Краткое введение в **Gnuplot**

А. Н. Нечаев

14 февраля 2005 г.

Это действительно очень краткое руководство. В некоторой степени оно просто суммирует опыт работы автора с программой. Например, здесь абсолютно не рассматривается сглаживание данных, а также трёхмерная графика. Тем, кому будет интересно узнать про дополнительные возможности, лучше всего будет заняться самостоятельными экспериментами с программой (тем более, что, по моему глубочайшему мнению, это есть самый лучший способ изучения чего-либо нового).

Содержание

1 Что такое Gnuplot?

Gnuplot — маленькая и удобная в обращении программа, предназначенная для построения всевозможных графиков. Управление программой осуществляется с помощью достаточно простого и гибкого внутреннего языка, чрезвычайно похожего на привычные языки программирования. Например, знак `+` в **Gnuplot** соответствует операции сложения, `*` — операции умножения и так далее.

Важно отметить, что **Gnuplot** различает верхний и нижний регистры. Например, если вместо команды `help` в командной строке написать `PLOT`, то **Gnuplot** выдаст сообщение об ошибке. Точно такие же правила выполнятся и для имён, определяемых пользователем. Например, можно определить две функции `f(x)` и `F(x)` (о задании функций пользователя речь пойдёт чуть позже), при этом обе функции будут различными.

Ну а теперь приведём "основные" команды

`cd` Изменение текущего каталога (эта команда очень полезна при визуализации числовых данных из файлов).
Пример: `cd "mydir"`

`pwd` Команда имеет тот же смысл, что и аналогичная команда в Linux — она позволяет посмотреть, какой каталог в текущий момент активен
Пример: `pwd`

`reset` Позволяет убрать все изменения в настройках программы, сделанные пользователем.
Пример: `reset`

`help` Можно сказать, что это одна из наиболее часто используемых команд. Её назначение — вызов справки по **Gnuplot**. Если вызвать команду без аргументов, то вызывается общий файл справки. При использовании аргумента (им является любая команда **Gnuplot**) происходит обращение к разделу справки, отвечающего аргументу.
В приведённом вызывается справка по команде `splot`
Пример: `help splot`

Кроме команд, требуется знать ещё несколько параметров **Gnuplot**

`xrange`

Параметр устанавливает диапазон независимой переменной, в котором будет рисоваться график. Параметр можно изменять с помощью команды `set xrange[{min}:{max}]`. `{min}` и `{max}` — необязательные параметры, определяющие нижнюю и верхнюю границы интервала. В качестве значений для них может выступать число или же символ `*` (в этом случае соответствующая граница будет вычисляться автоматически).

Примеры:

- `set xrange[0:10]` Нижняя граница 0, верхняя — 10
- `set xrange[*:1.2]` Теперь нижняя граница вычисляется автоматически, а верхняя устанавливается равной $1,2$
- `set xrange[-0.98:*]` Нижняя граница равна $-0,98$, верхняя вычисляется **Gnuplot**

"Откатить" сделанные изменения можно с помощью `reset`

`yrange`

Параметр делает точно то же самое, что и `xrange`, только в отношении оси ординат.

`key`

Отвечает за легенду рисунка, то есть метку, располагающуюся в верхнем правом углу графика. Лично мне нравится графики без какой-либо избыточной информации, поэтому чаще всего приходится использовать команду `set nokey`, отменяющую появление легенды. Однако можно и принудительно установить легенду с помощью команды `set key '{text}'`.

Пример: `set key 'This is the graph of Bessel function'`

`terminal`

Этот параметр определяет, куда именно **Gnuplot** будет выводить построенный график. По умолчанию это значение устанавливается таким образом, что пользователь получает график на мониторе. Если же требуется получить "твёрдую" копию (например, напечатанный график или же графический файл), то значение надо переустановить с помощью `set terminal {terminal-type}`. Возможные значения `{terminal-type}` можно посмотреть с помощью `help set terminal`.

Примеры:

- `set terminal postscript` После **Gnuplot** будет выводить данные в формате PostScript.
- `set terminal X11` X11 — это имя дисплея в операционной системе Linux, так что после этой команды **Gnuplot** будет выводить графики на монитор.
- `set terminal windows` То же самое, что и в предыдущем случае, но только в операционных системах Windows
- `set terminal latex` После такой команды график будет печататься в графике LaTeX. Если Вы пользуетесь LaTeX, то это неплохой способ для вставки графиков в текст, однако надо иметь в виду, что размеры получающихся файлов могут оказаться большими.

`output`

Имя файла или устройства для вывода данных. Команда для установки `set output {'filename'}`

Если Вы не установите его значения, но попробуете изменить текущее значение `terminal`, то очередной график **Gnuplot** напечатает прямо на экране (в текстовом виде). Например, после выполнения таких команд

```
gnuplot>set terminal postscript
gnuplot>plot sin(x)
```

у Вас возникнет возможность пронаблюдать текст PostScript на экране (никакого файла на диске при этом не возникает). Поэтому лучше всего не забывать устанавливать имя выходного файла.

Пример: `set output "myfile.ps"` После этого у Вас возникает на диске (в текущем каталоге) пустой файл с именем `myfile.ps`. После выполнения рисования в файл запишется содержимое — график в выбранном Вами формате. Важное замечание: если требуется получить несколько графиков, причём каждый из них должен быть записан в свой файл, то `set output` надо выполнять перед каждым процессом рисования, иначе все графики будут записаны в один и тот же файл.

2 Арифметика в Gnuplot

Перед переходом к описанию процедуры построения графиков, стоит несколько слов сказать об арифметике, используемой в **Gnuplot**. Дело в том, что внутренний язык используется две арифметики — целочисленную и действительную (программистам на C или Python можно дальше не читать!). На практике это означает, что результатом любой арифметической над двумя целыми числами всегда будет целое число. Если же в операции участвуют как действительные, так и целые числа, то результат будет действительным.

В общем-то, это свойство проявляется не слишком часто, но иногда может привести и к неприятной ситуации. Пусть, например, нам требуется нарисовать графики функций

$$f(x) = a \sin x,$$

где параметр a может пробегать некоторое множество значений, содержащее число $\frac{1}{2}$. Если же попробовать вычислить это в **Gnuplot**, то получится несколько необычный для начинающего пользователя результат (ниже приводится несколько команд — надеюсь, что их смысл будет понятен даже неподготовленному пользователю)

```
gnuplot> a = 1
gnuplot> b = 2
gnuplot> print a
1
gnuplot> print a/b
0
```

Итак, для **Gnuplot** результатом деления 1 на 2 оказывается равным нулю. В этом нет ничего неожиданного, если рассматривать нашу операцию как деление двух целых чисел[1].

Проще всего избежать таких проблем, если при задании постоянных сразу же указывать, какое именно число имеется виду — действительное или целое. Делается это с помощью обыкновенной точки: если **Gnuplot** находит в числе точку, то число считается действительным, в противном случае — целым. Например, `4` будет целым числом, а `4.` — действительным.

3 Построение графика функции одной переменной.

Для создания двумерной графики (то есть построения графиков функций или же просто изображения набора точек на плоскости) в **Gnuplot** используется команда `plot`. Общий синтаксис этой команды выглядит следующим образом

```
plot {<ranges>} {<function>}
      {axes <axes>} {<title-spec>} {with <style>}
       {, {definitions,} <function> ...}
```

Аргумент `<ranges>`

Необязательный аргумент `{<ranges>}` определяет часть плоскости, на которой будет изображаться график. В наиболее общем случае команда установления диапазона изменения независимой переменной имеет такой вид

```
[{<dummy-var>=}{{<min>}:{<max>}}]
```

Первый параметр, `{<dummy-var>}`, определяет имя независимой переменной[2]. Если его не указать, то применяется имя, установленное по умолчанию, то есть `x`. Параметры `{<min>}` и `{<max>}` устанавливают границы диапазона. Их значения могут быть или постоянной, определённой раньше, или же `*` (в этом случае значение границы определяется **Gnuplot** автоматически).

Для установления области изменения зависимых переменной используется аналогичный синтаксис

```
[{{<min>}:{<max>}}]
```

Параметры `{{<min>}` и `{<max>}}` имеют тот же самый смысл, что и при задании независимой переменной.

Первым всегда указывается диапазон изменения независимой переменной, затем — интервалы изменения зависимой, поэтому при обычном задании порядок записи должен быть такой: первым указывается диапазон для x, затем — y. При параметрическом задании сначала идёт диапазон `trange`, потом — `xrange` и `yrange`.

Примеры

Рисование графика $\sin x$ с использованием текущих диапазонов (если диапазоны не были установлены с помощью директивы `set`, то все границы **Gnuplot** определит автоматически)

```
gnuplot>plot sin(x)
```

[1] Напомним, что в целочисленной арифметике результатом деления двух чисел a и b будет число p, такое, что $a = pb + q$, $0 \leqslant q < 1$

[2] При параметрическом способе задания графика независимой переменной будет параметр

Рисование того же самого графика, но в пределах $[-\pi, \pi]$

```
gnuplot>plot [t=-pi:pi] sin(t)
```

Обратите внимание, что для обозначения независимой переменной используется символ `t`
Теперь изменим ещё и границы зависимой переменной

```
gnuplot>plot [-pi:pi] [-0.5:0.5] sin(x)
```

Теперь установим только нижнюю границу для независимой переменной и верхнюю — для зависимой

```
gnuplot>plot [0:*] [*:0.5] sin(x)
```

Аргумент `<function>`

С помощью этого аргумента пользователь указывает **Gnuplot**, что именно ему требуется нарисовать. Значением аргумента может быть математическое выражение или же его имя.

Конструкция математических выражений в **Gnuplot** выглядит точно так же, как и в обычном языке программирования типа C, Pascal или Basic. Любое математическое выражение состоит из чисел, известных **Gnuplot** функций и операторов[3].

Примеры

Нарисуем плотность вероятности отвечающую нормальному распределению с математическим ожиданием 2.35 и среднеквадратическим отклонением 1.6

```
gnuplot>plot 1/sqrt(2)/sqrt(pi)*exp(-(x-2.35)**2/1.6**2)
```

Иногда возникает необходимость нарисовать несколько графиков одной и той же функции (например, нам требуется посмотреть, как она ведёт себя в разных областях). В этом случае удобно будет сначала определить эту функцию, а потом уже рисовать её

```
gnuplot>p(x)=sin(x)*exp(-x**2)
gnuplot> plot p(x), p(x+1)
```

Аргумент `{<title-spec>}`

Эта опция определяет строку, появляющуюся в легенде рисунка. Общий синтаксис имеет следующий вид

```
title "<title>"|notitle
```

При отсутствии опции в текст легенды войдёт математическое выражение, описывающее функцию. Если же легенда задана, то в текст включается содержимое строки (сами кавычки в легенду не входят). Для задания специальных символов придётся указывать их восьмеричный номер. Возможно использование символа табуляции `\t`[4], однако символ перехода на новую строку игнорируется в любом случае.

Если требуется получить рисунок без легенды, можно использовать опцию `notitle` или же установить “нулевую” легенду (то есть `title ''`).

Примеры

Нарисуем графики двух функций, например, функций Бесселя и “подпишем” их (к сожалению, при печати рисунков иногда возникают проблемы с русскими буквами, поэтому в легенде мы будем использовать английский язык)

```
gnuplot>plot besj0(x) title '0th Bessel function',
```

[3] Полный список всех операторов и функций приведён в приложении A

[4] управляющие символы доступны только в строках, заключённых в двойные кавычки

Аргумент {with}

При построений графиков **Gnuplot** позволяет использовать множество стилей (например, для рисования графика одной функции можно использовать сплошную линию, а другой — сплошную линию с точками). В общем виде обращение к этой опции имеет такой вид

```
with <style> { {linestyle | ls <line_style>}
               | {{linetype | lt <line_type>}
                  {linewidth | lw <line_width>}
                  {pointtype | pt <point_type>}
                  {pointsize | ps <point_size>}} }
```

Первый аргумент, `<style>`, определяет стиль линии, используемой для рисования графика. Он может принимать такие значения: `lines`, `points`, `linespoints`, `impulses`, `dots`, `steps`, `fsteps`, `histeps`, `errorbars`, `xerrorbars`, `yerrorbars`, `xyerrorbars`, `boxes`, `boxerrorbars`, `boxxyerrorbars`, `financebars`, `candlesticks` или `vectors`. Некоторые из стилей требуют дополнительной информации.

4 Дискретные данные в Gnuplot

Для рисования дискретных данных[5] используется та же самая команда `plot`, но вместо математического выражения указывается имя файла (заключённое в простые или двойные кавычки). В наиболее общей форме команда представления данных имеет такой вид

```
plot '<file_name>' {index <index list>}
                       {every <every list>}
                       {thru <thru expression>}
                       {using <using list>}
                       {smooth <option>}
```

Аргумент `{index <index list>}`

Аргумент позволяет рисовать только часть данных из файла (разделителями между частями файла служат две пустые строчки). В дальнейшем мы будем называть части файла *группами*. Общий синтаксис команды

```
plot 'file' index <m>{{:<n>}:<p>}
```

Если указать только одно число (например, `index 1`), то при формировании рисунка будут использованы данные первой группы; указание двух чисел (`index 1:3`) говорит, что при построении изображения надо использовать данные из групп 1, 2, 3. Последняя форма записи, с использованием трёх чисел позволяет формировать график только по выбранным группам. Так, например, команда `index 0:10:2` выбирает для изображения группы данных с номерами 0, 2, 4, 6, 8 и 10.

По умолчанию считается, что при построении изображения требуется использовать все данные из файла

Аргумент `{every <every list>}`

Эта опция позволяет производить выборку определённых точек из файла с данными. Общий синтаксис команды имеет такой вид

```
plot 'file' every {<point_incr>}
                      {:{<block_incr>}
                        {:{<start_point>}
                          {:{<start_block>}
                            {:{<end_point>}
                              {:<end_block>}}}}}
```

[5] Для нас дискретные данные — это набор или таблица чисел, записанная в файл

Для построения графика используются циклически выбранные точки. Первая точка имеет номер <start_point>, последняя — <end_point>. Шаг цикла определяется значением <point_incr>. Остальные параметры определяют правила выборки точек из блоков.[6]

Аргумент {thru <thru expression>}

Опция введена для совместимости с ранними версиями **Gnuplot**. Например, следующая команда

```
plot 'file' thru f(x)
```

полностью эквивалентна команде

```
plot 'file' using 1:(f($2))
```

Аргумент {using <using list>}

Пожалуй, using представляет собой наиболее часто используемую опцию. Обращение к опции имеет такой вид

```
plot 'file' using {<entry> {:<entry> {:<entry> ...}}} {'format'}
```

Если задан формат данных, то при чтении записей из файла используется функция scanf с заданными параметрами[7]. Если же формат отсутствует, то **Gnuplot** пытается прочитать данные из файла по следующему правилу: читается очередная строка из файла, затем разбивается на отдельные столбцы (разделителем столбцов считается один или несколько пробелов). Все прочитанные данные записываются в массив.

Доступ к полученному массиву осуществляется по столбцам. Например, если требуется получить график зависимости второго столбца от первого (то есть по оси y откладываются значения из второго столбца, а по оси x — из первого), то требуемая команда имеет такой вид

```
gnuplot> plot "datafile.dat" using 1:2
```

Нумерация столбцов начинается с единицы. Вообще говоря, существует ещё и дополнительный столбец с номером 0, содержащий порядковые номера записей.

Возможно получить доступ и к каждой отдельной записи в столбце. Как это происходит, проще всего посмотреть на конкретном примере. Пусть файл datafile.1.dat состоит из двух столбцов чисел. Выполним такую команду

```
gnuplot> plot 'datafile.1.dat' u 1:(sin(pi*$2))
```

При выполнении этой команды **Gnuplot** производит следующие действия: читается очередная запись из файла (то есть пара чисел, например, α и β) и вычисляется значение $z = \sin \pi\beta$. Затем на график выводится очередная точка с координатами (x, z).

При выполнении этой команды возникает график, строящийся по следующему правилу: по оси x откладываются числа, берущиеся из столбца с номером 1, а соответствующее значение по оси y получается при вычислении $\sin \pi y$

Аргумент smooth

Gnuplot включает в себя небольшое количество подпрограмм для интерполяций и аппроксимаций функций. Доступ к этим подпрограмм осуществляется с помощью опции smooth. К сожалению, эти методы на обеспечивают выдачи аналитического материала (например, вычисленных коэффициентов аппроксимационного многочлена). Подробно о этом аргументе можно прочитать в справке **Gnuplot**:

```
gnuplot> help smooth
```

[6] Правда, у автора так и не получилось научится правильно работать с выборкой точек из блоков. Возможно, будет проще использовать сочетание index и every

[7] Более подробно о форматах можно прочитать в самом **Gnuplot** в справке: help plot datafyle

5 Специальные файлы

В **Gnuplot** существует вводить данные не только из уже существующих на диске файлов, но и формировать набор "на лету", прямо из текущей сессии. Для этой цели цели используется специальный файл `'-'`. Методика его применения практически полностью совпадает с использованием обычной команды `plot`: используются все те же самые стили линий, легенды и остальные параметры. Единственное отличие заключается в том, что данные вводятся непосредственно с клавиатуры. Вот простой пример использования этой возможности:

```
gnuplot> plot '-' with linespoints
> 1 2
> 2 1
> 3 4
> e
```

Признаком окончания ввода данных служит символ `e` (для нормального завершения это должен быть первый значащий символ в строке).

К сожалению, **Gnuplot** не хранит использованные данные, поэтому при использовании этой техники возможности по перерисовке ограничены. Это означает, что после ввода команды `replot`, возможно, потребуется заново ввести весь набор данных заново. Зато технология специальных файлов позволяет очень эффективно использовать **Gnuplot** в качестве встраиваемого компонента в различного рода программы.

6 Командные файлы Gnuplot

Gnuplot можно исполнять не только в обычном интерактивном режиме, но и в пакетном режиме. Если в интерактивном режиме все команды вводятся с клавиатуры, то в при пакетном исполнении **Gnuplot** получает свои инструкции из заранее написанного файла. Можно сказать, что пакетное исполнение — это написание программ на встроенном языке **Gnuplot**.

При написании командных файлов используются точно те же самые команды, что и при интерактивном исполнении. Тем не менее, для эффективного создания скриптов желательно знание большего количества команд, нежели чем для простой интерактивной работы. Поэтому изложение начнём с набора дополнительных команд.

Команда `if` означает обычное ветвление. Как и большинстве языков программирования, команда имеет несколько вариантов:

```
if (<condition>) <command-line>;
if (<condition>) <command-line1>; else <command-line2>
```

В первом случае выполнение условия `<condition>` влечёт за собой выполнение команды <commanf-line>. Во втором случае при выполнении условия выполняется `<command-line1>`, а в противном случае — `<command-line2>`. Важно заметить, что **Gnuplot** очень "требовательно" относится к расположению команд, а именно попытка выполнить строки

```
if (1>0) print '+';
else print '-'
```

приведёт к ошибке. Причина этого заключается в том, что **Gnuplot** выполняет все команды построчно. Так, в приведённом нами примере первая строка рассматривается как обычный вариант команды `if`. Так как условие `1>0` истинно, то **Gnuplot** выполняет требуемую команду и выводит на терминал знак `+` и переходит к выполнению второй строки, начинающейся со слова `else`. Но использование этого слова вне конструкции `if` невозможно, что и приводит к возникновению "чрезвычайной" ситуации. Ещё один факт, на которых стоит обратить внимание — условие всегда должно быть заключено в скобки, точно также как и язык программирования C.

При необходимости команду `if` можно расширить на использование трёх и более количества условий. Для этого используется стандартный приём — соединение нескольких операторов в один:

```
if (<condition1>) <command-line1>;
else if (<condition2>) <command-line2>;
else if ...
else <command-linen>;
```

В этом случае сначала проверяется выполнение условия `<condition1>`. Если оно оказывается истинным, то выполняется командная строка `<command-line1>` и управление передаётся на конец всего оператора. В противном случае выполняется проверка второго условия `<condition2>` и если оно истинно, то отрабатывается команда `<command-line2>` и выполнение всего оператора завершается. Это процесс совершается до тех пор, пока не будет найдено истинное условие (или же окажется, что все условия ложны — в этом случае выполняется командная строка `<command-linen>`).

Gnuplot выполняет последовательно все строки из командного файла. Время, затрачиваемое на отработку каждой из строк, очень мало. Иногда это оказывается очень удобно, но в других ситуациях мешает. “Затормозить” выполнение можно с помощью команды `pause`. Существует два варианта этой команды:

```
pause <time> {"<string>"}
pause mouse {"string"}
```

Первый вариант команды выводит на экран содержимое строки `<string>` и приостанавливает выполнение текущего командного файла на время `<time>`. Сам параметр `time` может принимать несколько значений. Так, если в качестве него передано значение `-1`, то есть использован вариант

```
pause -1 'Please press any key'
```

то выполнение приостанавливается до нажатия произвольной клавиши на клавиатуре. Значение `0` означает нулевую задержку. Можно сказать, что команда

```
pause 0 'A-A-A'
```

полностью эквивалентна `print`

```
print 'A-A-A'
```

Если в качестве аргумента `time` передано произвольное положительное число, то выполнение приостанавливается именно на такое количество секунд.

Второй вариант, с использованием аргумента `mouse`, вызывает остановку выполнения программы до тех пор, пока не будет нажата кнопка мыши или комбинация *Control-C*. При этом, правда, необходимо, чтобы существовало графическое окно **Gnuplot** (то есть был построен какой-либо график).

Команда `load` выполняет загрузку существующего командного файла. Общий синтаксис команды выглядит так

```
load '<filename>'
```

Команду можно использовать не только в пакетном, но и в интерактивном режиме. Это может оказаться полезно в случае, если часто приходиться большое количество одинаковой работы (например, часто приходится рисовать данные из одного и того же файла с использованием специфических типов линий).

В **Gnuplot** существует возможность повторной загрузки командного файла, это выполняется с использованием команды `reread`. Комбинируя `reread` с ветвлением можно организовать циклы.

Например, проследить изменение графика функции

$$f(x) = \frac{1}{1 + ax^2}$$

при изменении параметра a. Делается это следующим способом: создадим файл с именем `plotter`, содержащий следующие строки

```
# циклическое рисование
n = n+1
plot 1/(1+n*1.2*x*x)
pause -1 'Press any key to continue'
if (n < 10) reread
```

Именно этот файл будет играть роль тела[8] цикла.

Теперь необходимо создать основной файл программы. Запишем его в файл `cycle`:

```
# основная часть файла
n = 1
set xrange[-3:3]
set yrange[0:1]
load 'plotter'
```

Запуск программы осуществляется совершенно тривиальным способом, требуется только набрать в командной строке

```
gnuplot cycle
```

Разумеется, этот же самый файл можно загрузить и прямо из интерактивной сессии. Для этого служит та же самая команда `load`:

```
gnuplot> load 'cycle'
```

Единственное, что надо учесть — текущий рабочий каталог должен содержать оба файла, `plotter` и `cycle`.

У **Gnuplot** существует ещё один вариант перезагрузки файлов, кроме `load` — команда `call`:

```
call '<input-file>' <option1> <option2> ... <optionn10>
```

Её единственное отличие состоит в том, что в неё можно передавать до 10 дополнительных параметров. При выполнении этой команды **Gnuplot** начинает выполнение вызванного файла `<input-file>`, отыскивая в каждой строке комбинации вида `$n`, где `n` — одна из цифр от 0 до 10. Если такое выражение найдено, то оно заменяется соответствующим значением из переданных параметров.

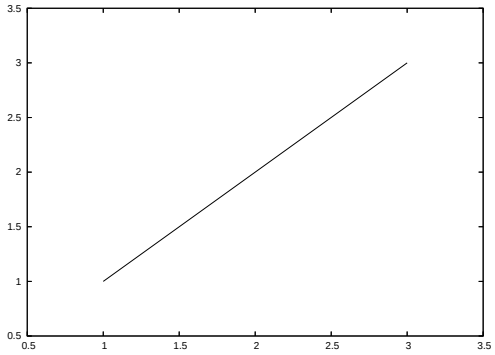
В версии 4.0 у **Gnuplot** добавился ещё одна команда, существенно расширяющая возможности по написанию командных файлов, `bind`. Она позволяет сопоставлять определённые комбинации системных событий командам **Gnuplot**. Например, если в интерактивной сессии подать команду

```
bind a 'print 1'
```

а затем нарисовать некоторый график, то при нажатии клавиши `a` на клавиатуре (при условии, что активным является окно с графиком!) в самой интерактивной сессии будет выводиться на экран единичка.

[8] Примечание для пользователей: телом цикла называется последовательность операторов, выполняемых в цикле

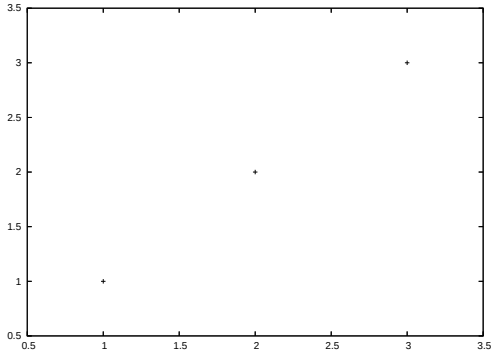
7 Стили линий



lines При использовании этого стиля две соседние точки графика соединяются отрезком. Ширина и тип линии определяются текущими значениями `linewidth` и `linetype` соответственно.

Пример

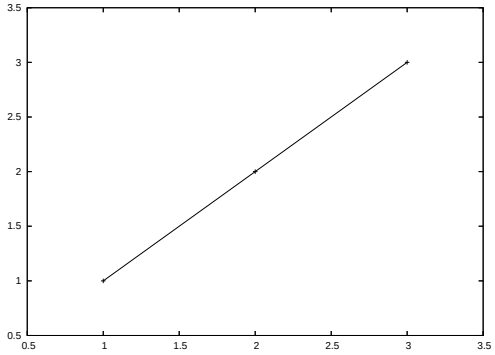
```
gnuplot> plot "datafile.dat"with lines
>linetype 1 lw 1'
```



points На месте каждой из точек рисуется маленький символ. Тип символа определяется текущим значением `pointtype` (возможное сокращение `pt`), а его размер — значением `pointsize` (сокращение `ps`)

Пример

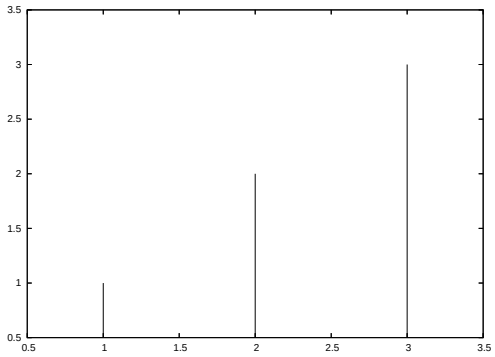
```
gnuplot> plot "datafile.dat"with points
>pt 3 ps 4
```



linespoints Стиль представляет собой комбинацию `lines` и `points`: сначала на месте каждой точки рисуется символ, а затем соседние точки соединяются отрезком. Параметры стиля такие: толщина линии `linewidth`, вид линии `linetype`, тип точки `pointtype` и размер символа `pointsize`

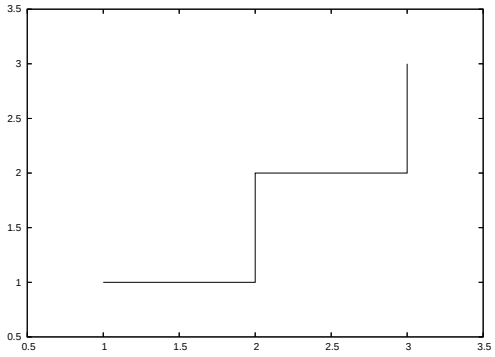
Пример

```
gnuplot> plot "datafile.dat" with linespoints \  >pt 2
```

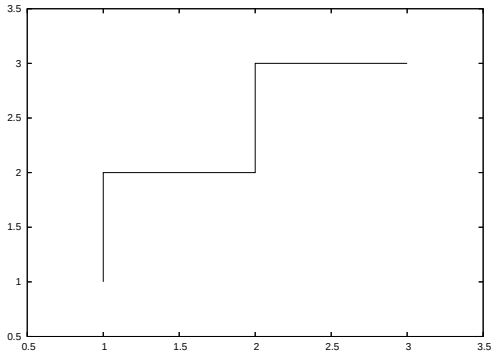


impulses При использовании этого стиля до каждой точки графика проводится вертикальная линия от оси **x**
Пример
`plot "datafile.dat" with impulses`

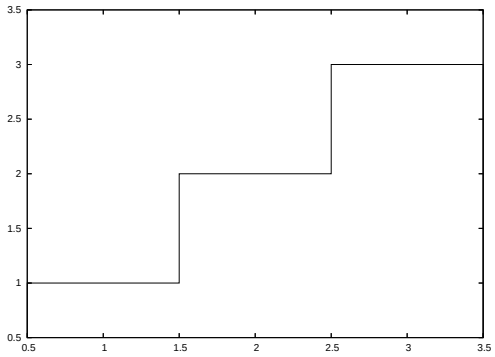
dots Пожалуй, это один из наиболее часто используемых стилей. На месте каждой из точек рисуется очень маленький кружочек (настолько маленький, что даже рисунка не будем приводит!).
Пример
`gnuplot> plot "datafile.dat" with dots`



steps Две соседние точки соединяются “ступенькой”, сначала проводится горизонтальная прямая, потом — вертикальная
Пример `gnuplot> plot "datafile.dat" with steps`



fsteps Стиль очень похож на предыдущий. Разница заключается в порядке построения “ступеньки”. Теперь сначала строится вертикальная, а уж потом — горизонтальная линии.

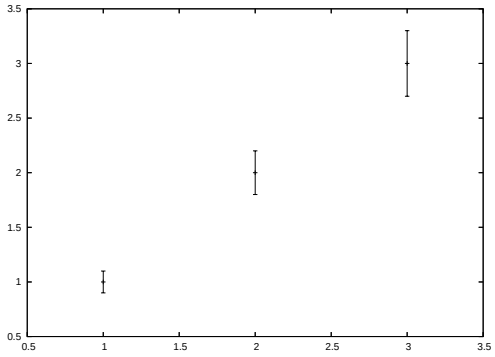


histeps. Стиль предназначен для построения гистограмм. Процесс построения изображения выглядит следующим образом.
Пусть (x_0, y_0), (x_1, y_1) и (x_2, y_2) — три последовательные точки. **Gnuplot** проводит горизонтальную линию, соединяя точки с координатами

$$\left(\frac{x_0 + x_1}{2}, y_1\right) \quad \text{и} \quad \left(\frac{x_1 + x_2}{2}, y_1\right).$$

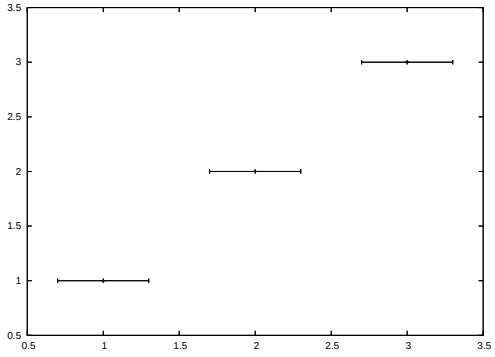
Затем проводятся вертикальные линии (до прямых $y = y_0$ и $y = y_2$). Для граничных точек горизонтальные отрезки проводятся таким образом, чтобы точки оказались в середине соответствующих интервалов.
Пример

```
gnuplot> plot "datafile.dat" with histeps
```

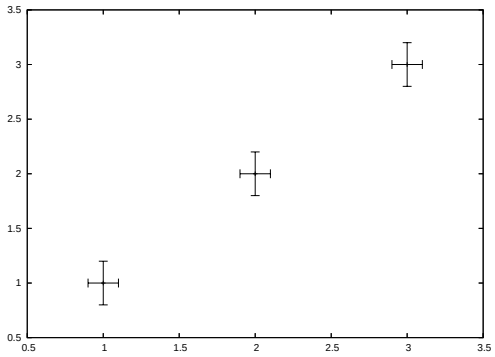


errorbars Другое название этого же стиля — `yerrorbars`. На месте каждой точки рисуется маленький кружочек, а затем проводится вертикальный отрезок. Для задания концов отрезка используется две формы: можно задать либо ширину отрезка, либо обе концевые точки.
В первом случае отрезок соединяет две точки $(x, y - \delta)$ и $(x, y + \delta)$, а во втором — точки с координатами (x, y_{low}) и (x, y_{high}). Вид определяется **Gnuplot** по количеству заданных данных: если заданы три колонки, то используется первый тип, если четыре — то второй.
Пример

```
gnuplot> plot [0.5:3.5] "datafile.dat" \
>notitle with errorbars
```



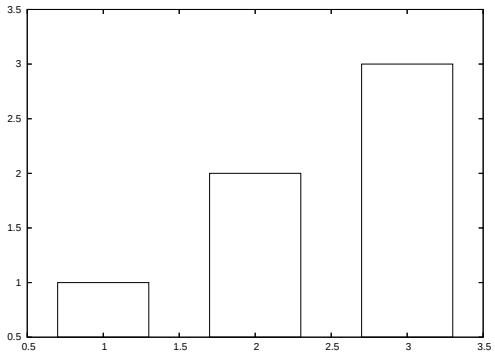
xerrorbars Стиль используется для рисования двумерных данных. Вокруг каждой точки рисуется два отрезка горизонтальных линий, характеризующих ошибки в измерениях. Формат задания данных точно такой же, что и в `errorbars`



xyerrorbars Этот стиль очень похож на предыдущий, только при его использовании вокруг точки строится два отрезка — как вертикальный, так и горизонтальный. Формат задания каждого из отрезком имеет точно такой же вид, что и в `errorbars`.

Пример

```
gnuplot> plot [0.5:3.5] "datafile.dat" \
>using 1:2:(0.1):(0.2) notitle\
>with xyerrorbars
```

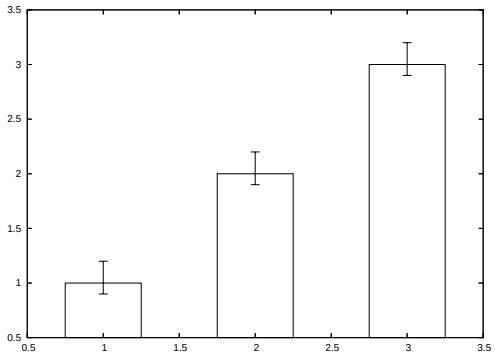


boxes Стиль используется только для представления двумерных данных. При его использовании на месте каждой точки рисуется прямоугольник, нижняя грань которого расположена на оси **x**, а верхняя совпадает с заданной ординатой точки.

Для определения ширины прямоугольника **Gnuplot** использует три различных алгоритма. Если при рисовании данных используются три столбца, то в качестве ширины прямоугольника берётся значение из третьего столбца. Если заданы только два столбца, то используется значение, установленное с помощью команды `set boxwidth`. Если же ни один их этих методов не доступен, то ширина каждого прямоугольника вычисляется таким образом, что соседние прямоугольники склеиваются между собой.

Пример

```
gnuplot> plot [0.5:3.5] [*:3.05] \
>"datafile.dat"\
>using 1:2:(0.95)
```
notitle with boxes

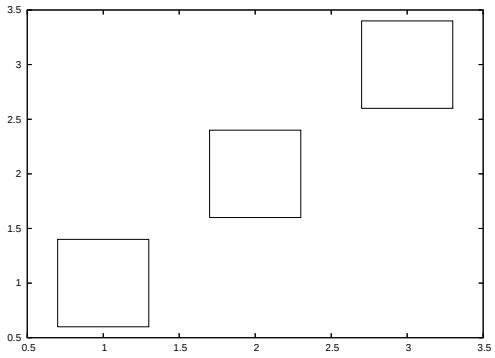


boxerrorbars Используется только для рисования двумерных данных. Стиль представляет собой комбинацию стилей `boxes` и `yerrorbars`.
Требуется пять колонок данных: первые две задают координаты точки графика, третья и четвёртая — границы ошибки. Пятая колонка определяет ширину прямоугольника.

Пример

```
gnuplot> plot [0.5:3.5] [0.3:*] \
>"datafile.dat" \
>u 1:2:($2-0.1):($2+0.2):(0.8) \
>notitle with boxerrorbars
```

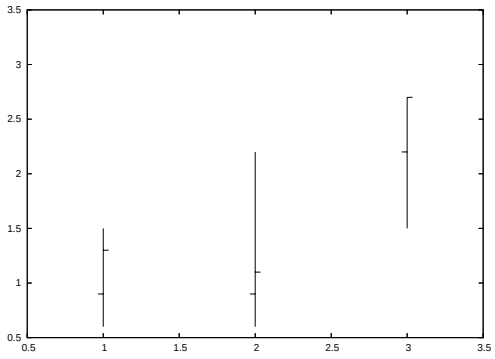
Обратите внимание, что прямоугольник строится с низа графика.



boxxyerrorbars Стиль является комбинацией `boxes` и `xyerrorbars`. При его использовании вместо точки рисуется прямоугольник, длин и ширина которого определяются величинами ошибок.

Пример

```
gnuplot> plot "datafile.dat" \
> u 1:2:($2-0.1):($2+0.1):($1-0.1):($1+0.2) \
> with boxerrorbars
```

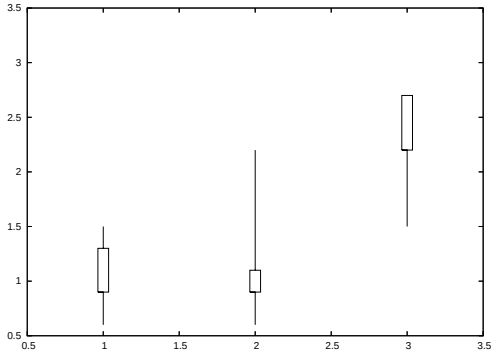


financebars Стиль используется только для представления двумерных данных финансовых данных (например, результатов торгов на бирже). Требуется пять колонок данных: первая колонка определяет значение **x** (обычно это дата), затем указываются значение открытия, минимальное, максимальное и закрывающее значения цен. При рисовании используется отрезок вертикальной линии, соединяющий максимальное и минимальное значение цен. Открывающее значение рисуется маленьким штрихом, направленным влево, а закрывающее — таким же штрихом, но направленным вправо. Длина штриха определяется параметром `bar` (директива `set bar`).

Пример

```
gnuplot>set bar 5
gnuplot>plot [0.5:3.5] "datafile.dat" \
>notitle with financebars
```

Обратите внимание на первую строчку — изменение длины штрихов. Установленное по умолчанию значение оказывается, на мой взгляд, слишком маленьким.

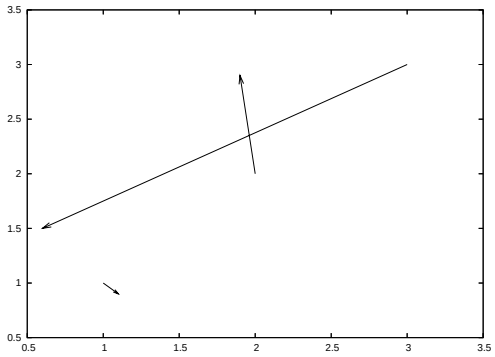


candlesticks Стиль используется для представления двумерных финансовых данных. Требуется пять столбцов чисел: координата **x** (обычно дата), открывающее, минимальное, максимальное и закрывающее значения цен. Для формирования графика используется открытый прямоугольник, центрированный относительно указанного значения **x** и ограниченный открывающей и закрывающей ценами. Между максимальным и минимальным значениями цен рисуется вертикальный отрезок. Ширина прямоугольника определяется значением параметра `bar`.

Пример

```
gnuplot> set bar 5
gnuplot> plot [0.5:3.5] "datafile.dat" \
>notitle with candlesticks
```

В первой строчке снова изменяется текущее значение параметра `bar`.



vectors Этот стиль может быть полезен для графического представления двумерных полей (например, поля скоростей). При его использовании рисуется вектор с началом в точке (x, y) и концом в $(x + \delta x, y + \delta y)$ (таким образом для применения стиля требуется четыре колонки данных). Кроме того, на конце вектора рисуется маленькая стрелка.
`gnuplot>plot "datafile.dat" with vectors`

8 Математические выражения

Математические функции в **Gnuplot**

`abs(x)`	Возвращает модуль аргумента. Для вещественных и комплексных аргументов результат будет вещественным, для целых — целым.
`acos(x)`	Вычисление $\arccos x$
`acosh(x)`	Вычисление гиперболического арккосинуса
`arg(z)`	Возвращает аргумент комплексного числа. Результат выдаётся в радианах или градусах (формат устанавливает директивой `set angles`)
`asin(x)`	Вычисление $\arcsin x$
`asinh(x)`	Вычисление гиперболического арксинуса
`atan(x)`	Вычисление $\operatorname{arctg} x$
`atan2(x,y)`	Возвращает арктангенс от отношения действительных частей `x` и `y`
`atanh(x)`	Вычисление гиперболического арктангенса
`besj0(x)`	Вычисление функции Бесселя первого рода порядка 0
`besj1(x)`	Вычисление функции Бесселя первого рода порядка 1
`besy0(x)`	Вычисление функции Бесселя второго рода порядка 0
`besy1(x)`	Вычисление функции Бесселя второго рода порядка 1
`ceil(x)`	Возвращает наименьшее целое число, большее `x`. Для комплексных аргументов результатом будет наименьшее целое число, большее действительной части `x`.
`column(x)`	Это выражение используется только при работе с файлами
`cos(x)`	Вычисление $\cos x$
`cosh(x)`	Вычисление гиперболического косинуса
`erf(x)`	Вычисление функции ошибок $\operatorname{Erf} x = \int_{-\infty}^{x} e^{-t^2} dt$ Если `x` — комплексное, то функция вычисляется только для действительной части
`erfc(x)`	Вычисление $1 - \operatorname{Erf} x$
`exp(x)`	Вычисление экспоненты
`floor(x)`	Возвращает целую часть `x`
`gamma(x)`	Возвращает $\Gamma(x)$
`ibeta`	
`igamma`	
`imag`	
`int`	
`inverf`	Вычисление функции, обратной к $\operatorname{Erf} x$
`invnorm`	
`lgamma`	
`log`	Вычисление натурального логарифма
`log10`	Вычисление десятичного логарифма
`norm`	Возвращает функцию нормального распределения
`rand`	Возвращает псевдослучайное число, равномерно распределённое на интервале $[-1, 1]$
`real`	Вычисление действительной части аргумента
`sgn`	Вычисление знака аргумента
`sin`	Вычисление синуса аргумента
`sinh`	Вычисление гиперболического синуса
`sqrt`	Вычисление квадратного корня
`tan`	Вычисление тангенса
`tanh`	Вычисление гиперболического тангенса
`tm_hour`	
`tm_mday`	

tm_min	
tm_mon	
tm_sec	
tm_wday	
tm_yday	
tm_year	
valid	

Бинарные операторы

Символ	**Пример**	**Значение**
**	5**3	возведение в степень
*	5*3	умножение
/	5/3	деление (результат операции зависит от аргументов)
%	a%b	остаток от деления первого аргумента на второй (оба аргумента должны быть целыми числами)
+	a+b	сложение
-	a-b	вычитание
==	a==b	сравнение двух чисел на равенство
!=	a!=b	сравнение двух чисел на неравенство
<	a<b	знак меньше
<=	a<=b	знак меньше или равно
>	a>b	знак больше
>=	a>=b	знак больше или равно
&	a&b	* побитное И
^	a^b	* побитное или
|	a|b	* побитное исключающее или
&&	a&&b	* логическое И
||	a||b	* логическое ИЛИ

Унарные операторы

Символ	**Пример**	**Значение**
-	-a	унарный минус
+	+a	унарный плюс
~	~a	* one's complement
!	!a	* логическое отрицание
!	a!	* вычисление факториала
$	$3	оператор взятия столбца[9]

В **Gnuplot** существует единственный оператор с тремя аргументами (так называемый тернарный оператор)

Символ	Пример
?:	a?b:c

Оператор работает точно так же, как и его аналог в C. Сначала вычисляется значение функции, указанной в качестве первого аргумента (это должно быть целым числом). Если оно ненулевое, то результатом

[9]Используется при работе с файлами

работы будет вычисленной значение второго аргумента. В противном случае возвращается вычисленное значение третьего аргумента.

Примеры

Предположим, что нам требуется нарисовать график функции $f(x)$, заданной следующим образом

$$f(x) = \begin{cases} \sin(x) & \text{при} \quad 1 \leqslant x \leqslant 2 \\ \frac{1}{x} & \text{при} \quad 2 < x \leqslant 3 \end{cases} \tag{1}$$

Получим требуемый график:

```
gnuplot> f(x) = 1<=x && x<=2 ? sin(x) : 2<= x && 3 ? 1/x : 1/0
gnuplot> plot f(x)
```

Для получения графиков тернарный оператор был использован дважды. При $x < 1$ и $x > 3$ значение функции не определено, поэтому на этом интервале функция принимает неопределённое значение. Единственный недостаток нашего графика — функция нарисована как непрерывная. Попробуем избавиться от этого.

```
gnuplot> f1(x) = 1<=x && x<=2 ? sin(x) : 1/0
gnuplot> f2(x) = 2<x && x<=3 ? 1/x : 1/0
gnuplot> plot f1(x) lt 1, f2(x) lt 1
```

Теперь график функции выглядит абсолютно правильно.

Аналогичным образом можно использовать оператор для рисования графиков данных