+
    \displaystyle\frac{1}{x+
     \displaystyle\frac{1}{x}}
   }
  }
\]
```

Вряд ли удовлетворит эстета вид формулы

$$x+\frac{1}{x+\frac{1}{x+\frac{1}{x+\frac{1}{x}}}}$$

Небольшие усовершенствования приводят к желаемому результату

$$x+\frac{1}{x+\displaystyle\frac{1}{x+\displaystyle\frac{1}{x+\displaystyle\frac{1}{x}}}}$$

Пробелы в математических формулах. Как уже говорилось ранее, в математической моде LaTeX игнорирует все пробелы. Но для создания красивых формул часто необходимо делать отступы разного размера. Для управления горизонтальными пробелами в формулах имеются следующие команды:

`\,`	узкий
`\␣`	средний
`\quad`	широкий
`\qquad`	очень широкий

3.8.3 Матрицы

Для создания матриц в LaTeX имеется процедура

`\begin{array}{`*cols*`} ... \end{array}`

где обязательный аргумент *cols* указывает количество столбцов в матрице. Каждому столбцу аргумента *cols* соответствует одна из букв `l,r, c`, которые отвечают за позиционирование элементов в колонках:

`l` — выравнивание по левой границе колонки,
`r` — выравнивание по правой границе солонки,
`c` — выравнивание по центру колонки.

Элементы столбцов в строке разделяются символом `&`, строки разделяются командой `\\`.

```
\[
\begin{array}{lcr}
a+x-y & b   & 4x \\
x+y   & 2+5 & a+b \\
x     & xz  & -4
\end{array}
\]
```

$$\begin{array}{lcr} a+x-y & b & 4x \\ x+y & 2+5 & a+b \\ x & xz & -4 \end{array}$$

Матрицы часто окружают круглыми или квадратными скобками, определитель матрицы обозначают вертикальными линиями. Подобрать нужный размер таких разделителей позволяют команды `left` и `right`.

```
\[
\left(
\begin{array}{c}
 \left |
 \begin{array}{cc}
  a & b\\ c & d
 \end{array}
 \right| \\
 x \\ y
\end{array}
\right)
\]
```

$$\left(\begin{array}{c} \left| \begin{array}{cc} a & b \\ c & d \end{array} \right| \\ x \\ y \end{array} \right)$$

Процедура `array` вместе с большими разделителями позволяет реализовывать самые разные структуры математических формул.

```
\[
\sigma(x)=\left\{
\begin{array}{rl}
-1, & \mbox{если } x>0\\
 0, & \mbox{если } x=0\\
 1, & \mbox{если } x<0
\end{array}
\right.
\]
```

$$\sigma(x) = \left\{ \begin{array}{rl} -1, & \text{если } x > 0 \\ 0, & \text{если } x = 0 \\ 1, & \text{если } x < 0 \end{array} \right.$$

3.8.4 Многострочные формулы

Для печати систем неравенств или длинных формул, которые не помещаются в одной строке можно использовать процедуры

```
\begin{eqnarray} ... \end{eqnarray}
\begin{eqnarray*} ... \end{eqnarray*}
```

В процедуре `eqnarray` каждая строка автоматически нумеруется, и только это отличает ее от процедуры `eqnarray*`, где нумерация отсутствует. Однако, отменить автоматическую нумерацию каких-то строк формулы поможет команда .

Тело процедуры `eqnarray` состоит из последовательности строк, разделенных командой `\\`, а каждая строка, по умолчанию, делится на три колонки, разделенных символом `&`. Выравнивание элементов в колонках соответствует процедуре `array` с аргументами `rcl`.

```
\begin{eqnarray}
x & = & 21y\\
y & < & a+b+c+ \nonumber \\
&& d-e
\end{eqnarray}
```

$$\begin{aligned} x &= 21y && (3)\\ y &< a+b+c+ \\ &\quad d-e && (4) \end{aligned}$$

Если в многострочной формуле используются большие скобки, масштабируемые командами `left` и `right`, то в каждой строке эти команды должны использоваться попарно. При форматировании длинных формул полезной может оказаться команда `lefteqn`, которая обычно используется в левой колонке. Длину текста в аргументе этой команды LaTeX принимает нулевой. Если при расщеплении формулы на несколько строк новая строка должна начинаться со знака бинарной операции, то перед ним полезно вставить команду `\mbox`.

```
\begin{eqnarray*}
\lefteqn{x+y+z=}\\
& & \left( a+b+2c\right.\\
& & \left. \mbox{}-d+m \right)
\end{eqnarray*}
```

$$\begin{aligned} x+y+z= \\ &(a+b+2c \\ &-d+m) \end{aligned}$$

Уберите команды `\lefteqn` и `\mbox` из предыдущего примера и увидите разницу в отображении формул.

Отметим, что если не делить тело процедуры `eqnarray` на колонки (т.е. не вставлять в строку знак &), то получится обыкновенная многострочная формула, каждая строка которой позиционируется справа.

```
\begin{eqnarray}
x+y+z=1\\
2x-y+3z=4\\
-x+2y=2
\end{eqnarray}
```

$$\begin{aligned} x+y+z &= 1 && (5)\\ 2x-y+3z &= 4 && (6)\\ -x+2y &= 2 && (7) \end{aligned}$$

3.9 Программируем сами

3.9.1 Определение новых команд

Переодическое написание длинных имен команд, повторяющихся конструкций по форматированию текста и т.п. может превратить подготовку печатного документа LaTeX в утомительное занятие. Облегчить жизнь поможет определение новых и/или переопределение уже имеющихся команд с именами, удобными для использования самому автору.

Определить новую команду поможет декларация

`\newcommand{`*cmd*`}{`*def*`}`

Например, если в тексте документа регулярно встречается выражение $\forall\,\varepsilon > 0\ \exists\,\delta > 0$, то постоянно прописывать набор команд LaTeX, его формирующих, весьма трудоемко. Гораздо удобнее было бы пользоваться одной командой, например, `\ed`. Все в ваших руках — определяйте и пользуйтесь

```
\newcommand{\ed}
{$\forall\,\varepsilon>0\
\exists\,\delta>0$}

Функция $f$ иммет в точке $a$ конечный
предел $\iff$ \ed\ т.ч. для любых
$x',\, x'': |x'-a|<\delta,\
|x''-a|<\delta$,
выполнено $|f(x')-f(x'')|<\varepsilon$.
```

Функция f имеет в точке a конечный предел $\iff \forall\,\varepsilon > 0\ \exists\,\delta > 0$ т.ч. для любых x', $x'' : |x' - a| < \delta$, $|x''-a| < \delta$, выполнено $|f(x')-f(x'')| < \varepsilon$.

Для переопределения уже существующей команды необходимо использовать декларацию

`\renewcommand{`*cmd*`}{`*def*`}`

```
\Large Большое
\renewcommand{\Large}{\tiny}
стало \Large маленьким.
```

Большое стало маленьким.

Создавать новые и переопределять существующие команды можно в любом месте текста, но до их первого вызова. Однако, если новоиспеченные команды планируется использовать по всему тексту, то традиционно их определение принято выносить в преамбулу. Область, где сохраняется определение команды, можно явно ограничить фигурными скобками. Имя команды всегда должно начинаться с символа обратного слеша (\) и состоять только из букв (никаких цифр, знаков препинания, пробелов и т.д.).

Новые команды могут иметь до девяти аргументов: количество аргументов задается как необязательная опция декларации `\newcommand`, обращение к аргументам идет по знаку решетка (#) и номеру аргумента

```
\newcommand{\F}[2]
{$#2_{0},\ldots,#2_{#1}$}

Последовательность \F{k}{x}
содержит $k+1$ член.
```

Последовательность $x_0, \ldots, x_k$ содержит $k + 1$ член.

Первый аргумент у новоиспеченной команды можно объявить необязательным. Для этого при определении команды необходимо указать его значение по умолчанию.

```
\newcommand{\F}[2][k]
{$#2_{0},\ldots,#2_{#1}$}

Сравним \F{x} и \F[N]{x}
член.
```

Сравним $x_0, \ldots, x_k$ и $x_0, \ldots, x_N$.

Резюмируя вышесказанное, заключаем что, для определения новых команд, используются две декларации

`\newcommand{`*cmd*`}[`*num*`][`*opt*`]{`*def*`}`
`\renewcommand{`*cmd*`}[`*num*`][`*opt*`]{`*def*`}`

Аргументы декларации имеют следующие назначения:

- *cmd* — имя команды, должно начинаться с символа \, состоять только из последовательности букв и не должно начинаться с `\end`. Для `\newcommand` команда с именем *cmd* не должна быть определена ранее. Напротив, для `\renewcommand` имя *cmd* уже должно быть введено.
- *num* — целое число от 1 до 9, задающее количество аргументов у определяемой команды. По умолчанию новая команда не имеет аргументов.
- *opt* — значение необязательного аргумента. При наличии *opt* аргумент с номером 1 объявляется необязательным и по умолчанию имеет значение *opt*. Все остальные аргументы являются обязательными.
- *def* — текст, который подставляется вместо каждого появления команды *cmd* в исходном файле. Если в *def* находится параметр вида $\#n$, то вместо него подставляется аргумент с номером n.

3.9.2 Теоремы

Книги и статьи по математике, как правило, содержат теоремы и другие теоремоподобные структуры, например, леммы, аксиомы, определения и т.п. Нематематический текст также может состоять из аналогичных структур: правил, законов, принципов и т.д. Конструировать подобные структуры помогает декларация

`\newtheorem{`*enu*`}{`*caption*`}[`*within*`]`

Аргументы декларации имеют следующее значение:

- *enu* — уникальное имя процедуры, для каждого имени создается свой счетчик (т.е. теоремы определяемые одним и тем же именем, будут последовательно нумероваться, согласно своему собственному счетчику);
- *caption* — текст, который должен быть напечатан в качестве заголовка теоремы;
- *within* — имя уже существующего счетчика. Счетчик процедуры *enu* будет автоматически обнуляться при каждом изменении значения *within*

Декларация `\newtheorem` является глобальной, поэтому **должна быть определена в преамбуле документа**.

Теоремоподобная процедура может иметь один необязательный аргумент

`\begin{`*enu*`} [`*text*`]`

Обычно в качестве *text* указывают имя автора теоремы, *text* печатается после номера теоремы.

Все вышесказанное поясним примерами.

```
\newtheorem{Th}{Теорема}
\newtheorem{Ax}{Аксиома}[section]
\newtheorem{St}{Утверждение}[section]

Теорема \ref{Pifagor} принадлежит
Пифагору.
\begin{Th}\label{Pifagor}
 Квадрат гипотенузы равен сумме
 квадратов катетов.
\end{Th}

\begin{Th}[Ферма]\label{Ferma}
 Нет целых чисел $n>2$, x, y, z
таких, что $x^n+y^n=z^n$.
\end{Th}
Доказательство Теоремы \ref{Ferma}
занимает целую книгу.

\begin{Ax}
Параллельные линии не пересекаются.
\end{Ax}

\begin{St}
Под лежачий камень вода не течет.
\end{St}

\begin{St}
Без труда не выловишь и рыбку
из пруда.
\end{St}
```

Теорема 1 принадлежит Пифагору.

Теорема 1. *Квадрат гипотенузы равен сумме квадратов катетов.*

Теорема 2 (Ферма)**.** *Нет целых чисел $n > 2$, x, y, z таких, что $x^n + y^n = z^n$.*

Доказательство Теоремы 2 занимает целую книгу.

Аксиома 3.1. *Параллельные линии не пересекаются.*

Утверждение 3.1. *Под лежачий камень вода не течет.*

Утверждение 3.2. *Без труда не выловишь и рыбку из пруда.*

3.10 Таблицы

Для создания таблиц LaTeX предлагает две процедуры: `tabbing` и `tabular`, главные различия между которыми состоят в следующем:

- Процедуру `tabbing` можно использовать только в текстовом режиме, тогда как `tabular` применима в любой моде.
- Строки таблицы, созданной процедурой `tabbing`, могут переносится на следующую страницу, чего нельзя сказать при использовании `tabular`.
- Таблица, созданная при помощи `tabbing` размещается с новой строки, а таблица формируемая процедурой `tabular` встраивается в текущую позицию.

- Ширина колонок в `tabular` устанавливается автоматически, для `tabbing` необходимо вручную устанавливать точки остановки табуляра.

Вызов процедуры `tabbing` осуществляется командами

```
\begin{tabbing} ...\end{tabbing}
```

Границами колонок служат точки табуляции, которые устанавливаются командой `\=`. Команда `\>` передвигает текст к следующему положению табулятора. Строки разделяются командой `\\`. Следующий код

```
\begin{tabbing}
\hspace{5cm} \= \hspace{4cm} \= \kill
\itshape Произведение \> Жанр \> \itshape Автор \\
Руслан и Людмила \> Роман в стихах  \> А.С. Пушкин \\
Колобок \> Сказка \> не известен \\
Кавказские пленники \> Повесть \> М.Ю. Лермонтов
\end{tabbing}
```

создает таблицу вида

Произведение	Жанр	*Автор*
Песня о буревестнике	Стихи	М.Ю. Лермонтов
Колобок	Сказка	не известен
Кавказский пленник	Повесть	А.С. Пушкин

Команда `\kill` в листинге кода означает, что текущую строку печатать не нужно. Эта строка используется только для установки точек табуляции.

Обратим внимание, что слово "Жанр" напечатано прямым шрифтом, хотя в предыдущей колонке имеется декларация `\itshape`. Это объясняется тем, что команды табуляции одновременно ограничивают область действия любых деклараций точно так же, как фигурные скобки.

Для форматирования сложных таблиц лучше использовать процедуру `tabular`. Эта процедура по синтаксису аналогична процедуре `array` (см. стр. 27), но при этом позволяет рисовать границы таблиц, проводить разделительные вертикальные и горизонтальные линии между элементами.

Чтобы создать таблицу

Романов	Петр	царь
Кутузов	Михаил	генерал
Колмогоров	Андрей	математик

исходный код документа должен содержать следующие строки

```
\begin{center}
\begin{tabular}{|l|c||r|}
\hline
Романов & Петр & царь \\
\hline
Кутузов & Михаил & генерал \\
\hline
Колмогоров & Андрей & математик\\
\hline
\end{tabular}
\end{center}
```

Здесь процедура `center` центрирует таблицу на странице по горизонтали. Сама таблица создана процедурой `tabular`, тело которой обрамляют команды

```
\begin{tabular} ...\end{tabular}
```

Три буквы `l, c, r` в аргументе `{|l|c||r|}` означают, что в таблице должно быть три колонки, причем первая выравнивается по левому краю (флаг `l` соответствует left), вторая размещается по центру (флаг `c` соответствует center), третья выравнивается по правому краю (флаг `r` соответствует right). Символ | означает, что между колонками нужно провести вертикальные линии во всю высоту таблицы. Горизонтальные линии проводит команда `\hline`.

Если указатель на колонку — это одна из букв `l, c, r`, то по умолчанию, ширина колонки определяется ее содержимым, причем следует помнить, что при этом слова одной ячейки не переносятся на следующую строку и команда `\\` не применима. Для формирования многострочных ячеек вместо опций `l, c, r` используется команда `p{`*wd*`}`, которая устанавливает ширину колонки равную *wd*, и при этом если текст не помещается в указанную длину, то он автоматически переносится по словам.

Сформировать таблицу вида

Микроэкономика	наука, изучающая функционирование экономических агентов в ходе их производственной, распределительной, потребительской и обменной деятельности.
Макроэкономика	наука, изучающая функционирование экономики страны в целом (либо её части, отрасли), такие общие процессы и явления как инфляция, безработица, бюджетный дефицит, экономический рост, государственное регулирование и т. п.

поможет следующий код

```
\begin{center}
\begin{tabular}{|l|p{13cm}|} \hline
Микроэкономика & наука, изучающая функционирование экономических агентов
в ходе их производственной, распределительной, потребительской и обменной
деятельности.
\\ \hline
Макроэкономика & наука, изучающая функционирование экономики страны в целом
(либо её части, отрасли), такие общие процессы и явления как инфляция,
безработица, бюджетный дефицит, экономический рост, государственное
регулирование и т. п.
\\
\hline
\end{tabular}
\end{center}
```

Процедура `tabular` обладает набором полезных команд, которые позволяют формировать сложные таблицы. Вот некоторые из них

> `@{`*text*`}` — вставляет *text* между колонками во все строки таблицы. Этот указатель называется @-выражением. Например, чтобы установить пробел в 1 см между двумя колонками, достаточно вставить команду `@{\hspace{1cm}}` между соответствующими указателями колонок.

`\multicolumn{`*n*`}{`*col*`}{`*text*`}` — создает ячейку, состоящую из текста *text*, занимающую *n* колонок таблицы и позиционированную в соответствии с *col*. Аргумент *col* должен содержать в точности одну опцию `r, c, l` и одно или более @-выражений или символ(ы) |.

`\vline` — проводит вертикальную линию на высоту строки.

`\cline{`*i*`-`*j*`}` — проводит горизонтальную линию через колонки с порядковыми номерами от *i* до *j*

Для пояснения работы приведенных команд сформируем таблицу с переменным числом колонок

Диета		
№ стола	**Энергетическая ценность(ккал) мин. – макс.**	**Примечание**
1	1500 – 2000	Показан при язвенной болезни желудка и 12-перстной кишки.
5	1200 – 1700	Показана при болезнях печени, желчного пузыря, желчевыводящих путей без обострений

Листинг программы для такой таблице следующий

```
\begin{center}
\begin{tabular}{|c||r@{\,--\,}l|p{5cm}|} \hline
\multicolumn{4}{|c|}{\bf Диета}
\\ \hline\hline
& \multicolumn{2}{c|}{\parbox{34mm}
{\centering\bf Энергетическая ценность(ккал)}}
&
\\ \cline{2-3}
\bf \No\,стола &  \parbox{17mm}{\hfill\bf мин.} &
\bf макс. &
\multicolumn{1}{c|}{\bf Примечание}
\\ \hline
1 & 1500 & 2000 & Показан при язвенной болезни желудка и 12-перстной кишки.\\
\hline
5 & 1200 & 1700 & Показана при болезнях печени, желчного пузыря, желчевыводящих путей без
\\ \hline
\end{tabular}
\end{center}
```

Здесь процедура `center` размещает таблицу по центру страницы. Таблица формируется процедурой `tabular`, аргумент `{|c||r@{\,--\,}l|p{5cm}|}` показывает, что всего в таблице 4 колонки. Выравнивание текста в первой колонке идет по центру, при этом по всей высоте таблицы слева от первой колонки будет идти одинарная, а справа — двойная вертикальная линия (`|c||`). Содержимое второй колонки будет позиционироваться справа (`r`). Между второй и третьей колонками вставляется среднее тире, обрамленное маленькими пробелами (`@{\,--\,}`). Текст в третьей колонке располагается слева, а сама колонка отделяется от четвертой вертикальной чертой (`l|`). Ширина четвертой колонки равна 5 см, текст в ней будет автоматически переноситься по словам, справа от колонки идет вертикальная черта (`p{5cm}|`).

Заголовок таблицы формирует команда `\multicolumn{4}{|c|}{\bf Диета}` — объединяются все 4 колонки (`{4}`), текст "Диета" центрируется и обрамляется вертикальными линиями (`{|c|}`).

Заголовки колонок условно делятся на две строки. Первая строка имеет вид

```
& \multicolumn{2}{c|}{\parbox{34mm}
{\centering\bf Энергетическая ценность(ккал)}}
&
\\ \cline{2-3}
```

Здесь первый знак амперсента (&) отделяет пустую первую колонку от второй. Вторая и третья колонки таблицы объединяются командой `\multicolumn` в одну, содержимое которой представляет бокс шириной в 34 мм (`\parbox{34mm}`) и содержит текст "Энергетическая ценность(ккал)", который внутри бокса центрируется по горизонтали декларацией `\centering`. Команда `cline{2-3}` проводит горизонтальную линию под 2 и 3 колонками. Вторая строка заголовок колонок формируется командами

```
\bf \No\,стола &  \parbox{17mm}{\hfill\bf мин.} &
\bf макс. &
\multicolumn{1}{c|}{\bf Примечание}
\\ \hline
```

Данная строка состоит из 4 колонок. В первой идет заголовок "№ стола". Вторая колонка сформирована как бокс шириной 17 мм (`\parbox{17mm}`), содержимое бокса "мин." при помощи команды `\hfill` располагается справа. Третья колонка содержит заголовок "макс.". Применение команды `\multicolumn` в четвертой колонке позволяет центрировать заголовок "Примечание", тогда как весь последующий текст будет позиционироваться слева.

Последующие строки таблицы формируются стандартным образом.

Создание сложно структурированных таблиц в LaTeX дело непростое, требующее времени, усидчивости и терпения. На помощь могут прийти специальные команды, описанные в разных стилевых пакетах, поставляемых с дистрибутивом LaTeX. Более того, можно определять свои стили указателей колонок. Все эти возможности остаются на самостоятельное изучение.

3.11 Импортирование графики

Созданию и импортированию графики в документ LaTeX посвящены отдельные книги. В рамках данной работы ограничимся описанием команды `\includegraphics` и ее основных аргументов.

Итак, печатный документ планируется украсить графическими изображениями. Для этого в преамбулу исходного файла необходимо включить вызов пакета, поддерживающего графику в LaTeX. При загрузке любого из графических пакетов указывают драйвер устройства, которое будет использоваться при печати документа. Драйвер объявляют в необязательном аргументе

декларации `\usepackage`. Выберем пакет `graphicx` и драйвер `dvips`. Таким образом в преамбуле должна появиться строка вида

```
\usepackage[dvips]{graphicx}
```

Все графические изображения можно разделить на две категории: растовые и векторные. Растовые изображения состоят из точек (пикселов), векторные — из отрезков линий и закрашенных областей, причем в самом файле векторные изображения хранятся в виде команд, описывающих форму и цвет геометрических объектов. Примерами растовой графики служат файлы форматов `bmp, gif, jpeg, pcd, psd, tiff`, к векторной графике относятся файлы, созданные в формате `ps, eps`. Недостатком любого растового формата является потеря качества при масштабировании изображения. Векторная графика лишена подобного недостатка.

Во избежании ошибок, непредвиденных результатов и плохого качества документа договоримся, что вся импортируемая в LaTeX графика должна быть переведена в векторный формат `ps` или `eps`.

Если рисунок записан в файл *gr-file*, то его можно вставить в печатный документ с помощью команды

`\includegraphics[`*keyval-list*`]{`*gr-file*`}`

Необязательный аргумент *key-list* может содержать список ключей, перечисленных через запятую. Сами ключи записываются в виде равенств, в левой части которых стоит параметр, а в правой — его значение. Список некоторых ключей приведен ниже.

`bb` — задает координаты углов ограничивающего бокса. Значение параметров должно содержать четыре числа, разделенных пробелами. Например, `bb=0 0 5cm 7cm` определяет ограничивающий бокс с координатами нижнего левого угла (0,0) и с координатами верхнего правого (5,7)*число пикселов в сантиметре.

`clip` — отсекает часть рисунка, может принимать два значения `true` или `false`. При `clip=true`, часть рисунка, выходящая за границы видимой области (см. ниже) отсекается. Если параметр `clip` присутствует в списке ключей, но не определен, то предполагается, что `clip=true`.

`viewport` — задает видимую часть рисунка. Значения определяются аналогично `bb`.

`trim` — дает альтернативный метод для выделения видимой части рисунка. Четыре числа задают смещение границ от левой, нижней, правой, верхней границ ограничительного бокса. Например, `trim=1mm 2mm 3mm 4mm` отрезает от рисунка 1мм слева, 2мм снизу, 3мм справа и 4 мм сверху.

`scale` — изменяет размер рисунка по заданному масштабу. Например, `scale=2` увеличивает рисунок вдвое. Пропорции рисунка сохраняются.

`width` — задает ширину рисунка.

`height` — задает высоту рисунка.

`keepaspectio` — сохраняет пропорции рисунка при масштабировании.

`angle` — угол поворота рисунка(в градусах).

`origin` — определяет положение оси вращения рисунка. По умолчанию рисунок вращается относительно точки привязки. Точки вращения изображены на Рис. 1. Наглядность рисунка делает комментарий излишним. Пример использования может быть следующим `origin=c` — устанавливает ось вращения в центре рисунка.

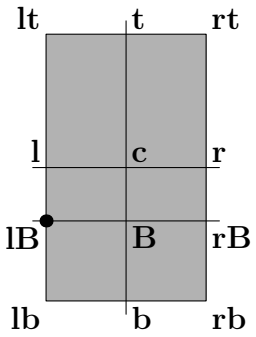
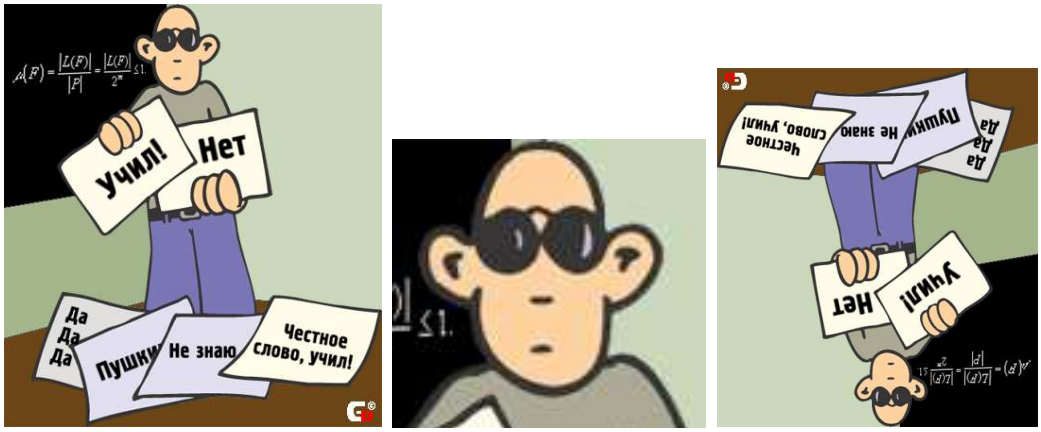


Рис. 1: Точки вращения для ключа `origin`. Точка привязки показана кружочком.

Следующие варианты отображения рисунка из файла `student.eps`



формирует код исходного файла

```
\begin{center}
\includegraphics[width=50mm,keepaspectratio]{student.eps}
\includegraphics[trim= 45mm 120mm 55mm 0,clip]{student.eps}
\includegraphics[scale=0.3, angle=180, origin=c]{student.eps}
{student.eps}
\end{center}
```

Важно помнить, что при повороте и одновременном масштабировании рисунка последовательность параметров `width, height, angle` играет существенную роль. Как показывает следующий пример, при одном и том же размере изображения `beauty.eps`

```
\begin{center}
\includegraphics[height=38mm]{beauty.eps}
\includegraphics[angle=45, height=38mm]{beauty.eps}
\includegraphics[angle=90, height=38mm]{beauty.eps}
\end{center}
```

мы получаем разные результаты

3.12 Плавающие объекты

В отличие от обычного текста, рисунки, а иногда и таблицы нельзя переносить на следующую страницу по частям. Чтобы избежать появления полупустых страниц, непереносимые объекты должны уметь "уплывать" в подходящее место. LaTeX самостоятельно найдет свободное место для объекта, если тот будет помещен в тело одной из процедур

```
\begin{figure}[loc]  ldots \end{figure}
\begin{table}[loc]  ldots \end{table}
```

Необязательный аргумент *loc* уточняет способ размещения плавающего объекта. Он может состоять из последовательности следующих четырех спецификаторов:

`h` — здесь: разрешает размещение плавающего объекта после заполнения текущей строки;

`p` — плавающая страница: разрешает размещение на отдельной странице, содержащей только плавающий объект;

`t` — вверху: выше текста на печатной странице;

`b` — внизу: ниже текста на печатной странице.

Процедура `figure` обычно используется для размещения рисунков, импортируемых командой `\includegraphics`, а `table` — для таблиц, созданных процедурой `tabular`. Однако LaTeX совершенно безразлично, для размещения каких объектов используются эти процедуры. Единственное различие между ними состоит в том, что они по разному подписывают размещаемый объект: Рис. и Таблица. Для каждой процедуры определен свой счетчик, поэтому оперируя командами `\label` и `\ref` в тексте можно ссылаться на те или иные рисунки и таблицы.

Рассмотрим пример текста: *Многие отдали жизнь, чтобы заполучить карту старика Флинта (Рис. 2)*

Рис. 2: Очень ценная карта

которому соответствует код

```
Многие отдали жизнь, чтобы заполучить карту старика
Флинта (Рис. \ref{fig_flint})
\begin{figure}[ht]
\centering \includegraphics[scale=0.5]{flint_map.eps}
\caption{Очень ценная карта}
\label{fig_flint}
\end{figure}
```

Тело процедуры `figure` состоит из четырех команд. Декларация `centering` обеспечивает центрирование рисунка, который импортирует команда `\includegraphics`, причем предполагается, что загружен пакет `graphicx`. Подпись к рисунку создает команда `\caption`. Как правило для изображений подпись располагается под картинкой, поэтому команда `\caption` вызывается после `\includegraphics`. Эта же команда печатает порядковый номер рисунка. Команда `\label` устанавливает метку, по которой можно сослаться на рисунок (с помощью команды `\ref`) в любом месте печатного документа.

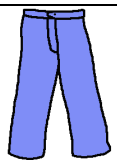
Возможный вариант использования процедуры `table` дает код

```
В нашем магазине вы можете приобрести самую модную
одежду этого сезона (см. Таб. \ref{tab_clothes}).
```

```
\begin{table}[ht]
\caption{Каталог одежды}\label{tab_clothes}
\centering\begin{tabular}[t]{|c|l|l|}\hline
Модель & Название & Цена\\ \hline\hline
\includegraphics[width=2cm, keepaspectratio]{dress.eps}
& платье женское & 300 у.е.\\ \hline
\includegraphics[width=2cm, keepaspectratio]{parka.eps}
& куртка демисезонная & 600 у.е.\\ \hline
\includegraphics[width=2cm, keepaspectratio]{jeans.eps}
& джинсы & 250 у.е.\\ \hline
\includegraphics[width=2cm, keepaspectratio]{shorts.eps}
& шорты летние & 20 у.е.\\
\hline
\end{tabular}
\end{table}
```

Подпись к таблицам принято располагать над таблицей, поэтому команда `\caption` вызывается перед процедурой `tabular`. Результат в печатном документе следующий. *В нашем магазине вы можете приобрести самую модную одежду этого сезона (см. Таб. 4).*

Таблица 4: Каталог одежды

Модель	Название	Цена
	платье женское	300 у.е.
	куртка демисезонная	600 у.е.
	джинсы	250 у.е.
	шорты летние	20 у.е.

3.13 Библиография и цитирование литературы

Реферат, курсовая или дипломная работа, научная или обзорная статья, как правило, содержат список используемой литературы, состоящий из пронумерованных записей, каждая из которых

перечисляет авторов, название, место и время публикации. Существуют определенные правила (ГОСТ) по оформлению библиографии и их необходимо придерживаться.

Современный метод цитирования литературы удачно вписывается в схему организации перекрестных ссылок, реализованную в LaTeX. Создать библиографический список позволяет процедура

`\begin{thebibliography}{`*wlab*`} ... \end{thebibliography}`

Обычно она помещается в конце печатного документа. Аргумент *wlab* служит для создания левой границы списка литературы. LaTeX определяет ширину текста, содержащегося в *wlab*, как если бы тот был напечатан, и сдвигает левую границу библиографии на соответствующее расстояние. Рекомендуется, чтобы ширина текста в *wlab* была равна самому широкому номеру (или самой широкой метке) записей в списке литературы.

Каждая запись в теле процедуры `thebibliography` должна начинаться командой

`\bibitem[`*lab*`]{`*key*`}`

Она генерирует запись, помеченную меткой *lab*. Если опция *lab* пропущена, то запись получает порядковый номер в списке литературы. Обязательный аргумент *key* является ключом, по которому команда

`\cite[`*text*`]{`*key*`}`

печатает номер или метку *lab* соответствующей записи.

```
В книге \cite{Ermolev1976} приведено
полное доказательство теоремы.
...
\begin{thebibliography}{00}
...
\bibitem{Ermolev1976}
Ермольев Ю.М. Методы стохастического
программирования // М.: Наука, 1976. 240
с.
\{thebibliography}
```

В книге [18] приведено полное доказательство теоремы.
...

Список литературы

...

[18] Ермольев Ю.М. Методы стохастического программирования // М.: Наука, 1976. 240 с.

В приведенном примере аргумент `00` процедуры `thebibliography` задает максимальную ширину метки, а именно ширину двузначного номера записи в списке литературы.

В качестве ключа *key* может выступать любая последовательность букв, цифр и знаков пунктуации (кроме запятой). Прописные и строчные буквы, а также русские и латинские буквы одинакового начертания рассматриваются LaTeX как разные символы.

Одной командой `\cite` можно сослаться на несколько записей в списке литературы, для этого ключи записей в аргументе этой команды необходимо перечислять через запятую. Опция *text* служит для уточнения ссылки на источник. Чаще всего в опции *text* указывают номера страниц или параграфов.

```
Смотри \cite{Petrov,Ivanov},
а также \cite[\S5]{Sidorov}.
```

Смотри [2,3], а также [9,§5].

Если необязательный аргумент *lab* присутствует в команде `\bibitem`, то именно он используется в качестве метки записи.

```
В книге \cite{Ermolev1976} приведено
полное доказательство теоремы.
...
\begin{thebibliography}{W:0000}
...
\bibitem[E:1976]{Ermolev1976}
Ермольев Ю.М. Методы стохастического
программирования // М.: Наука, 1976. 240
с.
\{thebibliography}
```

В книге [E:1976] приведено полное доказательство теоремы.
...

Список литературы

...

[E:1976] Ермольев Ю.М. Методы стохастического программирования // М.: Наука, 1976. 240 с.

В приведенном примере запись `E:1976` является меткой.

Заголовок списка литературы зависит от класса документа, его можно легко изменить, переопределив специально предназначенную для этого команду. Для класса документа `article` такой командой является `\refname`. Переопределение осуществляется с помощью команды `\renewcommand`, которая должна быть вызвана до начала процедуры `thebibliography`.

```
\renewcommand{\refname}{Библиография}
\begin{thebibliography}{00}
...
\end{thebibliography}
```

Заметим, что для организации цитирования литературы, как и для любых других видов перекрестного цитирования, LaTeX использует информацию, которую он записал в служебный файл `*.aux` при предыдущей обработке входного файла. Поэтому после внесения изменений в список литературы входной файл должен быть обработан дважды.

3.14 Постскриптум

В настоящем руководстве описаны далеко не все возможности издательской системы LaTeX по созданию стильных, виртуозных в оформлении и приятных для чтения документов. За рамками данного материала остались неосвещенными, например, такие возможности LaTeX, как создание боксов, работа с графикой и цветом, создание больших таблиц, построение сложных математических формул и использование математического алфавита и т.д.

Для более подробного изучения системы LaTeX рекомендуется ознакомиться с материалами из приведенных ниже источников.

[1] Котельников И., П. Чеботаев Издательская система LaTeX 2_ε. – Новосибирск: Изд-во Сибирский хронограф, 1998. – 496 с.

[2] Гуссенс М., Ратц С., Миттельбах Ф. Путеводитель по пакету LaTeX и его графическим расширениям. – М.: Мир: Бином ЛЗ, 2002. – 621 с.

[3] Не очень краткое введение в LaTeX 2$_\varepsilon$ или LaTeX 2$_\varepsilon$ за 84 минуты [Электронный ресурс] // Режим доступа: http://www.nsc.ru/win/docs/TeX/Tobias/lshort2e.html

4 Создание презентаций: пакет `beamer`

Возможности издательской системы LaTeX постоянно расширяются, появляются новые пакеты, которые легко интегрируются в систему и содержат описание дополнительных инструментов, классов, функций и т.п., что позволяет создавать качественные, аккуратно оформленные, приятные для чтения и просмотра документы. Так, например, в системе LaTeX существует несколько специальных пакетов для создания презентаций. В числе таких пакетов Beamer, Prosper, PPower4, PdfSlide и другие. Презентации, созданные в LaTeX ничуть не уступают по оформлению презентациям, созданным с помощью программы PowerPoint, а в случае научно-исследовательских работ, активно использующих математические формулы, таблицы, графики, могут выглядеть значительно лучше своих аналогов из PowerPoint. Более того, нет сомнений в том, что в случае, когда презентация должна быть подготовлена по материалам документа, созданного средствами LaTeX, то целесообразно создавать слайды инструментами этой же системы. В рамках курса будет рассмотрен только один пакет для создания презентаций в LaTeX, это пакет Beamer.

Пакет Beamer, из-за своих богатых возможностей, получил очень широкое распространение, особенно в Америке, и по-существу, стал де-факто стандартом для создания презентаций в LaTeX. В свободном доступе сети Интернет легко можно найти и инсталяционные файлы и подробную документацию пакета.

Как и для любого документа, созданного в системе LaTeX, все содержимое презентации формируется в виде входного файла с помощью команд LaTeX и Beamer. Тело файла состоит из преамбулы и основной части. В преамбуле, помимо обычных настроек, указывют стиль презентации, ее цветовое оформление. Основная часть состоит непосредственно из слайдов презентации.

4.1 Преамбула входного файла презентации

Преамбула начинается с команды

```
\documentclass[12pt, roman]{beamer}
```

Напомним, что в квадратных скобках указываются необязательные аргументы команды, которых вообще может не быть. Здесь, аргументы `[12pt, roman]` задают основной шрифт документа roman и его размер 12pt. По умолчанию используется шрифт san serif размера 11pt. Отметим, что размер виртуальной страницы в классе beamer составляет 128×96 мм, так что для 11pt — это достаточно большой размер шрифта.

Общую структуру и цветовое оформление презентации определяют пять команд

```
\usetheme{ ... }
\usecolortheme{ ... }
\usefonttheme{ ... }
\useoutertheme{ ... }
\useinnertheme{ ... }
```

Аргумент команды `\usetheme` определяет общий вид оформления презентации и каждого ее элемента в отдельности: цвет фона, шрифтов заголовков и основного текста, стили и маркеры списков, стиль и цвет кнопок, блоков и т.п. В комплект beamer входит много предопределенных тем, но можно создавать и свои. Вот список некоторых уже зарезервированных тем: default, Pittsburgh, Luebeck, CambridgeUS, Warsaw, Copenhagen, Malmoe, Bergen, Madrid, Berkeley, Frankfurt. Пример использования

```
\usetheme{CambridgeUS}
```

Узнать, какие темы доступны для подключения можно просмотретв список файлов с названиями `beamertheme`<*name*>`.sty`, где *name* и есть имя аргумента, который можно указать команде `\usetheme`. Файлы с такими названиями находятся в директории, где установлен пакет beamer.

Для каждой темы можно изменить четыре типа объектов: цветовое оформление — команда `\usecolortheme`, свойства шрифтов — команда `\usefonttheme`, способы выравнивания элементов внутри каждого слайда — команда `\useinnertheme` и общий вид внешнего обромления слайдов (верхний и нижний колонтитулы, боковые рамки, расположение логотипа, вид заголовков и т.п.) Как и для `\usetheme`, каждая из приведенных команд имеет предопределенный список аргументов. Определить доступные аргументы можно из имеющегося в установленной системе списка стиливых файлов:

для команды `\usecolortheme` это файлы с названием `beamercolortheme`<*name*>`.sty`;
для команды `\usefonttheme` — `beamerfonttheme`<*name*>`.sty`;
для команды `\useinnertheme` — `beamerinnertheme`<*name*>`.sty`;
для команды `\useoutertheme` — `beamerinnertheme`<*name*>`.sty`.

Если оформление презентации по теме, указанной в аргументе команды `\usetheme` полностью устраивает пользователя, то нет смысла применять остальные четыре команды. Если же хочется создать уникальную презентацию и продемонстрировать свое чувство прекрасного, то поле для экспериментов очень обширно.

Продолжают преамбулу привычные настройки кодировки и подключение необходимых пакетов, например,

```
\usepackage[koi8-r]{inputenc}
\usepackage[russian]{babel}
\usepackage{amsmath,amsthm,amsfonts}
```

Для класса beamer пакеты `hyperref`, `color` и `graphicx` подключаются автоматически.

Для формирования титульного слайда используют команды

```
\title{ ... }
\subtitle{ ... }
\author{ ... }
\institute { ... }
\date{ ...}
```

Отображение на слайдах логотипа организации, которую представляет докладчик, придает солидность и официальность презентации. Логотип подключается при помощи команды `\logo{ ... }`, например,

```
\logo{\includegraphics{pi.eps}}
```

4.2 Формирование слайдов

Слайды презентации формируются последовательно в основной части исходного файла, то есть внутри блока `\begin{document}` `\end{document}`.

Каждый слайд определяется либо процедурой `frame`, либо декларацией `\frame`:

```
\begin{frame}
...
\end{frame}
```

```
\frame{
...
}
```

Например, первый слайд с названием доклада, авторами и прочей информацией, определенной соответствующими командами в преамбуле, можно создать командой

```
\frame{ \maketitle }
```

Заголовок слайда формируется командой

5 Язык численной математики GNU Octave

GNU Octave — это свободно распространяемый язык программирования высокого уровня, ориентированный на проведение численных расчетов, и по сути являющийся альтернативой коммерческому пакету MatLab . Octave обладает богатым инструментарием для решения задач линейной алгебры, нахождения корней нелинейных уравнений, решения дифференциальных уравнений, вычисления интегралов, решения линейных и нелинейных оптимизационных задач, построения графиков и т.д. Только некоторые из этих возможностей Octave будут описаны в настоящем руководстве.

5.1 Запуск пакета GNU Octave

Все управление процессом вычислений в пакете GNU Octave осуществляется через командную строку. Также можно заранее подготовить на языке Octave скрипт-файл для последующего его исполнения.

Для того, чтобы запустить пакет GNU Octave необходимо в командной строке исполнить команду `octave`. Программа запустится в текущей консоли, при этом сперва будет выдано длинное сообщение о версии, авторах и прочая информация о пакете, а затем приглашение командной строки к работе с Octave

```
octave:1>
```

Если при запуске Octave дать команду `octave -q`, то начальное сообщение выводиться не будет.

Выйти из Octave можно либо нажав комбинацию клавиш <Ctrl>+<D>, либо набрав в командной строке Octave одну из команд `exit` или `quit`

```
octave:1> exit
```

Командой `exit` также можно закончить выполение любой программы, написанной на языке Octave.

Работая из командной строки Octave каждая введенная команда исполняется сразу после нажатия клавиши <Enter>. Рассмотрим простой пример по созданию матрицы

$$A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 7 \end{pmatrix}.$$

Любую матрицу или вектор с заданными элементами в Octave можно создать путем перечисления этих элементов в квадратных скобках (`[ ]`), разделяя столбцы пробелом или запятой, строки — знаком "точка с запятой"

```
octave:1> A=[1 3 5; 2 4 6]
```

После нажатия клавиши <Enter> на экране появится результат выполнения команды

```
A =

   1   3   5
   2   4   6
```

Теперь в памяти Octave есть матрица A и далее с ней можно производить любые доступные операции, например, транспонировать: операция "штрих" (`'`) после имени матрицы

```
octave:2> A'
```

Результат выполнения автоматически будет сохранен в переменную `ans` и выведен на экран

```
ans =

   1   2
   3   4
   5   6
```

Везде далее после введения команды Octave будем печатать результат ее исполнения.

Можно создать новую матрицу $B = A^T$ при помощи операции присвоения (`=`)

```
octave:3> B=A'
B =

   1   2
   3   4
   5   6
```

Далее можно оперировать уже двумя матрицами A и B.

Как уже было отмечено, работа через командную строку Octave — это один из способов проводить процесс вычислений. Альтернативой является создание скрипт-файла и последующая его обработка пакетом GNU Octave.

Опишем процесс создания матрицы A и B в отдельном файле, например, `matrix.oc` [2], который должен содержать следующие строки:

```
A=[1 3 5; 2 4 6]
B=A'
```

После сохранения файла `matrix.oc` из командной строки консоли можно запустить его на исполнение командой

```
octave matrix.oc
```

Результат работы будет выведен на экран.

В ходе реализации более сложных вычислений, как правило, существуют промежуточные операции, результат выполнения которых выводить нет необходимости. Чтобы отменить вывод следует ставить знак "точка с запятой" после соответствующей команды. Например, если поставить "точку с запятой" после команды создания матрицы A

```
A=[1 3 5; 2 4 6];
B=A'
```

то по мере обработки скрипт-файла `matrix.oc` будет выведена только матрица B.

Код программы может сопровождаться комментариями. Octave игнорирует строки, помеченные знаками "решетка" (`#`) или "процент" (`%`).

[2] Несмотря на то, что расширение файлов для UNIX-систем не является обязательным, для скрипт-файлов Octave будем писать расширение `.oc`

```
A=[1 3 5; 2 4 6];
# Транспонирование матрицы A
B=A'
```

Далее возможности языка покажем на примерах, реализованных через командную строку Octave, однако все описанные ниже команды можно записать в отдельный файл `*.oc` и обработать пакетом как самостоятельную программу.

5.2 Простые вычисления

Средствами языка Octave можно выполнить любые арифметические и алгебраические операции, доступные самому продвинутому калькулятору. Операндами могут быть как вещественные, так и комплексные числа.

Вещественные числа определяются в виде целого значения или конечной десятичной дроби, при этом дробная часть числа отделяется точкой, например `3.65`. Допустима экспонециальная форма записи вещественных чисел, например `1e-4`, что эквивалентно значению 0.0001, или `1e4`, что эквивалентно значению 10000. В вычислениях также можно использовать иррациональные константы $\pi = 3.146...$ команда `pi` и $e = 2.718..$ команда `e`.

Любое комплексное число вводится как формальная сумма x `+ iy`, где x и y — вещественные числа, `i` — мнимая единица, то есть число, удовлетворяющее уравнению $i^2 = -1$.

```
octave:16> c = 1-3e-2i
c =  1.000000 - 0.030000i
```

Операции над комплексными числами осуществляются согласно общих правил и отдельно в настоящем руководстве рассматриваться не будут.

Применяя операции сложения (`+`), вычитания (`-`), умножения (`*`), деления (`/`), возведения в степень (`^`) можно вычислять самые разнообразные математические выражения

```
octave:1> (1+2^3-4)/5
ans =  1
```

В Octave легко вычислить квадратный корень, логарифмические и тригонометрические функции, имеются функции округления, взятия минимума и максимума и т.д.Список некоторых(!) функций приведен в таблице 5. Аргумент x для тригонометрических функций задается в радианах.

Напомним, что

$$\mathrm{sign}(x) = \begin{cases} 1, & \text{если } x > 0; \\ 0, & \text{если } x = 0; \\ -1, & \text{если } x < 0. \end{cases}$$

Приведем некоторые примеры вычислений:

1) вычислить синус угла в 30 градусов

```
octave:2> sin(30*pi/180)
ans =  0.50000
```

2) найти наибольший общий делитель чисел 24, 86, 144

Таблица 5: Алгебраические функции Octave

`exp(x)`	вычисление функции e^x	`sign(x)`	сигнум-функция от x
`pow2(x)`	вычисление функции 2^x	`abs(x)`	абсолютное значение x
`pow2(f,e)`	вычисление функции $f \cdot 2^e$	`min(x,y)`	минимум из x и y
`log(x)`	натуральный логарифм x	`max(x,y)`	максимум из x и y
`log10(x)`	десятичный логарифм x	`round(x)`	округление x до ближайшего целого
`sqrt(x)`	квадратный корень x	`ceil(x)`	округление x до целого сверху
`sin(x)`	синус x	`floor(x)`	округление x до целого снизу
`cos(x)`	косинус x	`lcm(x,...)`	наименьшее общее кратное $x, \ldots$
`tan(x)`	тангенс x	`gcd(x,...)`	наибольший общий делитель $x, \ldots$
`cot(x)`	котангенс x	`rem(x,y)`	остаток от деления x на y

```
octave:4> gcd(24,84,144)
ans =  12
```

3) вычислить функцию $\ln 10 + 2^5 \cdot 13.7$

```
octave:5> log(10)+pow2(13.7,5)
ans =  440.70
```

5.3 Функции и переменные

Как и в любом языке программирования в Octave есть понятия переменной и функции, создавать которые намного проще, чем, к примеру, в таких языках как C или Pascal.

Переменные в Octave начинают существовать, как только им было присвоено значение. Тип переменной также определяется присвоенным значением. Одномерные и двумерные массивы переменных определяются как вектора и матрицы через квадратные скобки. Требования к именам переменных стандартные, как для многих языков: имя может состоять из букв латинского алфавита, цифр (везде, кроме первого символа имени) и символа нижнего подчеркивания.

```
octave:1> x1=2
x1 =  2
octave:2> x2=3.7
x2 =  3.7000
octave:3> v=[2 4.6 1]
v =

   2.0000   4.6000   1.0000
octave:4> A = [1 3 4; 2 4 6]
A =

   1   3   4
   2   4   6
```

В приведенном примере показано, как создаются целочисленная переменная $x1$, вещественная переменная $x2$, одномерный массив чисел v и двумерный массив чисел A.

Манипулировать переменной можно только после ее определения. Не вызовет ошибки присвоение значения другого типа существующей переменной.

```
octave:5> x1=v
x1 =

 2.0000  4.6000  1.0000
```

С переменными тесно связано такое понятие, как функция. Даже при всем многообразии встроенных в Octave функций на практике их может оказаться недостаточно. Отсюда возникает необходимость создавать новые функции для решения конкретных задач пользователя.

Как правило, новую функцию создают либо в отдельном файле, либо в скрипт-файле Octave до первого ее вызова. Если предполагается использовать пользовательскую функцию в разных скрипт-файлах, то, конечно, предпочтительно создать ее в отдельном файле. В GNU Octave файлы с функциями имеют расширение `.m` и загружаются автоматически. Имя файла должно строго совпадать с именем функции.

Общий синтаксис создания новой функции следующий

`function` [*res1, res2, ..., resM*] = *имя_функции* (*arg1, arg2, ..., argN*)
тело функции
`endfunction`

Начинает описание ключевое слово `function`, затем в квадратных скобках перечисляются выходные параметры функции *res1, res2, ..., resM*. Если выходной параметр один, то квадратные скобки можно опустить. Если выходных параметров нет, то после ключевого слова `function` сразу указывается имя функции. Требования к составлению имени функции аналогичны требованиям к именам переменных. Как правило, функция имеет один или несколько входных параметров *arg1, arg2, ..., argN*, которые перечисляются после имени функции в круглых скобках через запятую. Тело функции состоит из команд Octave. Заканчивает описание функции ключевое слово `endfunction`.

Все входные *arg1, arg2, ..., argN* и выходные *res1, res2, ..., resM* параметры в описании функции, а также все переменные, созданные в ее теле, являются локальными для данной функции.

В Octave нет необходимости подгружать описание функции из файла перед ее использованием.

Напишем функцию решения квадратного уравнения $ax^2 + bx + c = 0$

```
function [x1,x2] = quad_eq(a,b,c)
 D = sqrt(b^2-4*a*c);
 x1 = (-b-D)/(2*a);
 x2 = (-b+D)/(2*a);
endfunction
```

Заметим, что при вычислении дискриминанта опущена проверка на неотрицательность подкоренного выражения. Это необязательно, поскольку Octave умеет работать как с вещественными, так и с комплексными числами.

Сохраним описание функции в файле `quad_eq.m`. Теперь ей можно воспользоваться при работе с Octave

```
octave:6> a=2; b=3; c=-2;
octave:7> [y1,y2]=quad_eq(a,b,c)
y1 = -2
y2 = 0.50000
```

Вызов функции осуществляется по имени, количество входных и выходных параметров, указанных при вызове, должно точно совпадать с их количеством в описании функции. Имена передаваемых и получаемых параметров при вызове функции могут не совпадать с именами, указанными в описании.

что когда Octave встречает функцию, определенную пользователем, она сперва пытается найти ее в текущей директории, т.е. найти файл с расширением `*.m` и именем, совпадающим с именем функции

5.4 Матрично-векторные вычисления

В настоящем разделе покажем, как средствами Octave легко и быстро можно создавать случайные и специальные матрицы любой размерности и производить над ними разные арифметические и алгебраические действия.

5.4.1 Создание матриц

Чтобы создать в Octave матрицу или вектор с заданными элементами нужно просто перечислить эти элементы в квадратных скобках, разделяя строки знаком "точка с запятой", столбцы запятой или пробелом

```
octave:1> A = [ 2 4 5; 6 4 7; 6 1 8]
A =

   2   4   5
   6   4   7
   6   1   8
octave:2> v = [ 3 6.4 8]
v =

   3.0000   6.4000   8.0000

octave:3> w=[5 ; 7 ; 8]
w =

   5
   7
   8
```

Функция `rand` позволяет генерировать случайные числа из нтервала (0,1). Если функция вызывается без аргументов, то будет создано случайное число.

```
octave:5> r=rand
r =  0.67452
```

Если функция имеет один аргумент `rand(m)`, то будет создана квадратная матрица размерности $m \times m$.

```
octave:6> A=rand(5)
A =
```

```
   0.217300   0.272742   0.439678   0.040161   0.550001
   0.620170   0.379312   0.416012   0.295613   0.608457
   0.387311   0.649955   0.808548   0.166677   0.259772
   0.411485   0.903623   0.204110   0.443964   0.696127
   0.994622   0.598606   0.257824   0.637861   0.969088
```

Если передать два аргумента `rand(m,n)`, то будет создана матрица размерности $m \times n$

```
octave:7> B=rand(4,6)
B =

   0.1410003   0.6681751   0.9649029   0.0092332   0.9570230   0.7681693
   0.1151950   0.0958126   0.2708919   0.4965380   0.3816898   0.2960198
   0.3417755   0.1450904   0.1535524   0.5251388   0.9805246   0.2559105
   0.0609999   0.4968790   0.0741969   0.5419970   0.8560731   0.6810124
```

Если один из аргументов равен 1, то будет создан вектор случайных элементов

```
octave:8> v=rand(1,3)
v =

   0.294654   0.492241   0.079515
octave:9> w=rand(3,1)
w =

   0.767195
   0.027604
   0.159826
```

Аналогично `rand` вызывается функция `randn`, которая генерирует случайные числа, имеющие нормальное распределение

```
octave:10> r = randn
r = -0.41171
octave:11> B = randn(4,6)
B =

  -0.73689   0.46540  -0.72798  -1.64099  -1.11487  -0.30241
   0.53960  -1.54595   0.96346  -0.51525   0.46718  -0.54186
   0.19417  -0.35375  -2.79121  -0.50133   0.19464  -1.13815
   0.14187   2.23729  -0.92617   0.27607   2.62902  -0.25009
octave:12> v=randn(1,3)
v =

  -0.984558   0.070915  -0.781345
```

В теории линейной алгебры выделяют специальные виды матриц, например, единичная, диагональная, блочная или нулевая. Создать такие матрицы в Octave позволяют следующие функции:

- `eye(`m`)`, `eye(`m`,`n`)`
  Функция генерирует единичную матрицу (с одним аргументом — квадратная матрица размерности $m \times m$, с двумя аргументами — матрица размерности $m \times n$).

```
octave:16> I = eye(3)
I =

   1   0   0
   0   1   0
   0   0   1

octave:17> I = eye(3,5)
I =

   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
```

- `ones(`m`)`, `ones(`m`,`n`)`
  Функция генерирует матрицу с элементами 1 (количество аргументов трактуется аналогично `eye`.

```
octave:18>  = ones(3)
O =

   1   1   1
   1   1   1
   1   1   1
octave:19> e = ones(1,5)
e =

   1   1   1   1   1
```

- `zeros(`m`), zeros(`m`,`n`)`
  Функция генерирует нулевую матрицу (количество аргументов трактуется аналогично `eye`.

```
octave:29> Z = zeros(3)
Z =

   0   0   0
   0   0   0
   0   0   0

octave:30> z = zeros(1,3)
z =

  0   0   0
```

- `diag(`v`, `k`)`
  Функция генерирует диагональную матрицу с элементами вектора v на диагонали k. Аргумент k выступает в качестве опции: $k = 0$ означает главную диагональ, при $k > 0$ элементы вектора v ставятся на k-ую диагональ выше, а при $k < 0$ на k-ую диагональ ниже главной диагонали. По умолчанию $k = 0$. Размер матрицы определяется в соответствии с аргументами v и k.

```
octave:36> D = diag([1 2 3 4 5])
D =

   1   0   0   0   0
   0   2   0   0   0
   0   0   3   0   0
   0   0   0   4   0
   0   0   0   0   5

octave:37> D = diag([1 2],2)
D =

   0   0   1   0
   0   0   0   2
   0   0   0   0
   0   0   0   0

octave:38> D = diag([1 2],-1)
D =

    0   0   0
    1   0   0
    0   2   0
```

  Отметим еще одно полезное для практического использования свойство функции `diag`. Если первым аргументом v является матрица, то функция возвращает вектор-столбец, состоящий из элементов, стоящих на диагонали маркируемой опцией k.

```
octave:16> A = ceil(rand(3)*100)
A =

    52   100    61
    71    98    66
    41    91    41

octave:17> diag(A,0)
ans =

   52
   98
```

```
   41

octave:18> diag(A,-1)
ans =

   71
   91
```

- `repmat(`A`, `m`)`, `repmat(`A`, `m`, `n`)`
  Функция генерирует блочную матрицу размерности $m \times m$ ($m \times n$), каждым элементом которой является подматрица A.

```
octave:39> B = repmat(eye(2),2,3)
B =

   1   0   1   0   1   0
   0   1   0   1   0   1
   1   0   1   0   1   0
   0   1   0   1   0   1
```

Полезной в вычислениях может оказаться функция `randperm(n)` генерации целочисленного вектора, элементы которого представляют случайную перестановку чисел от 1 до n

```
octave:13> p=randperm(6)
p =

   6   1   4   3   2   5
```

Сгенерировать вектор индексов от M до N позволяет команда `[M:N]`

```
octave:14> q=[13:23]
q =

   13   14   15   16   17   18   19   20   21   22   23
```

Диапазон значений от M до N с шагом *step* задается командой `[it M:`*step*`:N]`. При положительном шаге для корректной работы данной команды должно быть выполнено условие $M \leq N$, при отрицательном шаге — $M \geq N$.

```
octave:11> q = 13:2:23
q =

  13  15  17  19  21  23

octave:12> q = 23:-2:13
q =

  23  21  19  17  15  13
```

Команда `linspace(`*base, limit, n)* возвращает вектор-строку из n элементов, равномерно распределенных на отрезке $[base, limit]$ числовой прямой.

```
octave:15> v=linspace(0, 1, 8)
v =

   0.00000   0.14286   0.28571   0.42857   0.57143   0.71429   0.85714   1.00000
```

Число элементов n должно быть больше 1. По умолчанию $n = 100$. Результирующий вектор включает элементы *base* и *limit*. Если *base* больше *limit*, то n элементов упорядочиваются по убыванию.

```
octave:16> v=linspace(1, 0, 8)
v =

   1.00000   0.85714   0.71429   0.57143   0.42857   0.28571   0.14286   0.00000
```

И наконец, в Octave можно определить пустую матрицу, для этого используют команду `[]`

```
octave:17> A=[]
A = [](0x0)
```

или вызывают одну из вышеописанных функций (например, `rand, zeros, ones` и т.д.), указав при этом хотя бы одну из размерностей равной 0.

```
octave:21> A=rand(0,0)
A = [](0x0)
octave:22> A=rand(3,0)
A = [](3x0)
octave:23> A=rand(0,3)
A = [](0x3)
```

Пустые матрицы удобно использовать, когда необходимо удалить из заданной матрицы некоторые строки или столбцы.

```
octave:5> A = rand(3,7)
A =

   0.562329   0.670663   0.046794   0.802909   0.019110   0.906351   0.299616
   0.217606   0.696152   0.573524   0.328735   0.355639   0.362626   0.475531
   0.680574   0.578710   0.254646   0.484237   0.080157   0.442885   0.125110

octave:6> A(:,1:2:7)=[]
A =

   0.67066   0.80291   0.90635
   0.69615   0.32874   0.36263
   0.57871   0.48424   0.44288
```

5.4.2 Действия над матрицами

Прежде всего научимся определять размерность матриц, с которыми предстоит работать, поскольку многие бинарные операции предполагают соблюдения определенной согласованности между размерностями своих операндов.

Узнать число строк и столбцов данной матрицы позволяет функция `size`. Аргументом функции является матрица или вектор, первый возвращаемый параметр определяет число строк, второй — число столбцов.

```
octave:43> A = rand(10,60);
octave:44> [rowA, colA]=size(A)
rowA =  10
colA =  60
octave:45> v = ones(1,100);
octave:46> size_v = size(v)
size_v =

     1  100
```

Если интересует только количество строк матрицы, то вместо `size` можно использовать функцию `rows`.

```
octave:49> rows(A)
ans =  10
octave:50> rows(v)
ans =  1
```

А если надо определить только количество столбцов, то используется функция `columns`.

```
octave:51> columns(A)
ans =  60
octave:52> columns(v)
ans =  100
```

Сложение (+), вычитание (-), умножение (*) матриц в Octave основано на тех же правилах, что и в линейной алгебре. Складывать и вычитать можно только матрицы одинаковой размерности.

```
octave:54> A = rand(2,3); B=randn(2,3);
octave:55> A+B
ans =

   9.7920e-04   2.4098e+00  -7.4398e-01
  -2.0905e+00   2.7025e+00   2.1091e-01

octave:56> A-B
ans =

   0.36546  -0.94482   1.32552
   2.13113  -0.97702   0.53200
```

При умножении матриц A на B число столбцов у A должно быть равно числу строк у B.

```
octave:57> A = rand(2,3); B=randn(3,2);
octave:58> A*B
ans =

   1.43483   0.18918
   0.77856   0.26710
```

Результатом умножение матрицы $A = (a_{ij})_{m\times n}$ на скаляр c является матрица $B = (c \cdot a_{ij})_{m\times n}$.

```
octave:3> A = [1 2 3; 4 5 6];
octave:4> B = A*10
B =

   10   20   30
   40   50   60
```

Не вызовет ошибки, если применить бинарные операции сложения, вычитания или деления к матрице и скалярной величине, при этом каждая операция будет выполнятся поэлементно.

```
octave:2> A+10
ans =

   11   12   13
   14   15   16

octave:3> A-10
ans =

  -9  -8  -7
  -6  -5  -4

octave:4> 10-A
ans =

   9   8   7
   6   5   4

octave:5> A/10
ans =

   0.10000   0.20000   0.30000
   0.40000   0.50000   0.60000
```

Транспонировать матрицу позволяет оператор ’ (штрих).

```
octave:5> A'
ans =

   1   4
   2   5
   3   6
```

Помимо упомянутых выше операций в Octave есть возможность выполнять поэлементное умножение (`.*`) и деление (`./`) матриц (при этом матрицы должны быть одной размерности).

```
octave:59> A = rand(2,3); B=randn(2,3);
octave:60> A.*B
ans =

   0.7426443   0.0971287  -0.0018915
   0.4088991  -0.2743542  -0.1120034

octave:61> A./B
ans =

   1.2119e+00   1.6707e+00  -7.4780e-04
   1.3371e+00  -2.3146e-01  -1.7738e+00
```

Для квадратных матриц имеется операция поэлементного возведения в степень (`.^`)

```
octave:62> A=randn(3);
octave:63> A.^3
ans =

   2.2750344   0.0389019  -8.4121279
  -0.0784881   0.0046370   1.2989715
   0.4925376  -1.7890019   1.7549553
```

Кроме того, большинство из функций, приведенных в таблице 5 (стр. 51) в качестве аргумента могут получать не только скалярную величину, но также матрицу или вектор, при этом вычисление функции проводится поэлементно.

```
octave:53> sqrt(A)
ans =

   3.1623   5.9161
   5.0000   3.8730
```

Отметим некоторые особенности работы функций вычисления минимального (`min(`x`)`) и максимального (`max(`x`)`) элементов. Для аргумента вектора данные функции возвращают два скалярных параметра, первый — это значение минимального (максимального) элемента вектора x, второй — это номер компоненты, на которой достигается минимум (максимум).

```
octave:19> v = randperm(8)
v =

   6   8   3   1   5   2   7   4

octave:20> [min_v,inx_min]=min(v)
min_v =  1
inx_min =  4
```

```
octave:21> [max_v,inx_max]=max(v)
max_v =  8
inx_max =  2
```

Если функции `min` и `max` в качестве аргумента получают матрицу, то они возвращают два векторных параметра, первый — это вектор минимумов в каждом столбце матрицы x, второй — вектор номеров строк, где достигается минимум в соответствующем столбце.

```
octave:23> A = round(rand(2,5)*100)
A =

   96   18   17   74    6
   60   21   82    5    4

octave:24> [min_A,inx_min]=min(A)
min_A =

   60   18   17    5    4

inx_min =

   2   1   1   2   2

octave:25> [max_A,inx_max]=max(A)
max_A =

   96   21   82   74    6

inx_max =

   1   2   2   1   1
```

Если результат работы функций `max` и `min` присваивать только одному параметру, то будут получены только минимальные значения

```
octave:31> max_v=max(v)
max_v =  8
octave:32> min_A=min(A)
min_A =

   60   18   17    5    4
```

Для матриц и векторов список функций таблицы 5 можно дополнить функциями поэлементного суммирования и умножения компонент вектора.

- `sum`(x)
  Функция поэлементного сложения компонент вектора. Если x — вектор, то возвращает сумму всех компонент x.

```
octave:26> x = randperm(4)
x =

   4   3   1   2

octave:27> sum(x)
ans =  10
```

Если x — матрица, то возвращает вектор сумм компонент каждого столбца x.

```
octave:28> A = [1 2 3 4; 5 4 7 8];
octave:29> sum(A)
ans =

    6    6   10   12
```

- `prod`(x)
  Функция поэлементного умножения компонент вектора. Принцип работы аналогичен функции `sum`.

```
octave:32> prod(x)
ans =  24
octave:33> prod(A)
ans =

    5    8   21   32
```

- `cumsum`(x)
  Функция кумулятивного сложения компонент вектора. Если x — вектор, то возвращает вектор каждая i-ая компонента которого равна сумме первых i компонент вектора x.

```
octave:37> cumsum(x)
ans =

    4    7    8   10
```

Если x — матрица, то возвращает матрицу, каждый столбец которой представляет кумулятивную сумму компонент соответствующего столбца x.

```
octave:38> cumsum(A)
ans =

    1    2    3    4
    6    6   10   12
```

- `cumprod`(x)
  Функция кумулятивного умножения компонент вектора. Принцип работы аналогичен функции `cumsum`.

```
octave:39> cumprod(x)
ans =

    4   12   12   24

octave:40> cumprod(A)
ans =

    1    2    3    4
    5    8   21   32
```

- `sumsq`(x)
  Функция вычисления суммы квадратов. Принцип работы аналогичен функции `sum`.

```
octave:41> sumsq(x)
ans =  30
octave:42> sumsq(A)
ans =

   26   20   58   80
```

5.4.3 Алгебра матриц

Одним из объектов изучения в дисциплине линейная алгебра является анализ системы линейных уравнений $Ax = b$ и поиск ее решения.

Для квадратной матрицы A можно выписать аналитическое решение системы $x = A^{-1}b$, где A^{-1} – обратная к A матрица. Для обращения матриц в Octave существует функция `inv`, аргументом которой должна быть только квадратная матрица.

```
octave:1> A = rand(3);
 octave:2> invA = inv(A)
 invA =

    2.69454  -1.52685   0.47050
   -1.05052   3.11871  -1.29602
   -1.55241   0.87150   1.12319

 octave:3> A*invA
 ans =

    1.00000  -0.00000  -0.00000
    0.00000   1.00000  -0.00000
    0.00000  -0.00000   1.00000
```

Таким образом зная значение вектора правых частей b, средствами Octave можно просто определить неизвестный вектор x

```
octave:4> b = rand(3,1);
```

```
octave:5> x = inv(A)*b
x =

   0.522555
  -0.015841
   0.614833
```

В правой части системы линейных уравнений может стоять матрица b, способ решения такой системы не меняется, главное соблюдать правила согласования размерностей A и b — число строк данных матриц должно быть одинаковым.

```
octave:6> b=rand(3,2);
octave:7> x = inv(A)*b
x =

   1.05918   0.66053
   1.20880   0.86253
   0.17613   0.35082

octave:8> A*x-b
ans =

  -1.1102e-16   0.0000e+00
   0.0000e+00  -1.1102e-16
  -1.1102e-16   0.0000e+00
```

Вместо функции обращения матрицы для решения системы $Ax = b$ можно использовать оператор матричного деления \

```
octave:9> x=A\b
x =

   1.05918   0.66053
   1.20880   0.86253
   0.17613   0.35082
```

Как известно из курса линейной алгебры обратные матрицы существуют только для невырожденных матриц, то есть для матриц, определитель которых отличен от нуля. Поэтому перед тем, как вычислить значения переменных x хорошо бы сперва проверить матрицу A на вырожденность. Функция `det` возвращает определитель своего аргумента — квадратной матрицы

```
octave:10> det(A)
ans =  0.10575
```

При исследовании систем линейных уравнений с произвольной матрицей A размерности $m \times n$ используют такое понятие как ранг матрицы (наибольший из порядков ненулевых миноров матрицы). Функция `rank` возвращает ранг своего аргумента.

```
octave:11> rank(A)
ans =  3
```

5.4.4 Поиск элементов и проверка условий

В ситуациях, когда необходимо определить удовлетворяют ли элементы данной матрицы некоторому заданному условию, могут пригодиться имеющиеся в арсенале Octave функции. Такие функции, как правило, в качестве выходного параметра возвращают булеву величину (скаляр, матрицу или вектор), которая и указывает выполнено условие (на выходе 1) или нет (на выходе 0).

- `any(`x`)`
  Для аргумента вектора проверяет является ли x ненулевым вектором.

```
octave:1> any(zeros(1,3))
ans = 0
octave:2> any([ 2 0 5 6])
ans =  1
```

  Для аргумента матрицы проверяет является ли каждый из столбцов x ненулевым вектором

```
octave:6> A = [1 0 3; 4 0 8];
octave:7> any(A)
ans =

   1   0   1
```

- `all(`x`)`
  Для аргумента вектора проверяет все ли элементы x ненулевые.

```
octave:8> all([2 0 5 6])
ans = 0
octave:9> all([2 1 5 6])
ans =  1
```

  Для аргумента матрицы проверяет все ли элементы каждого из столбцов ненулевые

```
octave:11> A = [1 1 3; 4 0 8];
octave:12> all(A)
ans =

   1   0   1

octave:13> A = [1 1 3; 4 1 8];
octave:14> all(A)
ans =

   1   1   1

octave:15> all(all(A))
ans =  1
```

Таблица 6: операторы сравнения

==	равно	<	меньше	>	больше
!=	не равно	<=	меньше или равно	>=	больше или равно

Помимо вызова специальных функций проверить выполнение условий можно и при помощи обычных операторов сравнения, список которых приведен в таблице 6.
Результатом таких операций также являются булевы величины. При сравнении скаляров результат 1 означает, что условие выполнено, 0 — условие не выполнено. Если операторы сравнения применяются к матрицам и векторам, то условие проверяется поэлементно, результатом является булева матрица или вектор соответственно. Например, если требуется определить какие элементы матрицы A не меньше некоторого заданного числа c, то просто надо дать команду `A>=c`.

```
octave:1> A = randn(2,5)
A =

   0.88935  -1.34448  -0.36394  -0.46218   1.21992
  -0.90290   2.48469  -0.81393   1.13628   1.33236

octave:2> c = 0;
octave:3> A >= c
ans =

   1   0   0   0   1
   0   1   0   1   1
```

Для проверки можно указывать несколько условий, для этого используются следующие булевы операторы

`&`: покомпонентная конъюнкция *boolean1* `&` *boolean2*
Элемент результата равен 1, если оба соответствующих ему элемента *boolean1* и *boolean2* не равны 0.

```
octave:10> A>=c & A<=2
ans =

   1   0   0   0   1
   0   0   0   1   1

octave:4> [1 0 0 1] & [1 0 2 3]
ans =

   1   0   0   1

octave:5> [1 0; 0 1] & [1 0 ;2 3]
ans =
```

```
   1   0
   0   1
```

`|`: покомпонентная дизъюнкция *boolean1* `|` *boolean2*
Элемент результата равен 1, если хотя бы один из двух соответствующих ему элементов *boolean1* и *boolean2* не равен 0.

```
octave:14> A<=-1 | A>=2
ans =

   0   1   0   0   0
   0   1   0   0   0

octave:6> [1 0 0 1] | [1 0 2 3]
ans =

   1   0   1   1

octave:7> [1 0; 0 1] | [1 0; 2 3]
ans =

   1   0
   1   1
```

`!`: покомпонентное отрицание *boolean*
Элемент результата равен 1, если соответствующий ему элемент *boolean* равен 0.

```
octave:15> !(A<=-1 | A>=2)
ans =

   1   0   1   1   1
   1   0   1   1   1

octave:8> ![1 0 ;2 3]
ans =

   0   1
   0   0
```

`&&`: замкнутая конъюнкция *boolean1* `&&` *boolean2*
Выполняется следующая последовательность шагов. Проверяется выражения *boolean1* и результат конвертируется в булев скаляр, как если бы была применена операция `all(`*boolean1*`)`. Если получен 0, то значение всего выражения 0. Если получена 1, то проверяется выражения *boolean2* и результат конвертируется в булев скаляр, как если бы была применена операция `all(`*boolean2*`)`. Если теперь получен 0, то значение всего выражения 0, в противном случае значение всего выражения равно 1.

```
octave:6> A>-1 && A<3
ans = 0
octave:7> A>-2 && A<2.5
ans =  1
```

||: замкнутая дизъюнкция *boolean1* || *boolean2*
Выполняется следующая последовательность шагов. Проверяется выражения *boolean1* и результат конвертируется в булев скаляр, как если бы была применена операция `all(`*boolean1*`)`. Если получена 1, то значение всего выражения 1. Если получен 0, то проверяется выражения *boolean2* и результат конвертируется в булев скаляр, как если бы была применена операция `all(`*boolean2*`)`. Если теперь получена 1, то значение всего выражения 1, в противном случае значение всего выражения равно 0.

```
 octave:12> A>-1 || A<3
 ans =  1
 octave:13> A>-1 || A<2
ans = 0
```

Замкнутые булевы операторы иногда могут заменить многократное вложение условного оператора `if`. Например, команда

```
A>0 && log(A)
```

предписывает Octave вычислять логарифм элементов матрицы A только в случае, когда все ее элементы положительны. Если это так, то результат выполнения команды равен 1, в противном случае 0.

5.4.5 Формирование подматриц

Обращение к элементам матрицы осуществляется через круглые скобки с указанием номера строки и столбца, на пересечении которых стоит элемент.

```
octave:8> A=[ 2 6 7 5; 8 4 1 3; 2 6 3 5];
octave:9> a_21=A(2,1)
a_21 =  8
```

Средства Octave позволяют указывать не только по одному номеру строки и столбца элемента, но и задавать наборы номеров строк и столбцов, по которым можно вырезать подматрицу. Такие наборы формируются в виде целочисленного вектора, элементы которого не должны превышать количества строк (столбцов) матрицы.

```
octave:11> M=A([1 3],[2 4])
M =

   6   5
   6   5

octave:12> Q=A([1:3],[2 4])
Q =
```

```
   6   5
   4   3
   6   5
```

Если подматрица содержит все строки (столбцы) исходной матрицы, то вместо полного набора номеров можно указать знак двоеточие (:).

```
octave:13> Q=A(:,[2 4])
Q =

   6   5
   4   3
   6   5
```

Вырезать подматрицы можно и по булевым векторам, при этом из исходной матрицы будут браться только столбцы и строки, соответствующие компонентам 1 булева вектора.

```
octave:14> bv = [0 2 0 4] > 0
bv =
   0  1  0  1
octave:15> Q=A(:,bv)
Q =

   6   5
   4   3
   6   5
```

Необходимо обратить особое внимание, что вектор созданный как

```
octave:16> v = [1 0 1 0 0 1]
v =
   1  0  1  0  0  1
```

не является для Octave булевым, поэтому попытка вырезать по нему подматрицу окончится неудачей. Получить булев вектор с аналогичной структурой можно командой

```
v = [1 0 1 0 0 1] > 0
v =
   1  0  1  0  0  1
```

5.5 Операторы управления ходом выполнения программы

На практике решение большинства задач не удается описать с помощью программ линейной структуры, поэтому как и любой высокоуровневый язык программирования Octave содержит условные операторы и операторы циклов. Использование каждого из таких операторов предполагает соблюдение определенного синтаксиса языка.

Для описания операторов примем следующие обозначения:

условие — выражение Octave, результатом выполнения которого является булевая величина;
тело — любая последовательность команд Octave.

5.5.1 Условный оператор `if`

Условный оператор `if` предназначен для проверки некоторого условия. По результатам такой проверки выполняется та или иная последовательность команд. Оператор начинается с ключевого слова `if` и заканчивается ключевым словом `endif`

Существует три основные формы вызова оператора `if`. Самая простая из них имеет вид

```
if (условие)
тело
endif
```

тело оператора будет выполнено, только если значение указанного *условия* истинно.

Заметим, что для оператора `if` любое ненулевое значение *условия* интерпретируется как истина. Если значение *условия* это вектор или матрица, то значение истинность будет только в случае, когда все элементы результирующего вектора или матрицы ненулевые.

```
octave:1> a = [ 2 4 6];
octave:2> b = [ 1 4 6];
octave:3> if a>=b c=a endif
c =

   2   4   6
```

Вторая возможная форма вызова оператора имеет вид:

```
if (условие)
тело1
else
тело2
endif
```

если значение *условия* истинно, то будет выполнено *тело1*, в противном случае будет выполнено *тело2*.

```
if det(A)
 x = A\b
else  x=0
endif
```

В приведенном примере, если определитель матрицы не равен нулю, то будет вычислен вектор x — решение системы уравнений $Ax = b$. В противном случае (если матрица A вырожденная), переменной x будет присвоено нулевое значение.

Третьей и самой общей формой оператора является

```
if (условие1)
тело1
elseif (условие2)
тело2
elseif (условие3)
тело3
...
```

```
elseif (условиеN)
телоN
else
тело(N+1)
endif
```

Последовательно проверяются перечисленные условия до первого их выполнения. Если одно из условий истинно, то выполняется соответствующее тело. Если ни одно из условий не выполнено, то выполняется тело, соответствующее разделу `else`. Ключевое слово *else* в данной конструкции оператора `if` должно встречаться только один раз и только в заключительной части оператора `if`.

```
if (!rem(x,2))
 y = x/2
elseif (!rem(x,3))
 y = x/3
else
 y = 0
endif
```

В приведенном примере сперва осуществляется проверка является ли x четным числом, если да, то значение $y = x/2$. Если x — нечетное, то проверяется делиться ли x на 3. При истинном результате $y = x/3$. Если x не делится ни на 2 ни на 3, то $y = 0$.

5.5.2 Оператор выбора `switch`

Кроме условного оператора `if` в качестве управляющей структуры может оказаться удобным использование оператора выбора `switch`. Эта структура позволяет переходить на одну из ветвей в зависимости от значения заданного выражения. Ее особенность состоит в том, что выбор решения здесь осуществляется не в зависимости от истинности или ложности условия, а является вычислимым.

Оператор начинается ключевым словом `switch`, внутри оператора должен быть хотя бы один раздел `case`, заканчивается оператор ключевым словом `endswitch`. Общая форма вызова оператора следующая

```
switch выражение
case значение1
тело1
case значение2
тело2
...
otherwise
тело
endswitch
```

В конструкции `switch` вычисляется *выражение* и выбирается ветвь, *значение* которой совпадает со значением *выражения*. После выполнения *тела* выбранной ветви происходит выход из конструкции `switch`. Если в последовательности нет *значения*, равного *выражению*, то при наличии раздела `otherwise` выполняется соответствующее ему *тело*, при отсутствии `otherwise` управление передается внешнему оператору, следующему за конструкцией `switch`.

```
switch x<y
case 1
 min=x
otherwise
 min=y
endswitch
```

В приведенном примере проверяется выполнение условия $x < y$, если оно выполнено (значение истинно), то переменной `min` присваивается значение переменной `x`, в противном случае `min= y`.

```
switch sign(x)
case 1
 disp("x>0");
case 0
 disp("x=0")
case -1
 disp("x<0")
endswitch
```

В зависимости от того, чему равно значение сигнум-функции, на экран выводится информация о знаке числа. Описание функции `disp` дано в разделе 5.6 на стр. 76.

5.5.3 Оператор цикла `while`

Как правило под циклом в языках программирования понимают многократное повторение некоторой части кода. Количество таких повторений зависит от условий остановки цикла. Самым простым в Octave можно считать оператор цикла `while`.

Оператор начинается с ключевого слова `while` и заканчивается ключевым словом `endwhile`. Цикл формируется по следующей схеме

`while (`*условие*`)`
тело
`endwhile`

тело цикла будет выполнятся до тех пор, пока *условие* имеет истинное значение. Как и в случае условного оператора `if` истинным считается любое ненулевое значение, если *условие* представляет из себя вектор или матрицу, то истинным оно будет считаться только тогда, когда все элементы вектора или матрицы не равны нулю.

```
F = ones(1,10);
i = 3;
while (i <= 10)
 F(i) = F(i-1) + F(i-2);
 i++;
endwhile
```

Приведен пример программы по созданию последовательности из первых 10 чисел Фибоначчи. Цикл `while` закончит свою работу, когда значение индекса `i` станет равным 11.

5.5.4 Оператор цикла `do-until`

Альтернативой `while` служит оператор цикла `do-until`. Общая схема вызова следующая

```
do
тело
until (условие)
```

Цикл `do-until` отличается от цикла `while` по двум позициям. Первая — *тело* цикла `do-until` выполняется до тех пор, пока *условие* не будет выполнено. Вторая — в `do-until` сначала выполняется *тело*, а потом проверяется *условие*, тогда как в `while` сперва проверяется *условие*, а потом, в зависимости от результата, выполняется *тело*. Таким образом как минимум один раз *тело* цикла `do-until` будет выполнено. Аналогично оператору `if` истинным считается любое ненулевое значение, если *условие* представляет из себя вектор или матрицу, то истинным оно будет считаться только тогда, когда все элементы вектора или матрицы не равны нулю.

Следующий код позволяет построить последовательность первых десяти чисел Фибоначчи с помощью цикла `do-until`.

```
F = ones(1,10);
i = 2;
do
 i++;
 F(i) = F(i-1) + F(i-2);
until (i == 10)
```

5.5.5 Оператор цикла `for`

Цикл с определенным числом повторений некоторых действий удобнее формировать при помощи оператора `for`. Начинается цикл с ключевого слова `for` и заканчивается ключевым словом `endfor`. Общая схема вызова следующая

```
for переменная = выражение
тело
endfor
```

Оператор последовательно перебирает по одной колонке *выражения* и присваивает ее *переменной*. Если *выражение* задает диапазон `[m:n]`, вектор-строку или скаляр, то при каждом выполнении *тела* цикла значением *переменной* будет скаляр. Если *выражение* задает вектор-столбец или матрицу, то при каждом выполнении *тела* цикла значением *переменной* будет вектор-столбец.

Приведем пример генерации последовательности первых десяти чисел Фибоначчи с помощью оператора `for`.

```
F = ones(1,10);
for i=3:10
 F(i) = F(i-1) + F(i-2);
endfor
```

Переменная `i` последовательно пробегает все значения диапазона от 3 до 10, при этом для каждого нового присвоения рассчитывается очередной член последовательности Фибоначчи `F(i)`.

На следующем примере показан процесс построения матрицы B, которая получается из заданной матрицы A путем нормировки каждой из колонок.

```
B = [];
for col = A
 B = [B col / sum(col)];
endfor
```

5.5.6 Дополнительные операторы

Аналогично языку Си, Octave включает в себя операторы досрочного прерывания или пропуска некоторых из операций *тела* цикла.

Оператор `break` обеспечивает прекращение выполнения самого внутреннего из объединяющих его операторов `switch, while, do-until, for`. После выполнения оператора `break` управление передается оператору, следующему за прерванным.

В качестве примера использования оператора `break` приведем функцию, определяющую наименьший делитель некоторого заданного числа.

```
function div = smalldiv(num)
 div = 2;
 while (div*div <= num)
  if rem(num,div) == 0
   break;
  endif
  div++;
 endwhile

 if rem(num,div) div = 1; endif
endfunction
```

Даже если текущее значение переменной `div` удовлетворяет условию `div*div <= num` оператор цикла `while` прервет свою работу, как только число `num` без остатка поделится на `div`.

Если необходимо не навсегда выйти из цикла, а просто досрочно завершить текущую итерацию, то используют оператор `continue`. В отличие от оператора `break`, при выполнении оператора `continue` управление программой передается на начало ближайшего внешнего оператора цикла `for` или `while`.

Рассмотрим следующий пример возможного использования оператора `continue`.

```
vec = randn(5); newvec = [];
for v = vec
 s = sum(v);
 if s == 1
  continue;
 elseif s > 0
  newvec = [newvec v/s];
 else
  newvec = [newvec abs(v)/sum(abs(v))];
 endif
endfor
```

Здесь из системы вектор-столбцов `vec` (по сути `vec` – это матрица) формируется новая система `newvec` по правилу: из `vec` помещаются в `newvec` только те вектора `v`, сумма компонент кото-

рых не равна 1, при этом если эта сумма положительна, то вектор `v` нормируется, если сумма отрицательна, то нормируется вектор `abs(v)`.

Стоит заметить, что в ряде случаев использование оператора `continue` можно заменить индексными выражениями. Например, синтаксис языка Octave позволяет переписать предыдущий пример без оператора `continue` в виде

```
vec = randn(5);
newvec = vec(:, sum(vec) != 1);
bv = sum(newvec) > 0;
newvec(:, bv) = newvec(:,bv) ./ ( ones(5,1) * sum(newvec(:,bv)) );
newvec(:, !bv) = abs(newvec(:,!bv)) ./ ( ones(5,1) * sum(abs(newvec(:,!bv))) );
```

5.6 Ввод и вывод информации

После завершения процесса вычислений результаты работы программы можно сохранить в отдельном файле, чтобы потом иметь возможность воспользоваться ими для анализа или дальнейших преобразований. Рассмотрим лишь самые простые способы сохранения и загрузки данных в Octave. Для более детального знакомства с тонкостями управления процессом чтения и записи данных рекомендуется изучить полное руководство GNU Octave [1].

Сохранить/загрузить данные задачи легко и быстро позволяет пара команд `save/load`. Нетрудно догадаться, что команда `save` сохраняет данные. Общий синтаксис вызова команды следующий

`save` *filename varname1 varname2 ... varnameN*

переменные *varname1 varname2 ... varnameN* и их значения сохраняются в файл *filename*. Если файл с именем *filename* уже существует, то он полностью переписывается новыми данными. Первый после ключевого слова `save` идентификатор интерпретируется Octave как имя файла, он должен быть обязательно. В случае отсутствия других идентификаторов, а именно имен переменных *varname1 varname2 ... varnameN* в файл *filename* будут сохранены все переменные, созданные программой.

```
octave:1> A = rand(2,4); b = ones(2,1);
octave:2> save data A b
```

В приведенном примере матрица `A` и вектор `b` сохраняются в файл `data`, который будет создан Octave в текущей директории. Команда `save` записывает данные в определенном формате, например, файл `data` имеет следующее содержание

```
# Created by Octave 3.0.0, Thu Apr 16 11:00:00 2009 VLAST <shamray@box1>
# name: A
# type: matrix
# rows: 2
# columns: 4
 0.1166630486449072 0.8174254215517049 0.8961450653725824 0.3748763513368052
 0.546587944252344 0.6862222302674722 0.2424641069675111 0.5096399183175931
# name: b
# type: matrix
# rows: 2
# columns: 1
 1
 1
```

Под знаком комментария (#) записывается информация о сохраняемых данных (имя переменной, тип, размерность), далее приводится значение переменной. Всей этой информацией можно будет воспользоваться при очередной загрузке сохраненных данных в исполняемый код Octave. Для этого применяют команду `load`, вызов которой аналогичен `save`

`load` *filename varname1 varname2 ... varnameN*

переменные *varname1 varname2 ... varnameN* и их значения загружаются из файла *filename* в программу Octave. Файл с именем *filename* должен существовать в текущей директории. Первый после ключевого слова `save` идентификатор интерпретируется Octave как имя файла, он должен быть обязательно. В случае отсутствия других идентификаторов, а именно имен переменных *varname1 varname2 ... varnameN*, в программу будут загружены все переменные, которые содержатся в в файле *filename*.

```
octave:1> load data
octave:2> A
A =

   0.11666   0.81743   0.89615   0.37488
   0.54659   0.68622   0.24246   0.50964

octave:3> b
b =

   1
   1
```

Для точного управления вводом и выводом в Octave предлагается модель, аналогичная стандартной I/O-библиотеки языка Си.

Вывести результаты в нужном формате, сопровожденные некоторым информационным сообщением позволяет функция `printf`. Синтаксис вызова функции аналогичен Си

`printf(`*fid, template1, template2, ..., templateN*`);`

Аргумент *fid* может содержать два типа информации — это символы, которые непосредственно выводятся на экран, и спецификаторы формата, определяющие, как выводить аргументы *template1, template2, ..., templateN*.

Спецификатор формата начинается с символа `%` за которым следует код формата. Ниже приведены некоторые из возможных спецификаторов.

- `%d` целое десятичное число
- `%f` десятичное число с плавающей запятой *xx.xxxx*
- `%e` десятичное число в виде *x.xx*`e+`*xx*
- `%c` символ
- `%s` строка символов
- `%%` символ %

Кроме спецификаторов формата данных в аргумент *fid* может содержать управляющие символы, например, такие как

- `\n` новая строка
- `\t` горизонтальная табуляция
- `\v` вертикальная табуляция

Пример использования функции `printf` демонстрирует следующая программа, печатающая квадратные корни всех чисел от 5 до 1.

```
for i=5:-1:1
 printf("Квадратный корень из %d равен %f\n", i, sqrt(i));
endfor
```

Альтернативным способом вывести данные является функция `disp(x)`, которая печатает значение аргумента `x`

```
octave:6> disp("Сгенерирована случайная матрица"), disp(rand(2))
Сгенерирована случайная матрица
   0.18649   0.26070
   0.22552   0.73934
```

5.7 Практические задания

Для закрепления пройденного материала рекомендуется самостоятельно выполнить следующие задания.

1. Сгенерировать случайную целочисленную матрицу размерности 5×8 с элементами не меньше 10.
2. Сгенерировать диагональную матрицу с элементами [2 5 8] на второй диагонали сверху и снизу относительно главной диагонали.
3. Сгенерировать квадратную блочную матрицу, состоящую из четырех квадратных блоков девяти случайных чисел от 0 до 1.
4. Сгенерировать случайный вектор столбец из десяти положительных и отрицательных элементов.
5. Сгенерировать вектор-строку состоящую из пятнадцати двоек.
6. Сгенерировать единичный вектор-столбец размерности 8.
7. Сгенерировать нулевой вектор-строку размерности 10
8. Найти максимальный элемент матрицы.
9. Вычислить натуральный логарифм случайной целочисленной матрицы.
10. За одну команду определить является ли некоторая заданная матрица A нулевой.
11. Сгенерировать произвольную матрицу A размерности 15×10 Найти определитель элементов стоящих в последних 10 строках и во всех столбцах.
12. Найти сумму чисел кратных либо только 3, либо только 5 натурального ряда, члены которого не превышают 1000. Программа не должна содержать ни одного цикла и ни одного условного оператора.
13. В одну строчку вычислений найти разность между суммой квадратов первых ста натуральных чисел и квадратом их суммы.

14. В одну строку вычислений найти положительную срезку[3] матрицы A (сгенерировать самостоятельно функцией `randn`).

15. Найти сумму всех четных чисел Фибоначчи[4], не превышающих 4 миллионов. Допустимо использование не более одного цикла, программа не должна содержать ни одного условного оператора.

16. В матрице A найти сумму всех коэффициентов стоящих в четных строках и нечетных столбцах. Программа не должна содержать ни одного цикла и ни одного условного оператора.

17. Последовательность треугольных чисел строится по правилу: член последовательности с номером $p_n = \sum_{i=1}^{n} i$. Например, седьмое треугольное число равно $1+2+3+4+5+6+7 = 28$. Первые десять членов такой последовательности это $1, 3, 6, 10, 15, 21, 28, 36, 45, 55 \ldots$. Найти произведение первых 100 членов треугольной последовательности кратных 77.

18. Построить вектор v, с компонентами v_i такими, что

$$v_i = \begin{cases} n/2, & \text{если } n \text{ – четное число;} \\ 3n+1, & \text{если } n \text{ – нечетное;} \end{cases}$$

где n пробегает значения от 13 до 23. Программа не должна содержать ни одного цикла и условного оператора.

19. Привести квадратную матрицу к нижней треугольной. Программа должна содержать не более одного цикла.

20. Вычислить все диагональные миноры квадратной матрицы A.

21. Написать функцию сложения булевого вектора и 1. На входе — вектор слагаемое, на выходе результирующий вектор. Например на вход 0 0 1, на выход 0 1 0.

22. Используя функцию булевого сложения вычислить все главные миноры[5] произвольной матрицы A. Программа должна содержать не более одного цикла.

[1] GNU Octave: [Электронный ресурс]. Режим доступа: http://www.gnu.org/software/octave/

[2] Числовой инструментарий: [Электронный ресурс]. Режим доступа: http://www.zabaykalsk.ru/docs/lib/other/numeric1.htm

[3] Положительная срезка матрицы $A = (a_{ij} : i = 1, \ldots, m,\ j = 1, \ldots, n)$, это матрица $P = (p_{ij} : i = 1, \ldots, m,\ j = 1, \ldots, n)$, где $p_{ij} = a_{ij}$, если $a_{ij} > 0$ и $p_{ij} = 0$, если $a_{ij} \leq 0$.

[4] Каждый член последовательности Фибоначчи равен сумме двух предыдущих. $F_1 = 1,\ F_2 = 1,\ F_i = F_{i-2} + F_{i-1},\ i = 3, 4, \ldots$

[5] Главный минор, это определитель подматрицы, стоящей на пересечении строк и столбцов с одинаковыми номерами в исходной матрице

6 Построение научной графики в Gnuplot

Пакет Gnuplot представляет собой интерактивную программу, предназначенную для создания графических изображений, задаваемых в виде функций или файлов данных. Управление работой программы осуществляется при помощи команд на специальном языке. Эти команды можно либо вызывать в командной строке Gnuplot, либо загружать из предварительно подготовленного файла. Gnuplot является некомерческим продуктом, работающим на самых разных платформах (например, UNIX, IBM OS/2, MS Windows, DOS, Macintosh, VMS, Atari).

Перечень возможностей Gnuplot весьма широк:

- графики строятся для функций одной или двух переменных;
- функция может быть задана в любой из трех форм: аналитической (формулой), параметрической или табличной (файл данных);
- кривые можно рисовать в декартовой или полярной системе координат;
- графики можно доплнять надписями или вспомогательными линиями (указатели, стрелки и т.п.);
- поддерживаются различные стили линий и шрифтов;
- вывод графика возможен на экран или в файлы различного типа;
- программа позволяет аппроксимировать экспериментальный набор точек кривыми заданного вида.

В данном руководстве предполагается, что работа с пакетом Gnuplot ведется на базе платформы Linux.

Вызвать Gnuplot можно непосредственно из консоли командой `gnuplot`, после чего пользователю будет предложена командная строка, начинающаяся с промпта (приглашения к действию)

```
gnuplot>
```

Теперь можно приступать к созданию графиков функций.

Основными командами начертания графиков в Gnuplot являются `plot` и `splot`. Команда `plot` рисует 2D-графики, то есть графики функций одной переменной. Команда `splot` предназначена для изображения 3D-графиков — графиков функций от двух переменных. Между синтаксисом, правилами вызова и свойствами этих двух команд много общего.

В простейшем случае вызов команды рисования сопровождается заданием функции, график которой необходимо построить

```
plot sin(x)
splot x**2 + y**2
```

По умолчанию в качестве аргумента функции одной переменной выступает переменная x, для функций двух переменных — x и y. При вызове команд `plot` и `splot` можно указать диапазон изменения аргументов функций

```
plot [-2*pi:2*pi] sin(x)
splot [-15:15] x**2 + y**2
splot [-15:15][-5:5] x**2 - y**2
```

Здесь, в первом случае переменная x изменяется в интервале от -2π до 2π; во втором случае переменная x изменяется от -15 до 15, диапозон изменения y выставляется по умолчанию самой программой; в третьем случае переменная x изменяется от -15 до 15, а y — от -5 до 5.

Можно также указать диапозон изменения возможных значений функции. Для команды `plot` это будет всегда второй из интервалов

```
plot [-5:5][0:10] x**2
```

Для команды `splot` диапазон изменения значений функции всегда задает третий интервал

```
splot [-15:15][-5:5][-100:100] x**2 - y**2
```

На одной картинке сразу можно изображать несколько графиков, для этого при вызове команд рисования функции перечисляются через запятую

```
plot [-5*pi:5*pi][-2:2] sin(x)/x, 1/x
```

Сложные функции задаются при помощи применения математических операторов к элементарным функциям. Списки допустимых операторов и некоторых вещественных функций Gnuplot приведены в таблицах 7 и 8 соответственно.

Таблица 7: Операторы Gnuplot

Символ	Пример	Название	Символ	Пример	Название
Унарные операторы					
`-`	`-a`	унапный минус	`+`	`+a`	унарный плюс
`!`	`!a`	логическое отрицание	`!`	`a!`	факториал
Бинарные операторы					
`**`	`a**b`	возведение в спепень	`!=`	`a!=b`	неравенство
`*`	`a*b`	умножение	`<`	`a<b`	меньше
`/`	`a/b`	деление	`<=`	`a<=b`	меньше или равно
`/+`	`a+b`	сложение	`>=`	`a>=b`	больше или равно
`-`	`a-b`	вычитание	`&&`	`a&&b`	логическое И
`==`	`a==b`	равенство	`||`	`a||b`	логическое ИЛИ
Тернарный оператор					
`?:`	`a?b:c`	если `a` истино, то вернуть `b`, иначе вернуть `c`			

В качестве примера сложной функции рассмотрим

$$f(x) = \begin{cases} \sin(x), & \text{при } 0 \le x < 1, \\ \frac{1}{x}, & \text{при } 1 \le x < 2, \\ \text{неопределена}, & \text{в противном случае.} \end{cases}$$

Для построения графика $f(x)$ необходимо набрать следующие команды

```
f(x) = 0<=x && x<1 ? sin(x) : 1<=x && x<2 ? 1/x : 1/0
plot [0:3] f(x)
```

Таблица 8: Библиотека вещестенных математических функций

Мат. функция	Функция Gnuplot	Мат. функция	Функция Gnuplot
$\|x\|$	`abs(x)`	$\sin(x)$	`sin(x)`
округление до целого вверх	`ceil(x)`	$\arcsin(x)$	`asin(x)`
округление до целого вниз	`floor(x)`	$\mathrm{sh}(x)$	`sinh(x)`
целая часть числа	`int(x)`	arcsh(x)	`asinh(x)`
x^a, a – заданное число	`x**a`	e^x	`exp(x)`
$\ln(x)$	`log(x)`	$\lg(x)$	`log10(x)`
$\cos(x)$	`cos(x)`	$\mathrm{tg}(x)$	`tan(x)`
$\arccos(x)$	`acos(x)`	$\mathrm{arctg}(x)$	`atan(x)`
$\mathrm{ch}(x)$	`cosh(x)`	th(x)	`tanh(x)`
arcch(x)	`acosh(x)`	arcth(x)	`atanh(x)`
$\mathrm{arctg}(y/x)$	`atan2(y,x)`		

Отметим, что Gnuplot игнорирует неопределенные значения, то есть последняя ветвь функции $(1/0)$ не даст точек для построения. Также необходимо обратить внимание, что $f(x)$ будет построена как непрерывная функция на разрыве. Для построения разрыва придется создать отдельные функции для двух кусков

...

или использовать параметрическое задание функции

...

Команды `plot` и `splot` строят графики функций заданных таблично. Таблица должна быть сохранена в виде текстового файла, каждая строка которого представляет наборы точек, разделенных пробелами. Вызов команд сопровождается именем файла, заключеным к ковычки (одинарные или двойные). По умолчанию значения в первых трех столбцах такого файла понимаются Gnuplot как координаты (x, y, z) соответственно.

Пусть каждая строка файла `data.1` содержит набор из четырех координат

```
-10.00000000   0.54402111  -0.83907153 100.00000000
 -9.90000000   0.45753589  -0.88919115  98.01000000
 -9.80000000   0.36647913  -0.93042627  96.04000000
 -9.70000000   0.27176063  -0.96236488  94.09000000
 -9.60000000   0.17432678  -0.98468786  92.16000000
 -9.50000000   0.07515112  -0.99717216  90.25000000
 -9.40000000  -0.02477543  -0.99969304  88.36000000
```

и так далее. Команда

```
plot 'data.1'
```

строит график функции $f(x)$, где значение аргумента x берется из первого столбца, а значение функции $f(x)$ — из второго столбца файла `data.1`. Вызов команды

```
splot 'data.1'
```

построит трехмерный график функции $f(x, y)$, где значения аргументов x и y берутся из первого и второго столбцов соответственно, значение функции $f(x, y)$ — из третьего столбца файла `data.1`.

Построить график функциональной зависимости между любыми столбцами файла данных поможет модификатор `using`, после которого указываются номера рассматриваемых столбцов через двоеточие (:)

```
plot "data.1" using 1:4
plot "data.1" using 4:1
splot "data.1" using 4:3:2
```

В первом примере аргумент функции берется из первого солбца файла `data.1`, а значение — из четвертого. Во втором примере наоборот, аргумент — из четвертого столбца, а значение — из первого. В третьем примере значения двух аргументов берутся из четвертого и третьего столбцов, значение функции — из второго. Важно помнить, что при построении двумерного графика (команда `plot`) в модификаторе `using` надо указать ровно два столбца, а при построении трехмерного графика (команда `splot`) — ровно три столбца.

Для каждой функции после ключевого слова with можно указать стиль графика. Это могут быть, например, линии (lines), точки (points), или точки с обозначением ошибки измерения (yerrorbars). Есть и другие стили. Далее можно указать тип линий или точек (lt, pt), толщину линий (lw) и размер точек (ps). Эти же команды позволяют рисовать графики функций, заданных наборами точек, записанных в файл. plot [-5*pi:5*pi] sin(x)/x with lines lt 1 lw 5, 1/x with lines lt 2 lw 1

A Алфавит математики LaTeX

В таблицах 9-18 приведена лишь часть специальных математических символов, которые могут пригодиться экономисту-математику.

Таблица 9: Математические акценты

$\hat{x}$	\hat{x}	$\tilde{x}$	\tilde{x}	$\bar{x}$	\bar{x}	$\vec{x}$	\vec{x}
$\dot{x}$	\dot{x}	$\ddot{x}$	\ddot{x}				

Таблица 10: Строчные греческие буквы

α	\alpha	β	\beta	γ	\gamma	δ	\delta	ϵ	\epsilon
ε	\varepsilon	ζ	\zeta	η	\eta	θ	\theta	ϑ	\vartheta
ι	\iota	κ	\kappa	λ	\lambda	μ	\mu	ν	\nu
ξ	\xi	τ	\tau	υ	\upsilon	ϖ	\varpi	ρ	\rho
ϱ	\varrho	σ	\sigma	ς	\varsigma	o	o	π	\pi
ϕ	\phi	φ	\varphi	χ	\chi	ψ	\psi	ω	\omega

Таблица 11: Прописные греческие буквы

Γ	\Gamma	Δ	\Delta	Θ	\Theta	Λ	\Lambda	Ξ	\Xi
Π	\Pi	Σ	\Sigma	Υ	\Upsilon	Φ	\Phi	Ψ	\Psi
Ω	\Omega								

Таблица 12: Многоточия

$\ldots$	\ldots	$\cdots$	\cdots	$\vdots$	\vdots	$\ddots$	\ddots

Таблица 13: Дополнительные символы

∞	\infty	$\emptyset$	\emptyset	$\|$	|	$\angle$	\angle	∇	\nabla
$\forall$	\forall	$\exists$	\exists	$\neg$	\neg	∂	\partial	$\backslash$	\backslash

Таблица 14: Символы бинарных операций

$\cap$	\cap	$\cup$	\cup	$\vee$	\vee	$\wedge$	\wedge	$\bigtriangleup$	\bigtriangleup
$\pm$	\pm	$\mp$	\mp	$\times$	\times	$\div$	\div	$\circ$	\circ

Таблица 15: Символы сравнения

$\leq$	\leq	$\geq$	\geq	$\subset$	\subset	$\supset$	\supset	$\sim$	\sim
$\subseteq$	\subseteq	$\supseteq$	\supseteq	$\in$	\in	$\ni$	\ni	$\notin$	\notin
$\approx$	\approx	$\equiv$	\equiv	$\neq$	\neq				

Таблица 16: Символы переменного размера

$\sum$	\sum	$\prod$	\prod	$\coprod$	\coprod	$\int$	\int	$\oint$	\oint
$\bigcap$	\bigcap	$\bigcup$	\bigcup						

Таблица 17: Стрелки

$\leftarrow$	\leftarrow	$\Leftarrow$	\Leftarrow	$\rightarrow$	\rightarrow (\to)
$\Rightarrow$	\Rightarrow	$\leftrightarrow$	\leftrightarrow	$\Leftrightarrow$	\Leftrightarrow
$\mapsto$	\mapsto	$\uparrow$	\uparrow	$\Uparrow$	\Uparrow
$\downarrow$	\downarrow	$\Downarrow$	\Downarrow	$\updownarrow$	\updownarrow
$\longleftarrow$	\longleftarrow	$\Longleftarrow$	\Longleftarrow	$\longrightarrow$	\longrightarrow
$\Longrightarrow$	\Longrightarrow	$\longleftrightarrow$	\longleftrightarrow	$\Longleftrightarrow$	\Longleftrightarrow (\iff)
$\longmapsto$	\longmapsto	$\Updownarrow$	\Updownarrow	$\nearrow$	\nearrow
$\searrow$	\searrow	$\swarrow$	\swarrow	$\nwarrow$	\nwarrow

Таблица 18: Специальные функции

`\cos`	`\arccos`	`\tan`	`\cot`	`\csc`	`\cosh`	`\exp`	`\ln`	`\lg`	`\log`
`\arg`	`\det`	`\sup`	`\inf`	`\max`	`\min`	`\lim`	`\liminf`	`\limsup`	`\ker`