SCILAB

Глава 5. Графика

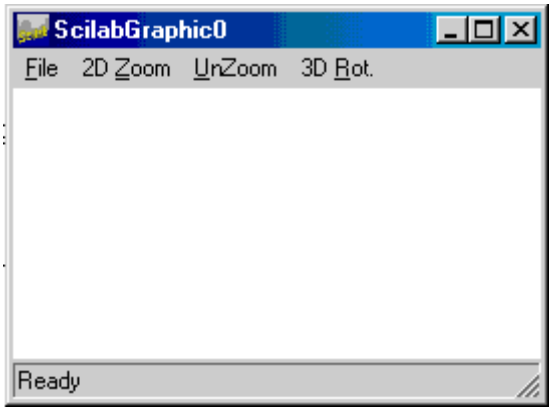
Это только краткое введение. Перечень всех команд графической библиотеки по разделам смотрите в файле manual.pdf документации Scilab (стр. 80-82).

Содержание главы:

- Изображение текста в графическом окне
    - Как изобразить текст в графическом окне?
    - Как узнать размер рамки (box), окружающей текст в команде string?
- Графические функции для решения линейных систем
- Интерактивное редактирование в графическом окне
    - Можно ли интерактивно (с помощью мыши) изменить (добавить или удалить) данные в графическом окне?
    - Как рисовать вручную в графическом окне простые графические объекты (прямоугольники, эллипсы и др.)?
    - Как из графического окна ввести данные в Scilab с помощью манипулятора мышь?
    - Как узнать текущее положение мыши?
- 3D графика
    - Как изобразить 3D поверхность?
    - Как построить 3D кривую параметрически заданной функции?
    - Как изобразить пространственную кривую в виде изолиний на плоскости?
    - Как изобразить 3D поверхность на 2D плоскости в виде заливки изообластей цветом?
    - Как нарисовать 3D гистограмму?
    - Построение смешанных графиков из 2D и 3D поверхностей
- Изображение нескольких рисунков в одном графическом окне
- Вывод содержимого графического окна на печать
    - Как получить бумажную версию графического окна?
    - Как внедрить Scilab графику в документ Latex?

Графическое окно

Графические приложения в пакете Scilab выводятся в отдельном окне, называемом графическим.
Графическое окно можно создать с помощью меню главного окна Scilab **Graphic Window "x" - Set (Create)Window**, где "x" является текущим номером графического окна. По умолчанию этот номер равен нулю и создается графическое окно с номером "0". Номер графического окна указан в его заголовке.



Допустимо открыть одновременно несколько графических окон, но активным из них является только одно, называемое текущим.

Если мы хотим создать графическое окно с другим номером окна, например "4", то мы должны вначале его пронумеровать с помощью команды меню **Graphic Window0-(increase current num)**, выполнив его столько раз, чтобы в меню главного окна вместо надписи **Graphic Window0** появилась надпись **Graphic Window4**. Затем создадим графическое окно с помощью меню **Raise (Create)Window** либо **Set (Create)Window**. В заголовке созданного графического окна тогда вместо `ScilabGraphic0` будет написано `ScilabGraphic4`.

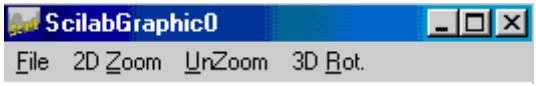
Заголовок текущего графического окна можно изменить с помощью команды **xname**.

Если вы хотите сделать активным, например, окно с номером 3, следует установить его номер в главном окне с помощью **Graphic Window0-(increase current num)**. Затем сделать его активным с помощью **Graphic Window "x" - Set (Create)Window**. Теперь оно стало текущим и все графические приложения будут изображаться в нем.

Команда **x=winsid()**, где x -вектор-строка, возвращает список номеров всех существующих окон.

Для очищения текущего графического окна используется команда меню **Graphic Window- Clear Current Window**.

В графическом окне есть своя панель меню:



Меню **File** имеет подменю и служит для работы с файлами: позволяет записывать содержимое графического окна в файл в родной формате **.scg** (**File - Save**), конвертировать в другие форматы(**File - Export**), загружать в окно картинки в формате **.scg** с диска (**File - Load**), печатать на принтере и др.
Команды меню **2D Zoom** и **UnZoom** служат для увеличения или уменьшения масштаба в графическом окне.
Команда меню **3D Rot** позволяет вращать 3D-объекты.

Как очистить графическое окно?

Способ 1.
С помощью меню **Graphic Window** - **Clear Current Window**.
Это аналог команды **xbasc()**.

Способ 2.
С помощью команды **xbasc**.
При очищении окна ассоциативная запись при этом тоже уничтожается.
Синтаксис
xbasc([window-id])
Параметры
window-id : целый скаляр или вектор, указывающий на номера тех окон, которые будут очищаться.

Без указания параметров очищается текущее окно. Команда `xbasc(1:3)` уничтожит изображение в окнах с номерами 1, 2 и 3. Команда `xbasc([1,3,5])` уничтожит изображение в окнах с номерами 1, 3 и 5. Если одно из этих окон не существует, оно будет автоматически создано, что является достаточно неожиданным. Уничтожается только изображение, сами окна продолжают существовать.
Пример.

```
plot(1:10)
xbasc() // Окно очищается
plot(1:5) // Изображается новый график в новых осях.
```

Способ 3.
С помощью команды **xclear**.
Синтаксис
xclear([window-id])
Параметры
window-id : целый скаляр или вектор, указывающий на номера тех окон ,которые будут очищаться.
Отличие этой команды от **xbasc**: окно очищается, а соответствующие команды записи остаются
В результате изображение графика, нарисованного до применения функции **xclear()** останется, а сверху наложится график, построенный после применения функции **xclear()**.

Графическая среда

В Scilab возможны различные графические устройства для засылки изображений в окна или на печатающее устройство. По умолчанию для вывода предназначено окно ScilabGraphic0.

Присутствуют следующие драйверы:

- **X11** вывод на экран компьютера
- **Rec** вывод на экран компьютера с записью всех графических команд. Этот драйвер используется по умолчанию.
- **Wdp**
- **Pos** - вывод в формате Postscript
- **Fig** - вывод в формате Xfig
- **GIF** - вывод в формате GIF

Замечания: Драйверы Fig и GIF могут иметь некоторые дефекты изображения и в некоторых прикладных программах могут не работать. Драйвер Xfig служит для образования Postscript файлов для дальнейшей вставки в документ на языке Latex.

Базовые команды для управления Scilab-графикой:

- **driver** -выбор графического драйвера
- **xclear** - очищение одного или более графических окон
- **xbasc** - очищение одного или более графических окон и уничтожает записей
- **xpause** - задание пауза в миллисекундах
- **xselect** - увеличение графического окна (для X-драйверов)
- **xclick** - ожидание щелчка мыши
- **xbasr** - перерисовка графического окна
- **xdel** - уничтожение графического окна (эквивалентно команде меню **Close**)
- **xinit** - инициализация заданного драйвера
- **xend** - закрытие графической сессии

На самом деле можно обойтись без регулярных драйверов при использовании экранного драйвера, пользуясь функциями **xbasimp**, **xs2fig**, чтобы посылать графику для Postscript принтера или для Xfig-систем.
Пример.

```
plot(1:10) // Это пример простейшего графика
xbasimp(0,"d:/zzz.ps") // 0 означает здесь номер текущего графического окна
```

В результате создан файл с именем zzz.ps.0
Это эквивалентно конструкции

```
driver("Pos")
xinit("d:/zzz.ps.0")
plot(1:10)
xend()
```

Замечание: К сожалению, ***.ps** формат непонятно как использовать: мои стандартные программы (например, GsView) его игнорируют. Формат же ***.gif**, видимо, приемлим для многих программ. В таком виде картинку можно сохранить из графического окна с помощью меню **File-Export** и далее выбор в окошке **Export Type** формата **GIF**.

Образованный файл понятен для текстового редактора Word, Photoshop и explorer, но не понятен, например, для верстальной программы PageMaker.

Эквивалентная командная запись:

```
driver("GIF")
xinit("d:/yyy.gif")
plot(1:5)
xend()
```

Замечания:
1)В команде **driver("GIF")** слово **GIF** должно быть набрано обязательно большими буквами.
2) Если Вы не выполнили завершающую команду **xend()**, то будет создан нечитаемый для приложений файл.
3) Графическое окно при применении **driver("GIF")** на экране компьютера не создается.

Глобальные параметры графики

Как установить пользовательские графические значения контекста?

Команда **xset** устанавливает графические значения контекста в текущем окне (установка шрифтов, цветов, размеров и т. д.). Для того, чтобы эта команда могла работать, предварительно должно быть выполнена установка моды **X11** либо **Rec**, например, с помощью **driver("X11")**.

Синтаксис
xset(choice-name,x1,x2,x3,x4,x5)
xset()
Параметры
choice-name : ключевая символьная строка. Она может принимать только определенные значения. Допустимые значения смотри ниже.
x1,...,x5 : значение этих параметров зависят от значения первого параметра команды (choice-name). Все возможные значения параметра **choice-name** и соответствующие ему значения параметров **xi** перечислены в help.

Без параметров команда **xset()** предложит выбрать из меню желаемые значения следующих параметров:
1)**fontld** - название шрифта
2)**fontsize** - размер шрифта
3) **markld** - какими символами будут отмечены точки на графике
4) **marksize** - размеры символов
5) **Thickness** - толщина линий
6) **pixmap/flag** (On или Off) - флаг чего-то
7) **use color** - использование цвета
8) **colors** - количество цветов
9) **alufunction** - задание соответствующей моды изображения пикселей в графическом режиме (смотри **help alufunctions**)

Замечание: Похоже на то, что команда **xset** с параметрами не дублирует полностью ручные установки **xset()**, а предоставляет гораздо более широкие возможности пользователю.

Рассмотрим некоторые примеры использования команды **xset**.
Пример 1.
```
color=5;
xset("foreground",color);
plot(1:10)
```
Устанавливается цвет переднего плана (foreground) для текущего окна в зависимости от текущего значения параметра **color**. В результате мы получили красные оси **y** графика (им соответствует значение "5") и черную линию графика.

Соответствие между значениями параметра **color** и цветами:
0 и 1- черный цвет
2- синий цвет
3- зеленый цвет
4- ярко-голубой цвет (cyan)
5- красный цвет
6- ярко-лиловый цвет (pink)
7- красный цвет другого оттенка
8- белый цвет
9, 10 и 11 - разные оттенки темно-синего цвета
12- бледно-голубой цвет
13, 14, 15- разные оттенки темно-зеленого цвета
и так далее (можно посмотреть экспериментально)
Устанавливает цвет переднего плана (foreground) для текущего окна.

Если хотим узнать, что соответствует значению "color" в терминах RGB, можно посмотреть это с помощью команды
```
my_map=xget("colormap")
```
В результате мы увидим большую матрицу. Ее размер можно определить командой:
```
size(my_map)
```
Результат:
```
ans =
! 32. 3. !
```
Это означает, что в нашей палитре установлено 32 цвета. Каждая строка матрицы означает разложение соответствующего цвета в цветовой модели RGB. Для тех, кто забыл: модель RGB (Red, Green, Blue) базируеся на смешении различных значений красного, зеленого и синего цвета. Номер строки соответствует значению параметра "color". Содержание строки под номером "5" соответствует R=1, G=0 и B=0, что соответствует красному цвету. С помощью команды **getcolor()** можно увидеть цветовую раскладку в новом графическом окне в виде цветных квадратиков, на каждом из которых нарисован соответствующий номер цвета.

Пример 2.
```
xset("background",4)
```
Устанавливает цвет заднего плана (background) в текущем графическом окне. Он будет определяться содержанием 4-той строки матрицы текущей палитры. Это голубой (cyan). Если графическое окно не было создано, создает графическое окно с фоном голубого цвета.

Пример 3.
```
xset("color",value)
```

Устанавливает цвет по умолчанию для заполнения, изображения текста или рисования линий (но не графиков!). Значение параметра value есть целое число, проектируемое в интервал [0,whiteid]. 0 используется для заполнения черным и стирания (whiteid) белым. Значение whiteid может быть получено с помощью команды **xget("white")**.

```
value=5;
xset("color",value); titlepage("Cat") // Вывод надписи "Cat" красными буквами
```

Пример 4.

```
xset("colormap",cmap)
```

Устанавливает собственную карту цветов в виде матрицы с именем "cmap", состоящей из m строк и 3-х столбцов, где m -число цветов. Цвет с номером "i" задается тремя элементами матрицы cmap(i,1), cmap(i,2) и cmap(i,3). Они соответствуют в указанном порядке красной, зеленой и синей интенсивности между 0 и 1.
Примечание: Матрица с именем `cmap` должна быть создана до исполнения команды **xset("colormap",cmap)**. Все элементы этой матрицы должны принимать значения в интервале [0,1].

Пример 5.

```
xset("use color",flag)
```

Устанавливается цветная или черно-белая мода. Для цветной моды flag=1. В этом случае команды **xset("pattern",.)** или **xset("dashes",.)** будут использоваться для изменения цвета по умолчанию для рисования или для шаблона заполнения. Если flag=0, то это - черно-белая мода и мы можем использовать серые тона и черные пунктирные линии.

Пример 6.

```
xset("dashes",i)
```

В черной-белой моде команда **xset("use color",0)** устанавливает стиль линии (dash style) равный стилю под номером i. Эта мода установлена по умолчанию.
В цветной моде команда **xset("use color",1)** устанавливает цвет линии, цвет выделения точек и текста. Ключевое слово абсолютно, желательно использовать вместо него команды **xset("color",i)** или **xset("line style",i)**.

```
xset("line style",5)
plot(2:10)
```

Пример 7.

```
xset("auto clear","on"|"off")
```

Выбор включения и выключения моды автоматической очистки для графики. Если **"auto clear"** мода включена (`xset("auto clear","on")`), то последующие графики не накладываются, то есть при выполнении команды Конструкция действует аналогично команде **xbasc()**: графические окна очищаются и ассоциативные записи стираются перед каждым уровнем графической функции. Значение по умолчанию **"off"**. Чтобы узнать текущее состояние, воспользуйтесь командой **xget("auto clear")**.

```
xset("auto clear","on");
x=(1:5);
plot(1:10);
plot(x^2)
```

В результате мы видим только изображение параболы.

```
xset("auto clear","off");
x=(1:5);
plot(1:10);
plot(x^2)
```

В результате мы видим изображение и прямой, и параболы.

Пример 8.
```
xset("default")
```
Возвращает все графические установки контекста к значениям по умолчанию.

Пример 9.
```
xset("font",fontid,fontsize)
```
Устанавливает текущие шрифты для графического окна и их размер. Это же можно выполнить в графическом режиме с помощью команды **[fId,fSize]=getfont()**. Выбор шрифта и размера тогда осуществится визуально, и внизу графического окна будут видны соответствующие им значения `fontid` и `fontsize`. Дальше их можно применять в других командах.

```
xset("font",1,4)
titlepage("abcdefghijklmnop") // Получим надпись греческими буквами
```
Можно сделать надпись на русском языке:
```
xbasc(); // очищение экрана
xset("font",5,5);
titlepage("Это на русском языке!");
```

Пример 10.
```
xset("font size",fontsize)
```
Устанавливает размер шрифта.

Пример 11.
```
xset("fpf",string)
```
Устанавливает формат чисел (floating point) для изображения их в графическом окне (аналогично синтаксису языка C). Например, `string="%.3f"`. Для возврата к значению по умолчанию используйте `string=""`.

Пример 12.
```
xset("hidden3d",colorid)
```
Устанавливает номер цвета для невидимых (спрятанных) сторон в режиме **plot3d**. В случае `colorid=0` подавляется рисование задней стороны объекта для 3d-объектов. Технически это называется "выбраковка" и используется для замкнутых пространств.

Пример 13.
```
xset("line mode",type)
```
Функция используется для установки моды рисования линий. Абсолютная мода устанавливается со значением `type=1` и относительная - со значением `type=0`. Говорят, что мода `type=0` содержит ошибки.

Пример 14.
```
xset("line style",value)
```
Устанавливает текущий стиль линии. Для сплошной линии `value=1`, для штрихованных линий `value>1`.

Пример 15.
```
xset("mark",markid,marksize)
```
Устанавливает текущий маркер и текущий размер маркера. Используйте **xset()** для того, чтобы посмотреть, какие маркеры возможны. Чтобы узнать его текущее значение используйте команду **mark=xget("mark")**. Полученное значение **mark=[markid,marksize]**.

Пример 16.

```
xset("mark size",marksize)
```
Устанавливает размеры маркера.

Пример 17.
```
xset("pattern",value)
```
Устанавливает текущий шаблон для функции закраски. Параметр `value` - может принимать целое значения, спроектированное на интервал `[0,whiteid]`, где `whiteid` - число цветов в палитре. Значение 0 используется для заполнения черным и стирания (whiteid) белым. Значение `whiteid` может быть получено с помощью команды **xget("white")**. Параметр **"pattern"** эквивалентен параметру **"color"**.

Пример 18.
```
xset("pixmap",flag)
```
Если `flag` равен 0, то графика изображается непосредственно на экране. Если `flag` равен 1, то графика исполняется в формате карты "pixmap" и посылается в дальнейшем в графическое окно с помощью команды **xset("wshow")**. Карта "pixmap" очищается с помощью команды **xset("wwpc")** или с помощью команды **xbasc()**. Команде **xset("pixmap",1)** аналогичны команды **xset("wshow")** и **xset("wwpc")**.

Пример 19.
```
xset("thickness",value)
```
Устанавливает толщину линий в пикселях. Значения `value`, равные 0 и 1, означают одну и ту же толщину: 1pixel. Чтобы узнать текущую толщину линии воспользуйтесь командой **xget("thickness")**.
```
xset("thickness",2);
plot(1:10)
```
Теперь изменим толщину линий:
```
xset("thickness",5);
plot(5:7)
```

Пример 20.
```
xset("wdim",width,height)
```
Устанавливает ширину (`width`) и высоту (`height`) текущего графического окна. Значения `width` и `height` задаются в абсолютных единицах экрана. Если Ваш монитор настроен на разрешение 1024x768, то это и есть максимально возможное графическое окно. Реально следует задавать окно несколько меньших размеров.
```
w=500;
h=350;
xset("wdim",w,h)
```
Для того, чтобы узнать размер текущего окна выполним
```
dim=xget("wdim")
dim =
! 500. 350. !
```

Замечания:
1) Эта опция не используется для драйверов типа Postscript.
2) По-видимому, существуют внешние ограничения на минимально и максимально возможный размер окна. Если пользователь нарушил эти ограничения, Scilab подменяет размер (размеры) заданные пользователем на свои. Об истинных размерах окна можно судить опять-таки с помощью команды **xget("wdim")**. Моя версия: минимальный размер окна лимитирован возможностью размещения в панели управления графического окна.
3) Можно изменить с помощью мышки размер окна и узнать его новые размеры с помощью команды **xget("wdim")**. Таким же способом можно узнать размер окна по умолчанию.

Пример 21.
```
xset("viewport",x,y)
```
Устанавливает положение просмотрового окна (panner). По умолчанию просмотр установлен в положение, соответствующее x=0 и y=0.

Пример 22.
```
xset("wpdim",width,height)
```
Устанавливает ширину и высоту текущего физического графического окна (которое может быть отлично от реального размера в моде "wresize 1"). Эта команда не является столь глобальной, как команда **xset("wdim",width,height)**. Не используется для драйверов типа Postscript.

Пример 23.
```
xset("window",window-number)
```
Устанавливает в качестве текущего окна окно с номером "window-number" или создает такое окно, если ранее оно не существовало.
```
xset("window",0) // создается окно с номером 0
xset("window",1) // создается окно с номером 1
xset("window",4) // создается окно с номером 4
```

Пример 24.
```
xset("wpos",x,y)
```
Устанавливает положение верхней левой точки графического окна. Координаты x и y заданы в абсолютных единицах экрана.
```
xset("wpos",250,0)
```
Будет построено окно, с координатами левого верхнего угла (250,0). Для определения текущих координат окна служит комнада **xget("wpos",x,y)**.

Пример 25.
```
xset("wresize",flag)
```
Если flag=1, то изображение в графическом окне автоматически перерисовывается для наилучшего заполнения графического окна. Этот флаг реализуется по умолчанию. Для определения текущего состояния флага служит команда **xset("wresize")**.

```
xdel(); // уничтожение окон
xset("wresize",1);
xset("wdim",200,200);
plot(1:10)
```
После просмотра результата измените размер окна:
```
xset("wdim",500,500) // изменение размеров текущего окна
```
В результате графическое окно увеличилось и соответственно увеличился график.
Если flag=0, то шкала графика слева остается неизменной при изменении размеров графического окна. Просмотреть все изображение можно при помощи полосы прокрутки (scroll) или развернув окно с помощью стандартного инструмента раскрытия окна в правом верхнем углу.
```
xdel();
plot(1:10);
xset("wresize",0);
```
После просмотра результата измените размер окна:
```
xset("wdim",1000,500)
```
Шкала графика в левой части не изменилась.

Пример 26.
```
xset("clipping",x,y,w,h)
```
Устанавливается зона обрезания (clipping zone) (зона графического окна, где могут рисоваться графики) как прямоугольник с размерами (x,y,w,h), где "x" и "y" - координаты

верхнего левого угла, а "w" и "h" - высота и ширина окна. Эти параметры задаются в текущих координатах графика.

Пример 28.
```
xset("alufunction",number)
```
Используется для установки логической функции для рисования. Обычные значения полей `number` следующие: 3 -для копирования (по умолчанию), 6 -для анимации и 0 -для стирания. Для большей детализации смотри команду **alufunctions** .

Как узнать установленные графические значения контекста?

Для получения текущей информации о графическом контексте используйте команду **xget**. Она является обратной команде **xset**.
Синтаксис
[x1]=xget(str,[flag])
xget()
Параметры
str : строковая переменная, могущая принимать только определенные значения. Возможные значения смотри в help.
flag : Устанавливается равным 1 для пространственной моды. Этот параметр не является обязательным.

При наличии аргумента **x1** команда возвращает значение выбранного аргумента **str**, иначе значение возвращается переменной с именем **ans**.
Без аргументов команда **xget()** показывает полное меню текущих графических элементов, которые могут быть изменены с помощью сервисных кнопок значений. Это не полный список всех возможностей **xget** в командном режиме с указанием параметра **str**.

Пример 1.
```
plot(1:10)
xset("background",3)
```
Наш график прямой линии оказался на зеленом фоне. Ему соответствует цвет под номером "3". Проверим это, получив эту величину в строковую переменную `my_color`.
```
my_color=xget("background")
```
Результат
```
my_color = 3.
```
Пример 2.
```
xset("color",5)
 nn=xget("color")
```
Результат:
```
nn =
5.
```
Полное перечисление всех полей смотри с помощью команды **help xget**. Они такие же, как и у команды **xset**. С помощью **help xset** Вы получите больше информации. Некоторые примеры применения **xget** рассмотрены в разделе описания функции **xset** этого руководства.

Как установить шрифты в графическом окне?

Способ 1.
С помощью команды **xlfont**.
Синтаксис

xlfont(font-name,font-id)
fonts=xlfont()
Параметры
font-name : строка, являющаяся названием семейства шрифтов.
font-id : целое число, принимающее значения от 0 до 6. Это идентификатор для загрузки фонта с именем **font-name**.
fonts : вектор-столбец текущих загруженных имен шрифтов.

Без аргументов **xlfont()** возвращает список текущих загруженных шрифтов и используется для загрузки нового шрифта другого размера. Допустимые размеры шрифтов: 8, 10, 12, 14, 18 и 24.

Шрифты по умолчанию: "Courier Roman" (0), "Symbol" (1), "Times Roman" (2), "Times Italic" (3), "Times Bold" (4) and "Times Bold Italic" (5).

Имя шрифта (**font-name**) может быть 2 типов:
1) Если содержит знак "%", оно сопоставляется с X11 именем фонта с %s в размере поля в имени, для примера "-b&h-lucidabright-demibold-r-normal--%s-*-75-75-p-*-iso8859-1"
2) Если не содержит знак "%", оно сопоставимо с загрузкой фонтов font-name08,...,font-name24.

Пример.
```
xlfont("Times Bold",4)
titlepage("Cat")
```
Замечание: В версии для Windows, по-видимому, содержится ошибка: у меня выполнение конструкции `f=xlfont()` приводит к аварийному завершению работы Scilab.

Способ 2.
Установить фонты в рамках общей функции **xset**. Смотри описание этой функции выше.

Способ 3.
Использовать наглядный способ установки с помощью команды **[fId,fSixe]=getfont()**. В результате выполнения этой команды мы получим возможность устанавливать шрифты из графического окна с помощью манипулятора "мышь". Установив нужный шрифт, выберите в строке меню **FIle-OK**.
Пример.
```
[fId,fSize]=getfont()
```
Выберите шрифт с помощью мыши и продолжите:
```
xset("font",fId,fSize) // Не забывайте про эту команду!
titlepage("AbCs")
```

Некоторые простые команды для манипуляций с графиками

isoview - изометрическое шкалирование (масштабирование) без изменения окна
square - изометрическое шкалирование (масштабирование) с изменением окна
scaling - шкалирование (масштабирование) данных
rotate -вращение
xgetch, **xsetech** - изменение масштаба внутри графического окна. Подробно смотрите в Help.

2D графика

Как построить простейший одномерный график y=f(x)?

С помощью команды **plot**.
Синтаксис
plot(x,y,[xcap,ycap,caption])
plot(y)
Параметры
x,**y**: два вектора одинакового размера
xcap,**ycap**,**caption**: строки или строковые матрицы

Команда **plot** изображает параметр **y** как функцию от параметра **x**. Если **y** задано в виде формулы от **x**, то первый параметр в формуле можно опустить. Параметры **xcap** и **ycap** являются соответствующими наименованиями осей координат графика **x** и **y**. По умолчанию у меня значение параметра **ycap** находится сверху справа от вертикальной оси, а значение параметра **xcap** находится сверху справа от конца горизонтальной оси, что не очень красиво. Параметр **caption** является заголовком графика. У меня он пишется сверху над графиком. Смотри пример 4(ниже). Первый аргумент **x** может быть опущен, если **y** задан как функция от **x** и является вектором. В противном случае используйте функцию **plot2d**.

Напомним способы задания векторов:
1) Можно задать аргумент тремя значениями: минимальным, шагом и максимальным значением: **a=(amin:step:amax)**

```
x=(2:0.2:3);
```

Можно определить аргумент и без круглых скобок:

```
x=0:0.1:2*%pi;
```

2)Указать все значения:

```
x=[2 2.2 2.4 2.6 2.8 3.0];
```

Пример 1.

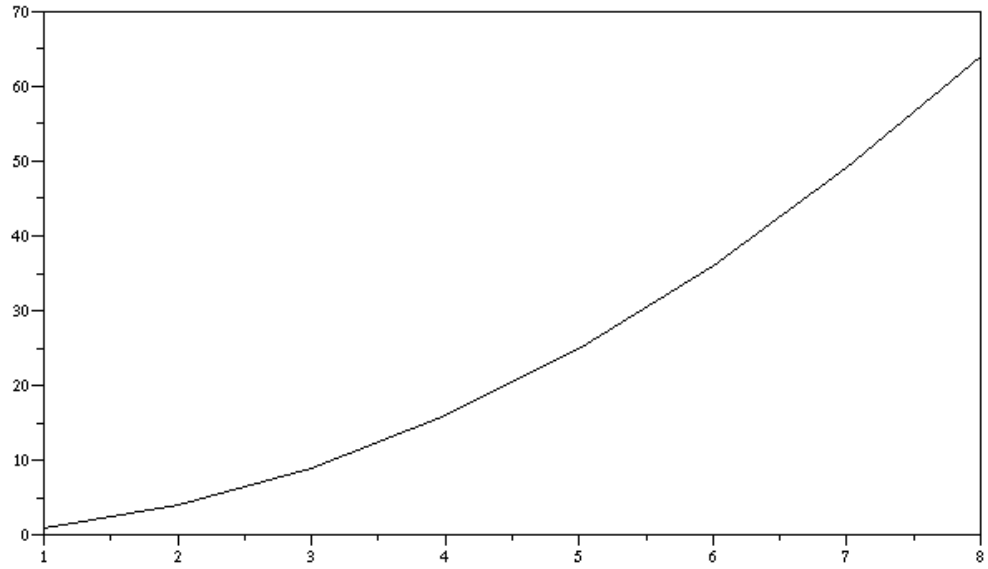
```
x=[1 2 3 4 5 6];
y=x^2;
plot(y)
```

Замечание: Поскольку аргумент `x` функции `y` задан неявно, то использование команды **plot** с одним параметром все же нежелательно.

Пример 2.
Пусть мы хотим построить кривую зависимость y[i] от x[i]. Пусть эти значения заданы экспериментально. Введем их для примера сами.

```
x=[1 2 3 4 5 6 7 8];
y=[1 4 9 16 25 36 49 64];
plot(x,y)
```

В результате получим график параболы.
Результат:



Пример 3.
Пусть мы имеем файл `d:/my/my_file.dat` экспериментальных данных, записанных в формате ASCII-кодов, в виде нескольких столбцов (не меньше трех). Для выполнения примера создайте его сами в текстовом редакторе. Число строк в файле данных равно N, но нам его знать не обязательно. Данные будут считываться с помощью оператора **read**. Хотим считать только первых три столбца из файла.

```
my_dat= read("d:/my/my_file.dat", -1, 3)
// Параметр -1 означает, что мы не знаем сколько строк в файле данных
my_dat =
! 1. 1.  6.  !
! 2. 4.  9.  !
! 3. 9.  14. !
! 4. 16. 21. !
! 5. 25. 30. !
! 6. 36. 41. !
plot(my_dat(:, 1), my_dat(:, 2));
```

Замечание: Цифра "3" в команде **read** означает, что мы считываем из каждой строки по три числа. На самом деле число столбцов в файле данных может быть больше (но не меньше!!!). `my_dat(:, 2)` означает второй столбец файла данных. `my_dat` - это матрица из трех столбцов и N строк. Использование **plot** для этого примера возможно, но нежелательно.

Пример 4.

Пусть значение вектора **x** задано в виде дискретных точек, а **y** задано в виде аналитической зависимости от **x**:

```
x=[1 2 3 4 5 6 7 8];
y=(x+1)^2-x+7;
plot(x,y)
```

В результате получим график параболы.

Пример 5.
Формирование заглавия и наименований осей координат.

```
x=0:0.1:2*%pi;
plot(x,sin(x),"sin","time","plot of sinus")
```

Пример 6.
Два графика в одних осях координат.

```
x=0:0.1:2*%pi;
plot(cos(x));
plot(sin(x))
```

или правильнее (и тогда они будут разного цвета) использовать конструкцию:

```
x=0:0.1:2*%pi;
plot([sin(x);cos(x)])
```

Замечание: Функцию от одного аргумента можно (и лучше) изобразить с помощью команды **plot2d**. Ее описание смотри ниже.

Как напечатать рисунок из графического окна?

Способ 1.
Выполнение с помощью строкового меню графического окна **File-Print(Scilab)** и далее не приводит к желаемому результату печати на принтер, по крайней мере в Windows98,(вместо этого печатается какой-то программный текст). Однако, если в качестве принтера у Вас установлен Acrobat Disltiller, то можно создать полноценный pdf-файл изображения в графическом окне. По умолчанию создается файл с именем scilab-0.pdf для графического окна с номером "0" и т.д. Для печати в Windows можно пользоваться меню **File-Print(Windows)**, где разрешены форматы бумаги **Portret** и **Landscape**. При этом печатаемое графическое окно деформируется, чтобы полностью заполнить выбранный формат.

Способ 2.
Можно поместить изображение в буфер обмена (Clipboard) с помощью команд меню управления графического окна **File-Copy** to **Clipboard(EnhMetafile)** или **File-Copy to Clipboard(Metafile+DIB)** и затем вставить его из буфера в какую-либо стандартную программу. В случае применения **Clipboard(EnhMetafile)** изображение в буфере идентично тому, которое мы копировали, а в случае **Clipboard(Metafile+DIB)** удалены почему-то подписи к делениям по осям (по-видимому, это ошибка в пакете?). Буфер обмена - удобное средство для быстрого обмена информацией между разными программами. Для Windows подходит режим **Clipboard(EnhMetafile)**. Это векторный формат. Формат **DIB** - видимо Clipboard в растровом формате. Для интересующихся: Смотри обзор формата DIB.(http://delphi.mastak.ru/articles/dib/) Для чего хорош Clipboard в формате DIB я не знаю...

Способ 3.
С помощью команд меню **File-Export** можно экспортировать изображение графического окна в форматы Postscript, Postscript NoPream, Postscript-Latex, Xfig и GIF. По моему мнению, наиболее удобен формат GIF, все разновидности форматов Postscript

большинство программ Windows сочло непонятными для себя.

Замечания:
а) Почему-то после печати на принтере, а иногда после загрузки в буфер картинка из графического окна частично или полностью стирается и не хочет восстанавливаться с помощью меню **Redraw** (может быть это ошибка операционной системы и лично моего компьютера?).

б) В некоторых случаях перенос с помощью буфера обмена Clipboard дает результат несколько отличный от того, что мы видит в графическом окне. Попробуйте выполнить следующую комбинацию:

```
x=[0:0.1:2*%pi]';
plot2d(x,[sin(x) sin(2*x) sin(3*x)],1:3,"011"," ",[0,0,6,0.5])
```

Теперь запишите ее в какой-нибудь текстовый редактор с помощью Clipboard(EnhMetafile). Как ни странно, оказывается, что в текстовом редактора картинка шире, чем была в Scilab и содержит больше информации!!!

Как трансформировать систему координат?

Способ 1.
С помощью команды **isoview**.
Команда **isoview** устанавливает шкалу (масштаб) по осям для изображения графика в прямоугольной системе координат. При этом размер самого графического окна не меняется.

Синтаксис
isoview(xmin,xmax,ymin,ymax)
Параметры
xmin,**xmax**,**ymin**,**ymax** : действительные числа, означающие минимальные и максимальные значения осей координат.

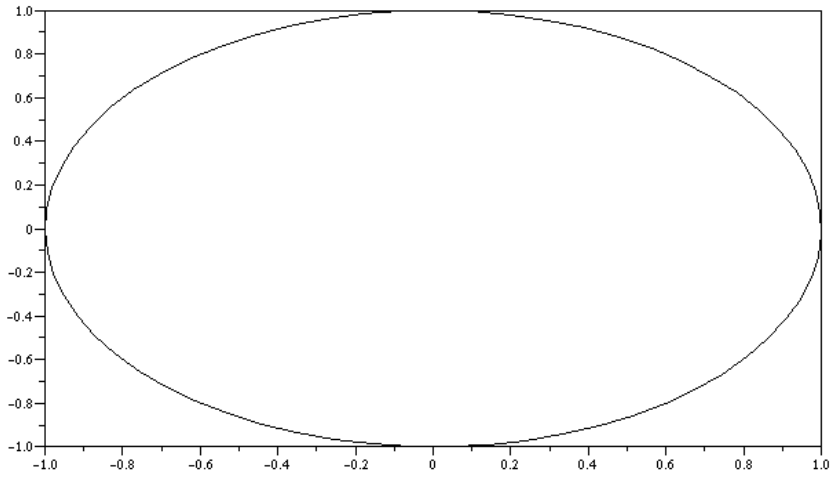
Эта функция абсолютна, используется предпочтительно с опцией **frameflag=4** для функции **plot2d**. Подробно смотри **help isoview**.

Замечание: Если области определения и принимаемых значений выводимого графика окажутся шире области, выбранной с помощью **isoview**, то эта область автоматически расширяется. (Лично я такую самодеятельность не одобряю).

Пример.

```
t=(0:0.1:2*%pi);
plot2d(sin(t),cos(t));
```

В результате будет нарисован эллипс в системе координат x=[-1,1] и y=[-1,1]. Система координат выбирается автоматически для максимально возможного масштаба представления данных.
Результат:



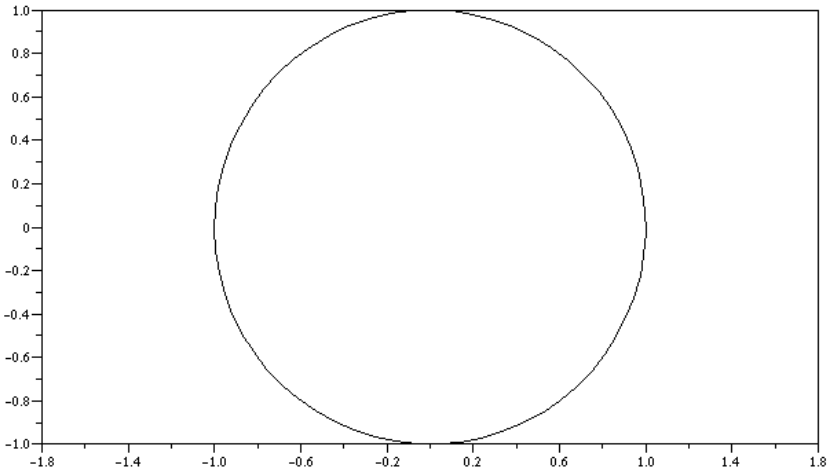

```
xbasc()
```

Все стерли из графического окна.

```
isoview(-1,1,-1,1);
plot2d(sin(t),cos(t))
```

В результате будет нарисован эллипс в системе координат x=[-1.8, 1.8] и y=[-1, 1]

Результат: Рисунок



Способ 2.
С помощью команды **square**.
Команда **square** устанавливает шкалу (масштаб) по осям для изображения графика в прямоугольной системе координат и изменяет при этом размер самого графического окна.
Синтаксис
square(xmin,xmax,ymin,ymax)
Параметры
xmin, **xmax**, **ymin**, **ymax** : действительные числа, означающие минимальные и максимальные значения осей координат.
Значения параметров **xmin**, **xmax**, **ymin**, **ymax** являются границами графической рамки (frame).

Пример.
```
t=[0:0.1:2*%pi]';
plot2d(sin(t),cos(t))
xbasc()
square(-1,-1,1,1)
```
После этой операции окно станет квадратным. Выполните еще раз команду построения графика и убедитесь.
```
plot2d(sin(t),cos(t))
```

Способ 3.
С помощью команды **xsetech** создания дочернего (sub-window)графического окна.

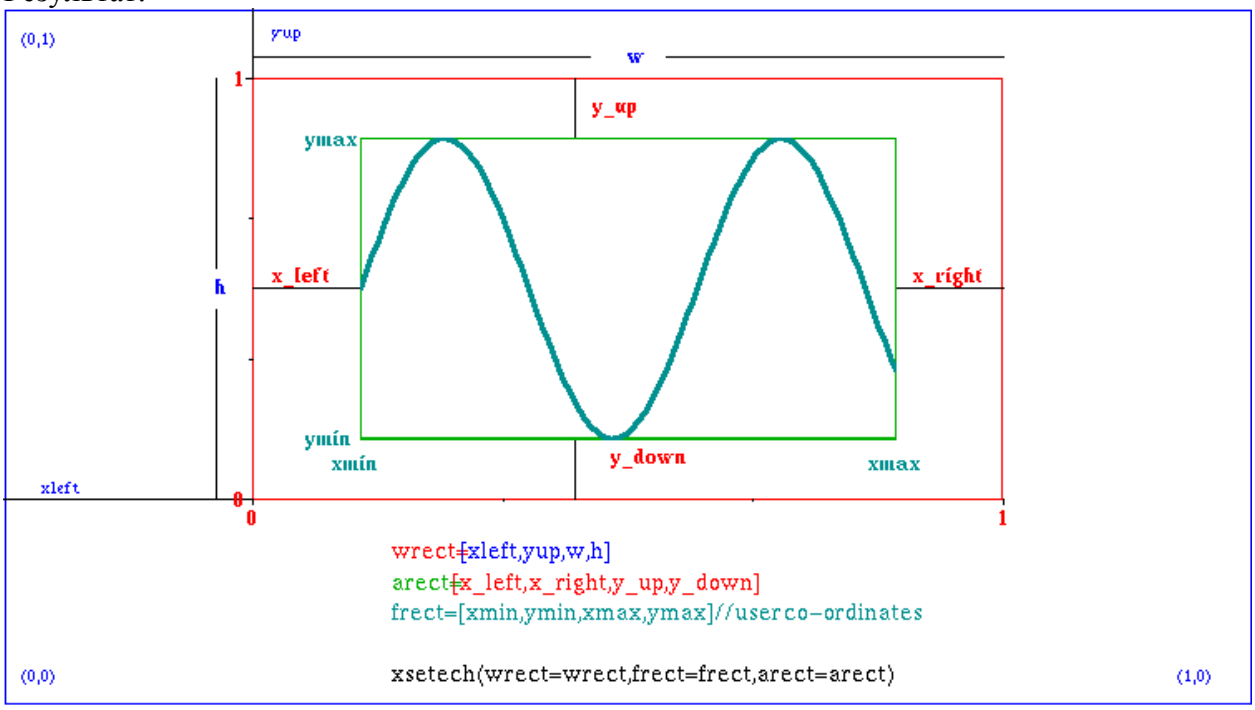
Синтаксис
xsetech(wrect,[frect,logflag])
xsetech(wrect=[...],frect=[..],logflag="..", arect=[...])
xsetech()
Параметры
wrect : вектор размерности =4, определяющий размеры дочернего окна.
frect :вектор размерности =4.
logflag : строка размером 2 "xy", где x и y могут принимать значения "n" либо "l". "n" и "l" означают, соответственно, нормальную и логарифмическую шкалы. x соответствует x-оси

и y - y-оси.

arect : вектор размерности =4.
Дочернее окно задается с помощью параметра **wrect=[x,y,w,h]**, где `x` и `y` - координаты левого верхнего угла, ширина и высота создаваемого окна в относительных единицах к размеру главного окна. Это значит, что значение параметров **wrect** определяют пропорции ширины и высоты дочернего окна относительно текущего окна. Например, **wrect=[0,0,1,1]** означает, что дочернее окно будет совпадать с главным. **wrect=[0.5,0,0.5,1]** будет означать, что областью дочернего окна будет правая половина главного окна.
Параметр **frect=[xmin,ymin,xmax,ymax]** используется для задания шкалы аналогично команде **plot2d**. Если **frect** не задано, графическая шкала не меняется.
arect=[x_left, x_right,y_up,y_down] используется для установления "полей" внутри дочернего окна. Единицы измерения для **x_left**, **x_right**, **y_up** и **y_down** как и для параметра **wrect** относительны и являются пропорциями от ширины и высоты рамки (frame) текущего графического дочернего окна. Значение по умолчанию 1/8*[1,1,1,1]. Если **arect** не задано, текущее значение не меняется.

Пример.
Наглядную картинку с изображением всех параметров можно получить запустив:
```
exec('SCI/demos/graphics/xsetechfig.sce');
```
Файл написан в ASCII кодах, поэтому желающие могут ознакомится с его содержимым.

Результат:



Как нарисовать двумерный график?

С помощью команды **plot2d**. Обычно применяется для линейных графиков. Для уничтожения предыдущего содержания окна используйте команду **xbasc()**. Посмотрите

демонстрационный пакет **plot2d()**

Существуют аналогичные функции более высокого уровня:
plot2d1 эквивалентна команде **plot2d**.
plot2d2 : аналогична команде **plot2d**, но выглядит как гистограмма без вертикальных делений на прямоугольники (огибающая гистограммы). Является кусочно-непрерывной функцией.
plot2d3 : аналогична команде **plot2d**, но кривая изображается в виде столбчатой диаграммы, у которой прорисованы только вертикальные линии.
plot2d4 : аналогична команде **plot2d**, но кривая изображается стрелочками.
Посмотрите демонстрационный пакет **plot2d2()**, **plot2d3()**, **plot2d4()**. Подробно они будут рассмотрены в дальнейшем.

Синтаксис функции **plot2d**
plot2d([x],y)
plot2d([x],y,[opt_args])
plot2d([logflag],x,y,[style,strf,leg,rect,nax])
Параметры
x,**y** : две матрицы или два вектора

Если **y** является вектором, то **x** должен быть вектором той же размерности. Если значение **x** не задано, оно полагается равным вектору [1:]. Это означает, что если `y=[2.2,4.4,9.3]`, то команда `plot2d(y)` будет считать, что `x=[1,2,3]`.
Замечание: Применяя такие конструкции, очень легко получить в дальнейшем ошибку.

Если **y** является матрицей, то **x** может быть:
1) вектором размера, равного размерности строки матрицы **y** Каждый столбец матрицы **y** будет изображаться относительно вектора **x**.
2) матрица того же размера, что и **y**. Каждый столбец **y** изображается относительно соответствующего столбца **x**.
3) если **x** не задан, он полагается равным вектору [1:[row dimension of y]].

[opt_args] : указывает последовательность соответствующих значений ключей
key1=value1,
key2=value2,... где **key1**, **key2**,... , которые могут принимать одно из следующих значений:

style : устанавливает стиль для каждой кривой. Принимаемые значения смотри ниже.

leg : устанавливает заголовок (подпись) кривой. Если этот ключ задан, а strf не задан, тогда "x"-символ для strf полагается равным 1. Смотри значения ниже.

rect : устанавливает границы графика. Если этот ключ задан, а ни frameflag, ни strf не заданы, то "y"-символ для strf полагается равным 7. Смотри значения ниже.

nax : устанавливает определение сетки (grids). Если этот ключ задан, и ни **axesflag**, ни **strf** не заданы, то "z"-символ для **strf** полагается равным 1. Смотри значения ниже.

logflag : устанавливает тип шкалы по осям (линейная или логарифмическая). Смотри значения ниже.

frameflag : устанавливает, как вычисляется рамка (frame) графика. Принимает целые значения от 0 до 8. Соответствует у character для strf. Смотри значения ниже.

axesflags : устанавливает, какого сорта оси будут нарисованы вокруг графика. Принимает целые значения от 0 до 5. Соответствует z character для strf. Смотри значения ниже.

Перечисленные выше ключи могут принимать следующие значения:
style :действительный (вещественный) вектор-строка размера nc. Стиль для i-той кривой определяется style(i). По умолчанию стиль равен 1:nc (1 для первой кривой, 2 - для второй и т. д.)
- если значение **style(i)** отрицательно или равно 0, то кривая изображается отмеченной id abs(style(i));
Используйте **xset()** для установки mark id и **xget("mark")** для получения его текущего значения.

- если значение **style(i)** строго положительно, то используется сплошная линия цвета id style(i) или пунктирная линия для id style(i). Для просмотра значений colors ids используйте команду **xset()**.

- если только изображается одна кривая, стиль может быть вектор-строкой размера 2 [sty,pos], где sty используется для определения style, а pos и является целым от 0 до 6 и определяет положение для написания заголовка.(caption). Это полезно, если пользователь хочет изобразить несколько кривых на график, вызывая функцию plot2d несколько раз и при этом хочет дать заглавие для каждой кривой отдельно.

strf :строка длиной 3 "xyz".

x : контролирует изображение заголовка
x=0 : нет заголовок.
x=1 : заголовок изображается. Задается аргументом leg.

y : контролирует вычисление текущей области координат для минимально требуемых размерах. Они могут быть больше минимально требуемых. Смотри значения y из таблицы в **help plot2d**.

z : контролирует вывод информации на обрамлении окна (frame) вокруг графика. Если необходимы оси, число промежуточных делений на осях (tics) может быть определены с помощью необязательного аргумента nax.
z=0 : вокруг графика ничего не изображается.
z=1 : нарисованы оси. Ось "y" слева.
z=2 : график окружен с 4-х сторон осями координат с делениями.
z=3 : нарисованы оси. Ось "y" справа.
z=4 : оси в центре окна.
z=5 : оси проходят через точку (0,0). Если эта точка не попала в изображаемую область, то и оси не изображаются.

leg : строковая переменная. Используется, если первый знак x аргумента **strf** равен 2. Переменная leg имеет вид "leg1@leg2@....", где leg1, leg2, и т.д. являются соответственно названиями первой кривой, второй кривой и т. д. По умолчанию переменная равна "".

rect : Этот аргумент используют, если первый знак y-аргумента **strf** равен 1, 3 или 5. Это вектор-строка размером 4 и определяет размер окна (frame): **rect=[xmin,ymin,xmax,ymax]**.

nax : Этот аргумент используют, если первый знак y-аргумента **strf** равен 1. Это вектор-

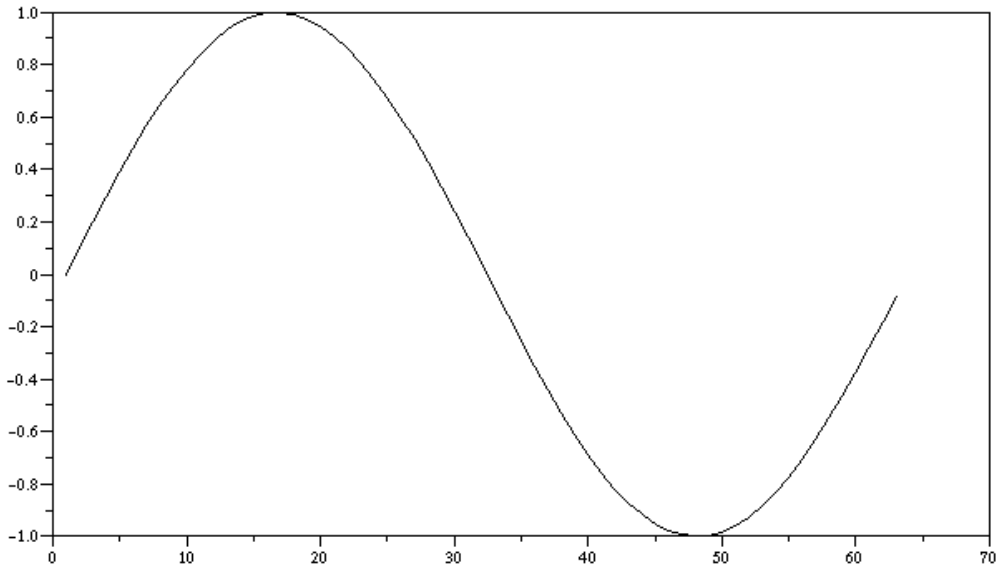
строка с 4 полями [nx,Nx,ny,Ny], где nx (ny) - число подуровней деления (не снабженных изображением цифр) оси x(y) и Nx(Ny) - число делений, подписанных цифрами на оси x(y).

logflag : строка формирующаяся с помощью characters h (для горизонтальной оси) и v (для вертикальной оси). Каждая из них может принимать значение "n" или "l". "l" означает логарифмический масштаб, а "n" - обычный. "ll" означает, что обе оси логарифмические. По умолчанию значение "nn".

Примеры.

```
x=[0:0.1:2*%pi]';
plot2d(sin(x))
```

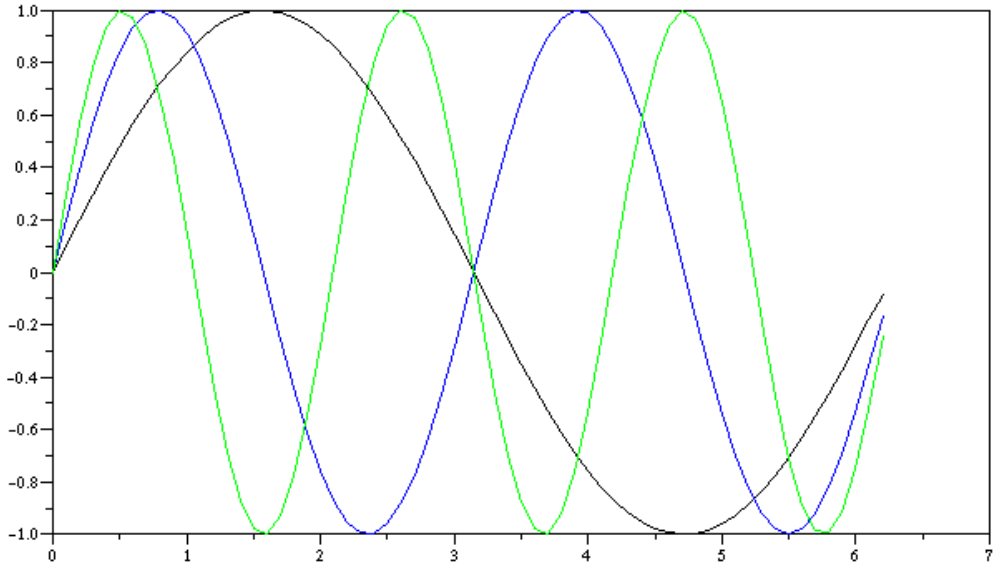
Результат:



Замечание: Вы обратили внимание, что у построенного графика координаты по оси абсцисс оказались совсем не такие, как Вы, может быть, ожидали. Это следствие пропущенного первого аргумента в функции **plot2d**.

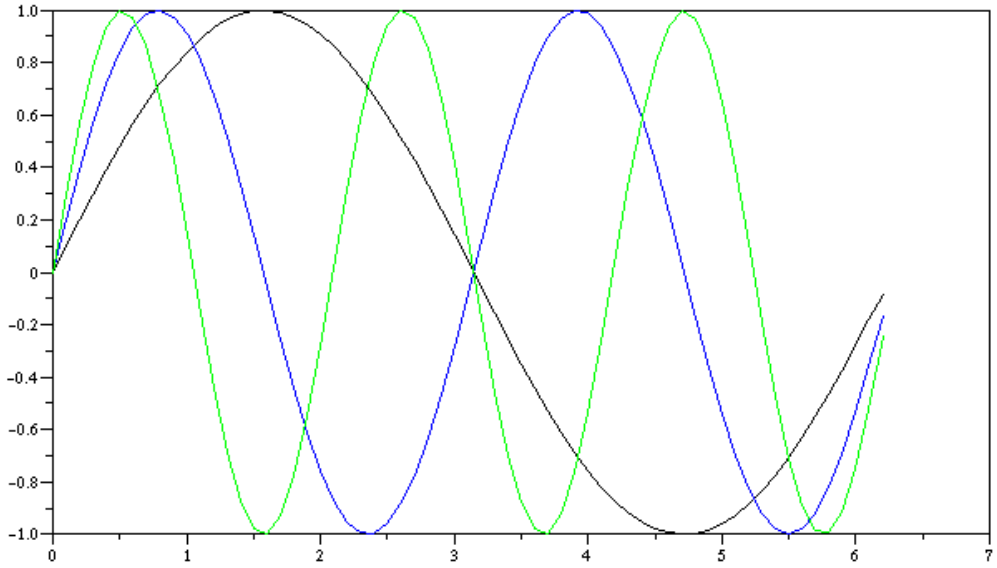
```
xbasc()
// Рекомендуемый способ
plot2d(x,sin(x))
```

Результат:



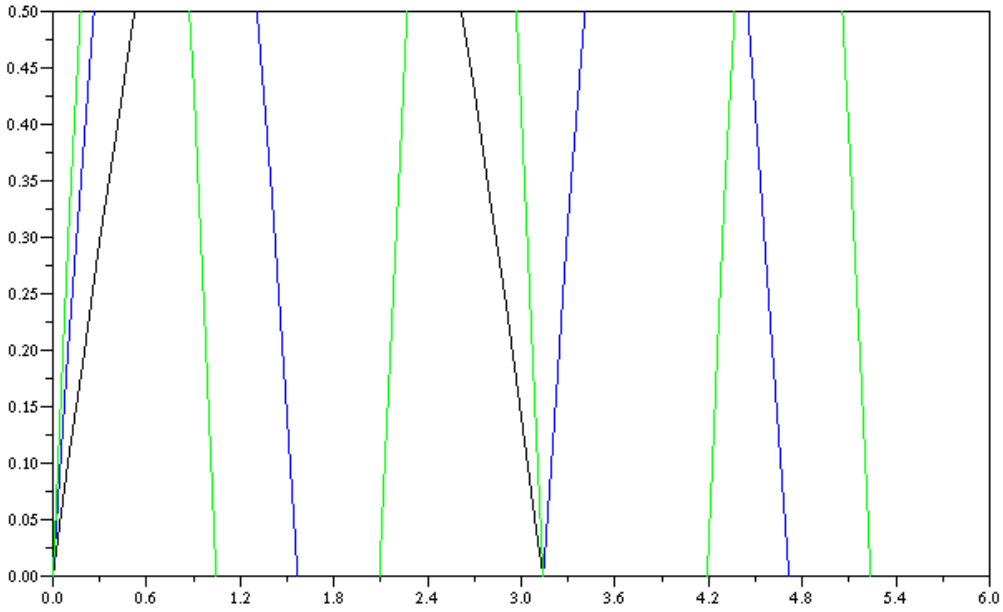

```
xbasc()
// несколько графиков в одних осях
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
```

Результат:




```
xbasc()
// несколько графиков, ограниченных фреймом окна
// старый синтаксис
plot2d(x,[sin(x) sin(2*x) sin(3*x)],1:3,"011"," ",[0,0,6,0.5])
```

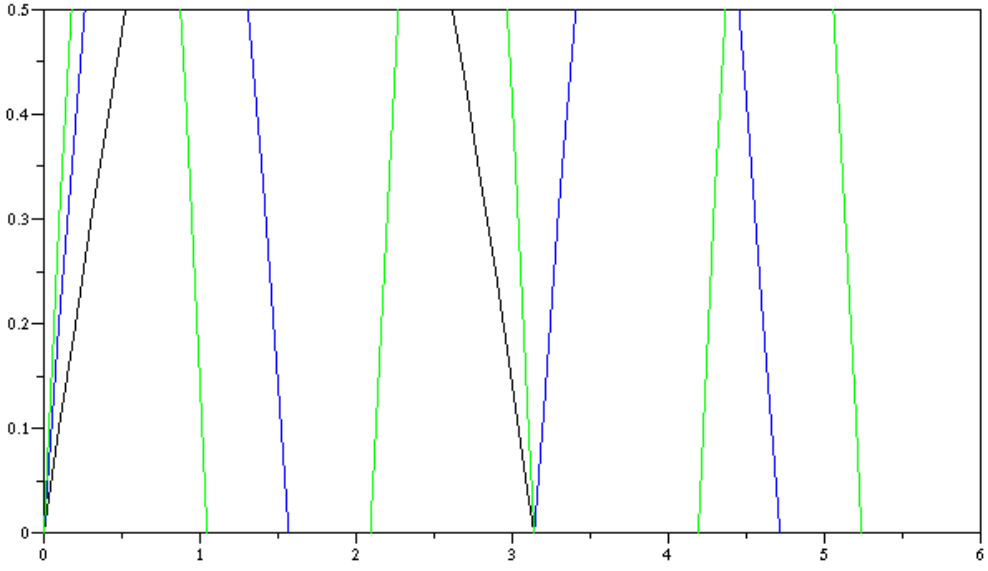
Результат:



Замечание: Это перенос из clipboard. Он шире, чем то, что я вижу в окне Scilab: там все ограничено рамкой осей с небольшими полями. В файле GIF все соответствует точно тому, что я вижу.

```
xbasc()
// несколько графиков, ограниченных фреймом окна
// новый синтаксис
plot2d(x,[sin(x) sin(2*x) sin(3*x)],rect=[0,0,6,0.5])
```

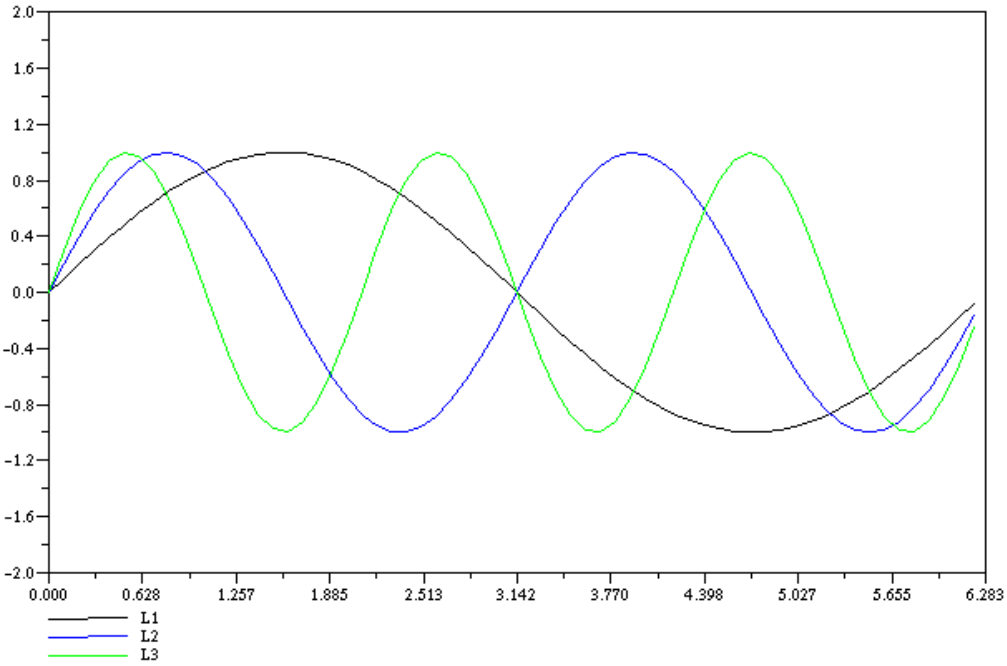
Результат:



Замечание: Этот рисунок отличается от предыдущего только маркировкой осей x и y.

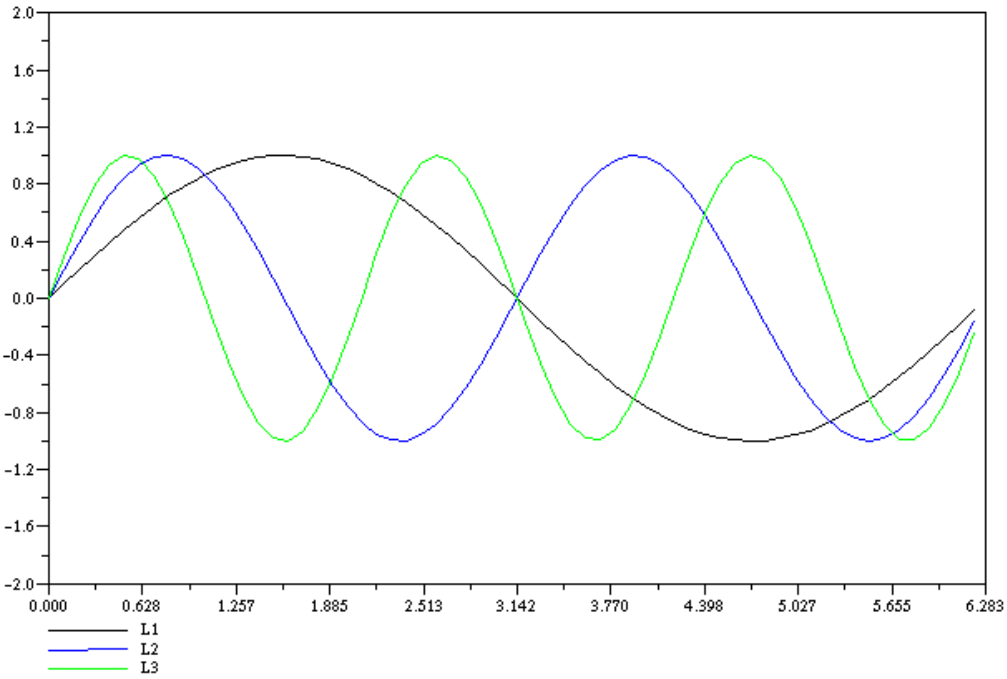
```
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],[1,2,3],"111","L1@L2@L3",[0,-
2,2*%pi,2],[2,10,2,10])
// несколько графиков с легендой для каждого графика и заданными
промежуточными делениями на осях ( "tics")
```

```
// старый синтаксис
```
Результат:




```
xbasc()
plot2d(x,[sin(x) sin(2*x)
sin(3*x)],[1,2,3],leg="L1@L2@L3",nax=[2,10,2,10],rect=[0,-2,2*%pi,2])
// несколько графиков с легендой для каждого графика и заданными
промежуточными делениями на осях ( "tics")
// новый синтаксис
```

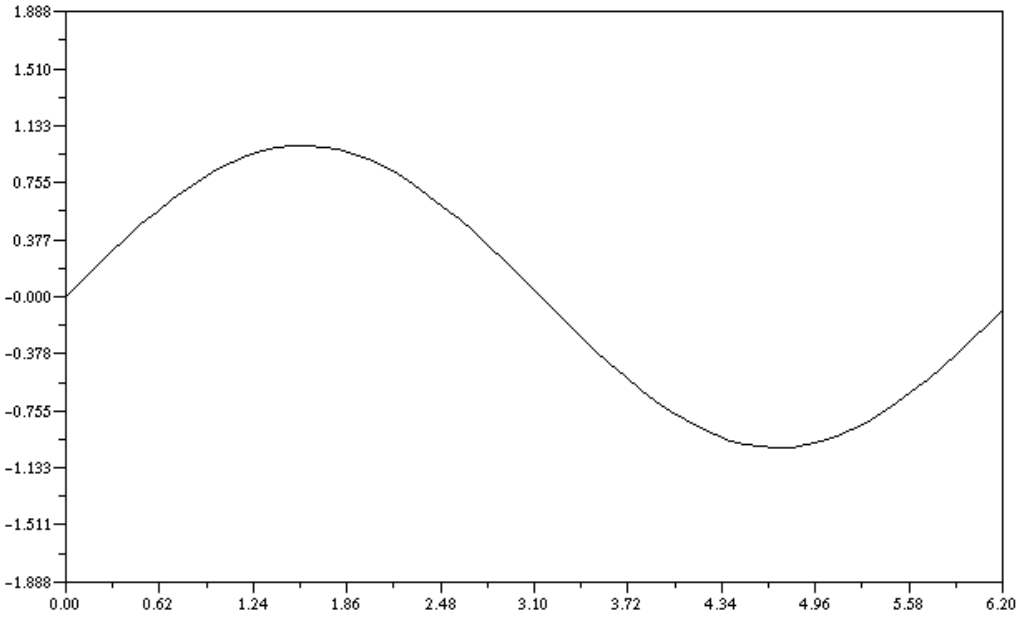
Результат:




```
// масштабирование
xbasc()
```

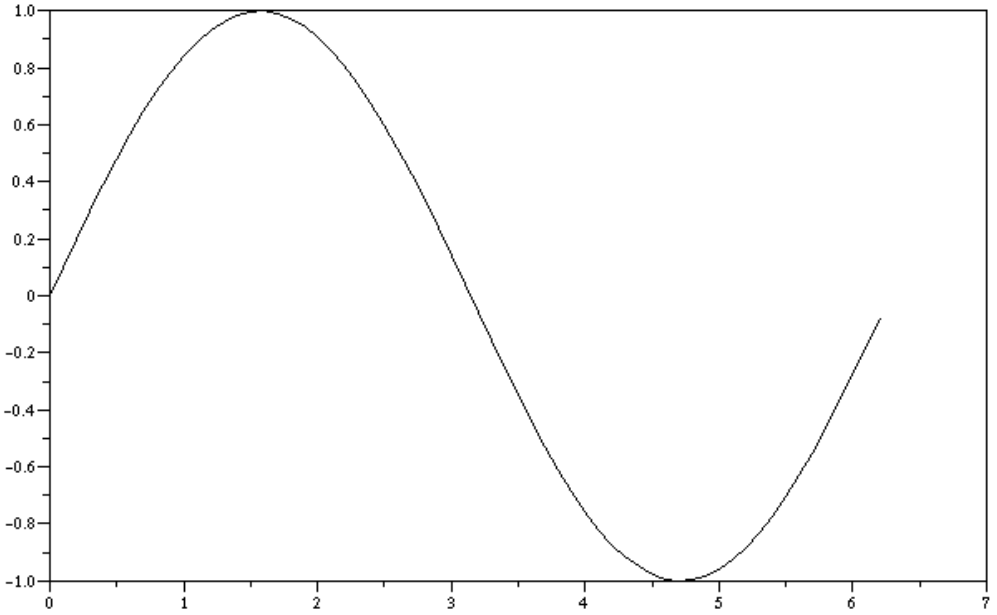
```
plot2d(x,sin(x),1,"041")
```

Результат:



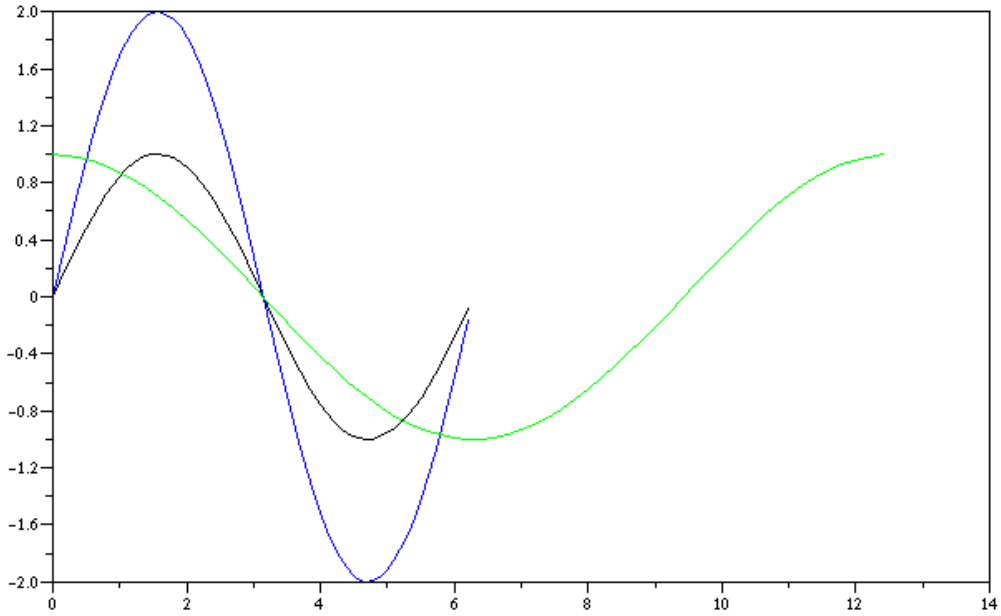

```
xbasc()
plot2d(x,sin(x),1,"061")
// масштабирование
```

Результат:



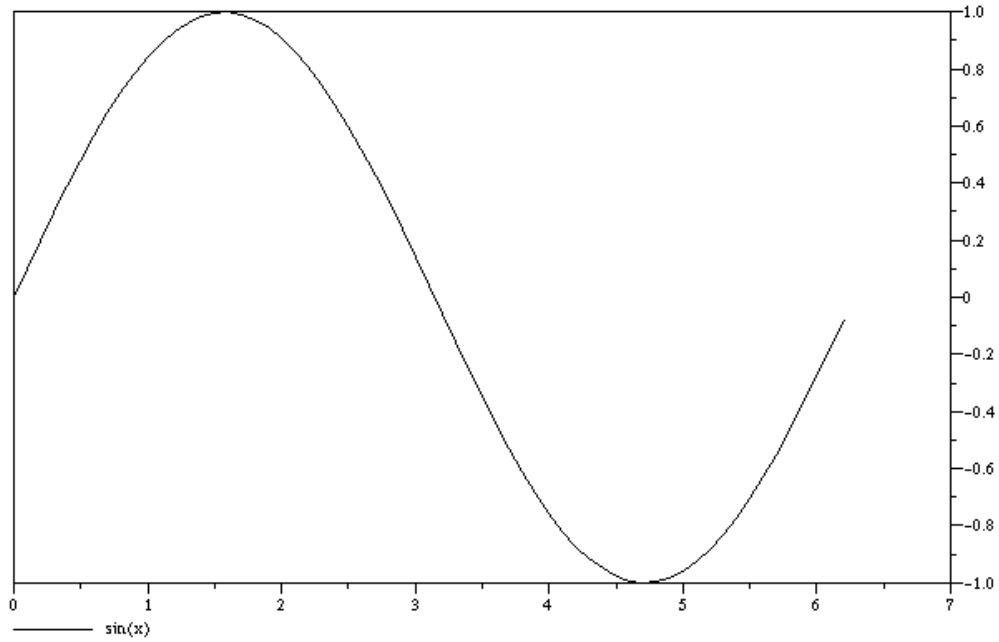

```
// автомасштабирование с предыдущими графиками
xbasc()
plot2d(x,sin(x),1)
plot2d(x,2*sin(x),2)
plot2d(2*x,cos(x),3)
```

Результат:



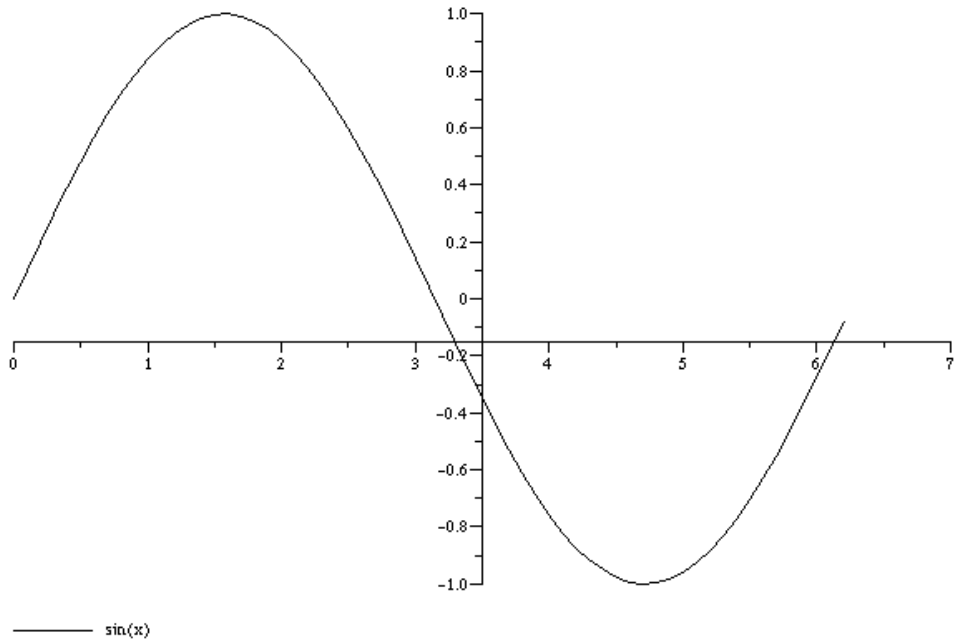

```
//оси справа
xbasc()
plot2d(x,sin(x),1,"183","sin(x)")
```

Результат:



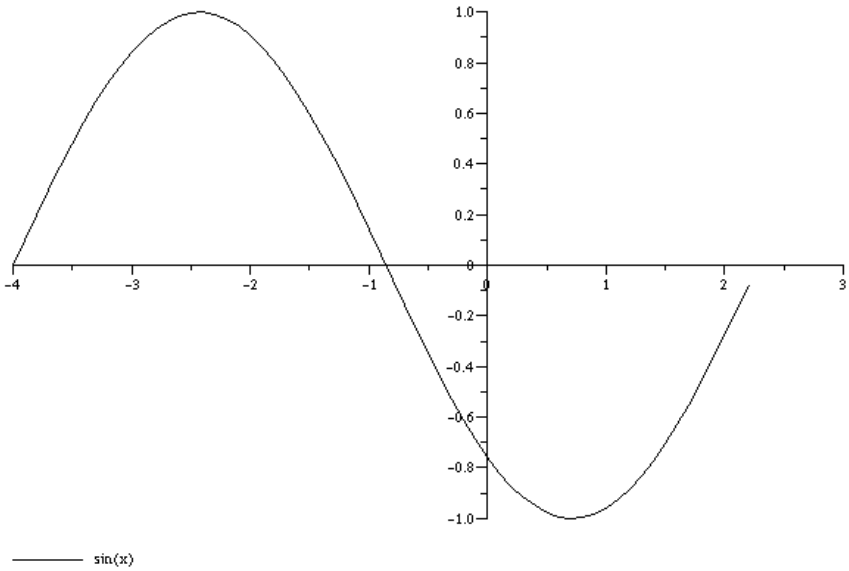

```
// оси через центр окна
xbasc()
plot2d(x,sin(x),1,"184","sin(x)")
```

Результат:




```
// оси координат пересекаются в точке (0,0)
xbasc()
plot2d(x-4,sin(x),1,"185","sin(x)")
```

Результат:



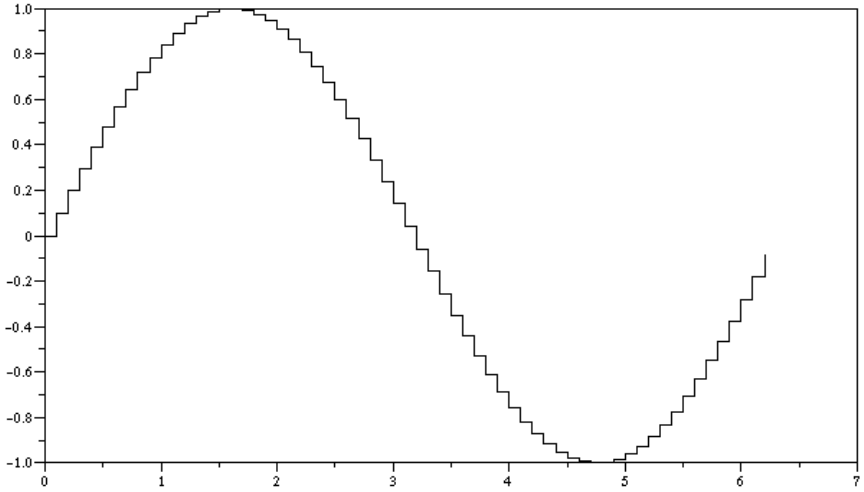
Какие еще есть варианты для изображения 2D-графика?

1)Команда **plot2d2** - 2D график в виде ступенчатой ломаной линии.
Полный синтаксис смотри с помощью **help plot2d2**.

Пример.

```
x=[0:0.1:2*%pi]';
xbasc()
plot2d2(x,sin(x))
```

Результат:



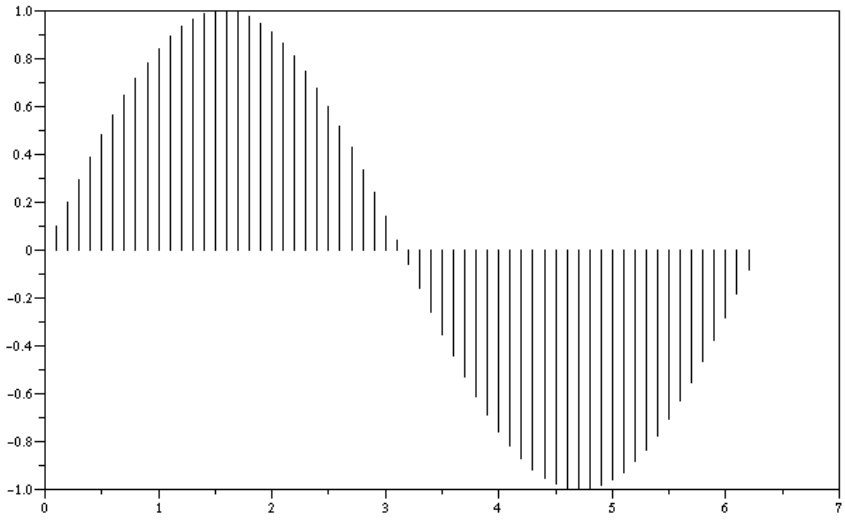
Команда **plot2d2()** даст подробный демонстрационный пример.

2)Команда **plot2d3** - аналогична команде **plot2d**, но кривая изображается в виде столбчатой диаграммы, у которой прорисованы только вертикальные линии.

Пример.

```
x=[0:0.1:2*%pi]';
xbasc();
plot2d3(x,sin(x))
```

Результат:



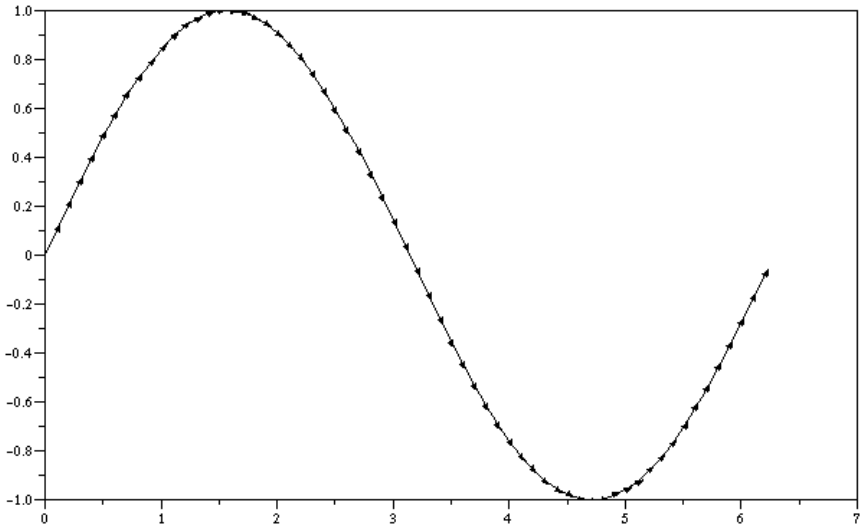
Команда **plot2d3()** даст более подробный демонстрационный пример.

3)Команда **plot2d4** аналогична команде **plot2d**, но кривая изображается стрелочками от

одной точки графика к другой.
Пример.

```
x=[0:0.1:2*%pi]';
xbasc();
plot2d4(x,sin(x))
```

Результат:

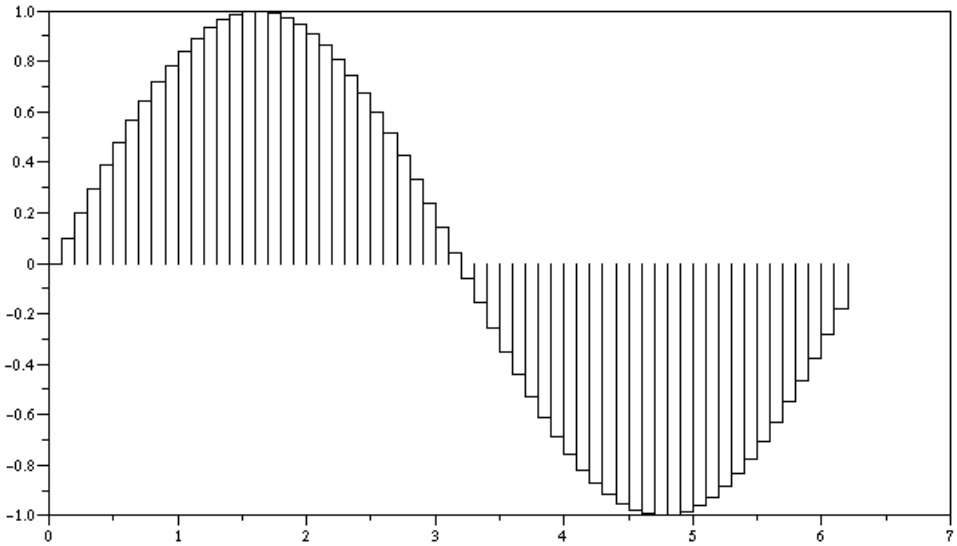


Команда **plot2d4()** даст подробный демонстрационный пример.

Классическую столбчатую гистограмму можно получить последовательно применив команды **plot2d2** и **plot2d3**.
Пример.

```
x=[0:0.1:2*%pi]';
plot2d2(x,sin(x));
plot2d3(x,sin(x))
```

Результат:



4) Команды для получения гистограмм: **histplot** для двумерной и **hist3d** для трехмерной гистограмм. Изучите их сами.

Дополнительные возможности в графике (заголовки, надписи и др.)

Как изобразить вспомогательную сетку на графике?

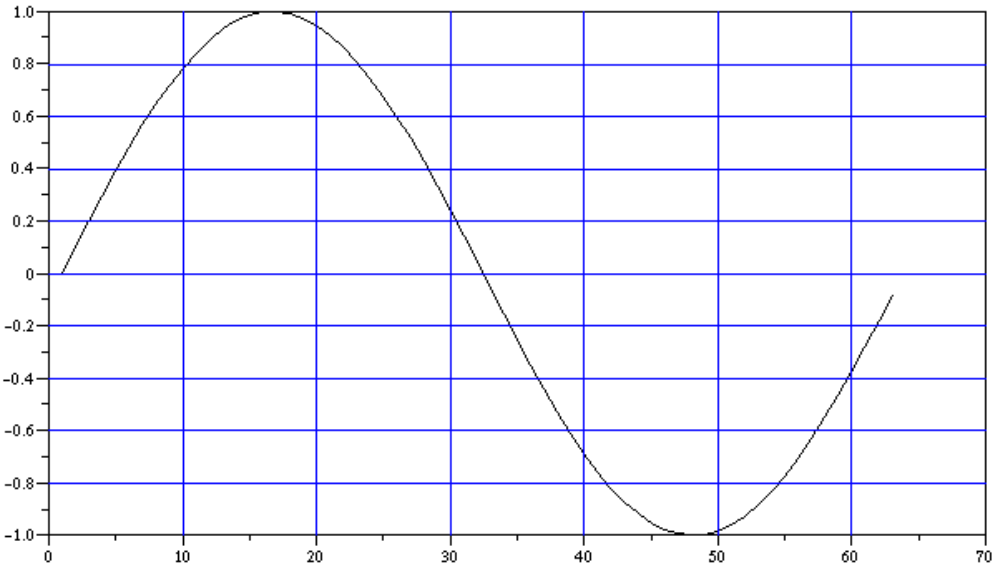
С помощью команды **xgrid**.

Синтаксис
xgrid([style])
Параметры
style : целое число. Указывает на стиль изображения вспомогательной сетки. Это - цвет или тип черной штрихованной линии, предварительно задаваемый с помощью команды **xset()** для текущего графического окна.

Пример.

```
x=[0:0.1:2*%pi]';
plot2d(sin(x));
xgrid(2)
```

Результат:



Как сделать надпись в графическом окне?

Способ 1.
С помощью команды **titlepage**. Изображает заголовок максимально возможного размера шрифта в центре графического окна. Возможности изменения координат надписи отсутствуют.
Синтаксис
titlepage(str)
Параметры
str : матрица из строковых переменных
При создании нового окна будет создана просто рамка с заданной надписью в середине. Если окно с каким-то изображением уже создано, надпись будет добавлена в центр окна поверх изображения.

Пример.
```
titlepage("Привет!")
```

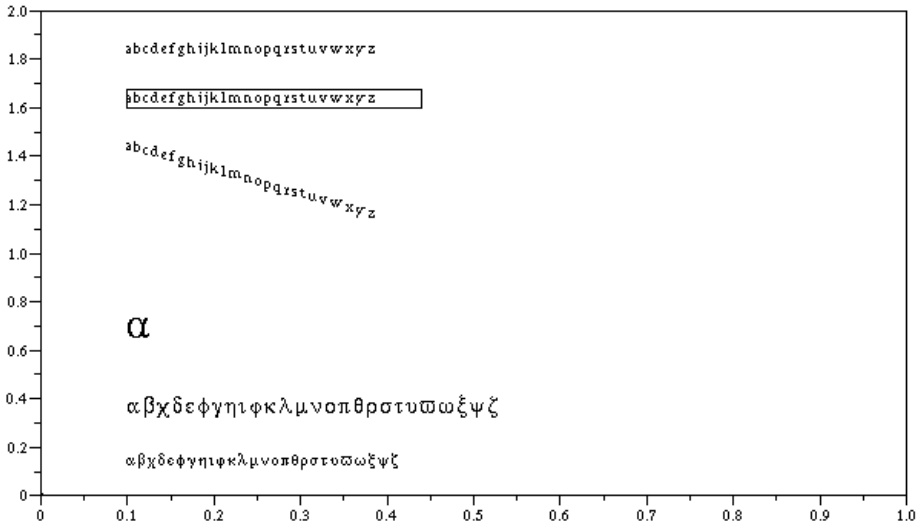
Замечание: Надпись на русском языке в главном (не графическом) возможна не для всех шрифтов. Для русского языка выберите с помощью строкового меню главного окна **Edit - Choose Font** подходящий шрифт, например, "Courier". В этом случае в главном окне Вы сможете пользоваться русским языком. Шрифт в графическом окне при этом не изменяется.

Способ 2.
С помощью команды **xstring**. Эта команда позволяет более гибко менять параметры надписи.
Синтаксис
xstring(x,y,str,[angle,box])
Параметры
x,**y** :действительные числа, определяющие положение левого нижнего угла надписи. Учтите, что это единицы в масштабе заданного Вами окна. Например, этот масштаб определит выполненная предварительно функция plot2d([0;1],[0;2],0)
str : строковая матрица, определяющая содержание надписи.
angle : действительное число, определяющее в градусах угол поворота надписи по часовой стрелке, по умолчанию равно 0.
box : целое число, по умолчанию =0. Если равно **box**=0, то рамки вокруг надписи нет, если =1, то рамка есть.

Примечание: При повороте надписи рамка почему-то не желает вслед за ней поворачиваться.

Пример 1.
```
alphabet=["a" "b" "c" "d" "e" "f" "g" ..
"h" "i" "j" "k" "l" "m" "n" ..
"o" "p" "q" "r" "s" "t" "u" ..
"v" "w" "x" "y" "z"];
xbasc()
plot2d([0;1],[0;2],0) // Это важно для определения масштаба графического окна
xstring(0.1,1.8,alphabet) // алфавит
xstring(0.1,1.6,alphabet,0,1) // алфавит в рамке
xstring(0.1,1.4,alphabet,20) // поворот на угол
xset("font",1,1) // использование символьных шрифтов
xstring(0.1,0.1,alphabet)
xset("font",1,3) // изменение размера шрифта
xstring(0.1,0.3,alphabet)
```

```
xset("font",1,24);
xstring(0.1,0.6,"a") // большая буква "альфа"
```
Замечание: Для возврата к начальным установочным значениям следует выполнить xset("default").
Результат:



Пример 2.
Хотим изобразить матрицу размером 3 на 3.
```
q=["11" "22" "33";"21" "22" "23";"31" "32" "33"];
xstring(0.7,0.3,q)
```
Пример 3.
Для изображения нескольких строк отдельным блоком:
```
z=["It is first line";"I is second line"];
xstring(0.2,0.6,z)
```

Способ 3.
С помощью команды **xstringb**. Эта команда изображает текст в заданной рамке (box). Смотри подробно **help xstringb**.

Способ 4.
Если вы хотите изобразить числа, то вместо **xstring** лучше использовать **xnumb**. Смотри подробно **help xnumb**.

Как вычислить размеры рамки (box), окружающей заданную строку?

С помощью команды **xstringl**. Эта команда иногда полезна для макетирования текста в презентациях.

Синтаксис
rect=xstringl(x,y,str)
Параметры
rect : вектор из 4-х действительных чисел, определяющих рамку вокруг строки.
x, **y** : действительные числа, координаты левого нижнего угла строки.
str : матрица строк.
Команда **xstringl** возвращает в вектор **rect=[x,y,w,h]** (нижняя левая точка, ширина, высота) размер прямоугольника в масштабе графическом окне.

Пример.

```
plot2d([0;1],[0;1],0) //для определения масштаба графического окна
str=["Scilab" "is";"not" "elisaB"];
r=xstringl(0.5,0.5,str)
xrects([r(1) r(2)+r(4) r(3) r(4)]')
xstring(r(1),r(2),str)
```

Важное замечание: Проделайте тот же пример еще раз, заменив первую строку на `plot2d([0;5],[0;5],0)`, и Вы увидите, что полученное значение `r` изменится.

Как получить фрейм с масштабом и сеткой?

С помощью команды **plotframe**.
С помощью этой операции мы контролируем положение делений на осях, выбираем размер прямоугольной области для изображений и добавляем координатную сетку (grid). Выполненная один раз, эта команда может быть использована и для изображения последующих графиков. Используется совместно с командами **plot2d**, **plot2d1** и т.д. Должна быть использована до применения **plot2d**, которая, в свою очередь, должна быть вызвана с совмещением моды "000". (например, `plot2d(x,y,2,"000")`).

Синтаксис
plotframe(rect,tics,[arg_opt1,arg_opt2,arg_opt3])
Параметры
rect : вектор=[xmin,ymin,xmax,ymax].
tics : вектор=[nx,mx,ny,my], где mx и nx (соответственно my и ny) являются числами интервалов и подинтервалов на x-оси (соответственно y-оси).

arg_opt1, arg_opt2, arg_opt3 : необязательные аргументы следующего содержания:
1) **flags** : вектор **[wantgrids,findbounds]**, где **wantgrids** и **findbounds** - булевские переменные (%t или %f). Переменная **wantgrids**, указывает на наличие или отсутствие координатной сетки.
Если **findbounds** =%t, то правая граница может быть слегка уменьшена для более простой градации. В этом случае tics(2) and tics(4) игнорируются.

2) **captions** : вектор из трех строк=**[title,x-leg,y-leg]**, соответствующих названию графика и заглавиям x- и y-осей

3) **subwin** : вектор размером =4, определяющий положение и размер подокна. Для этого служат параметры **subwin=[x,y,w,h]** (координаты верхней левой точки окна, ширина, высота). Значения размеров подокон определяются с использованием пропорций ширины или высоты текущего графического окна, т.е. все эти параметры (x,y,w,h) лежат в интервале [0,1].

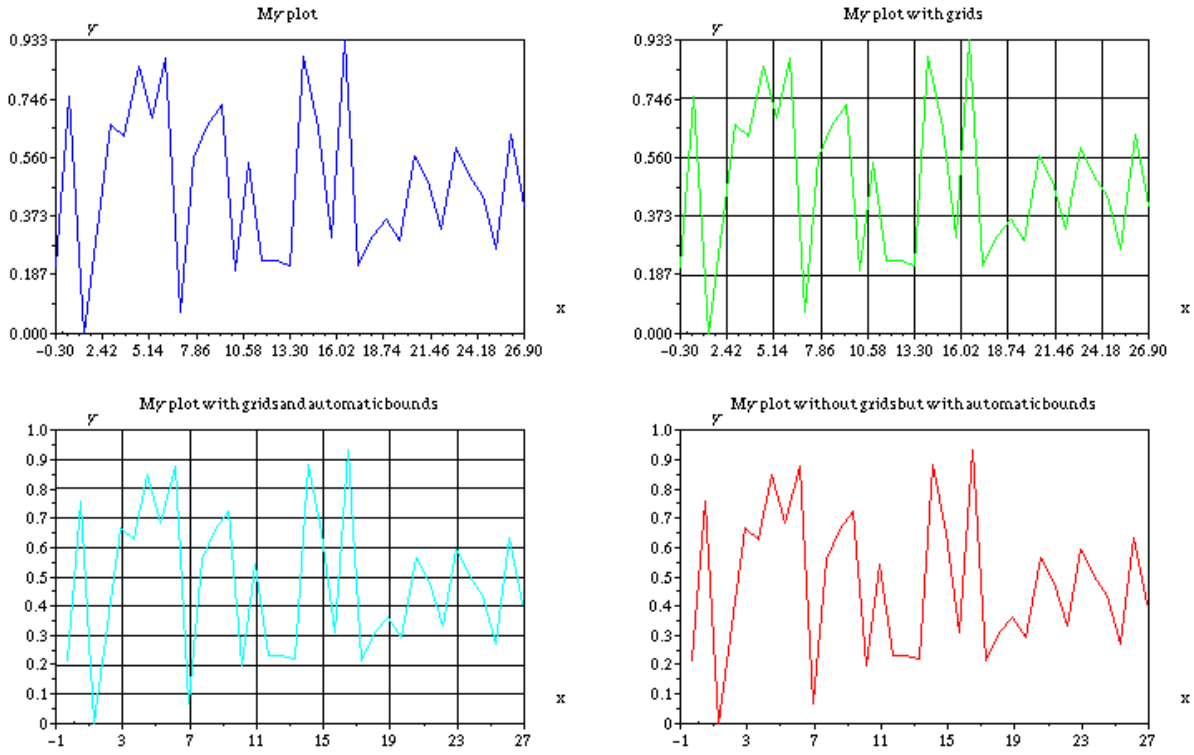
Если subwin=[0,0,1,1], то изображение графика займет целиком все графическое окно.
Если subwin=subwin=[0,0,1,0.5], то график будет находиться в верхней половине графического окна.

Пример.
Построим рисунок, состоящий из четырех графиков.

```
x=[-0.3:0.8:27.3]';
```

```
y=rand(x);
rect=[min(x),min(y),max(x),max(y)];
tics=[4,10,2,5]; //5-число интервалов по оси y, 2-число подинтервалов по оси
y
plotframe(rect,tics,[%f,%f],["My plot","x","y"],[0,0,0.5,0.5])
plot2d(x,y,2,"000")

plotframe(rect,tics,[%t,%f],["My plot with grids","x","y"],[0.5,0,0.5,0.5])
plot2d(x,y,3,"000")

plotframe(rect,tics,[%t,%t],..
["My plot with grids and automatic bounds","x","y"],[0,0.5,0.5,0.5])
plot2d(x,y,4,"000")

plotframe(rect,tics,[%f,%t],..
["My plot without grids but with automatic bounds","x","y"],..
[0.5,0.5,0.5,0.5])
plot2d(x,y,5,"000")

xset("default") // для возврата к установкам по умолчанию
```

Результат:



Какие оси будут считаться "красивыми" для Scilab и как их сформировать?

В Scilab положение осей считается (это постулат!!!) **"красивым"**, если они отвечают следующим условиям: ищется такой минимальный интервал **[xi,xa]** и такое число промежуточных делений на осях **np**, что:
1) **xi <= xmi <= xma <= xa**
2) **xa - xi / np = k(10^n),k в [1 3 5] для целого n**
3) n1 < np < n2

Параметры калибровки "красивого вида" вычисляются с помощью команды **graduate**. В

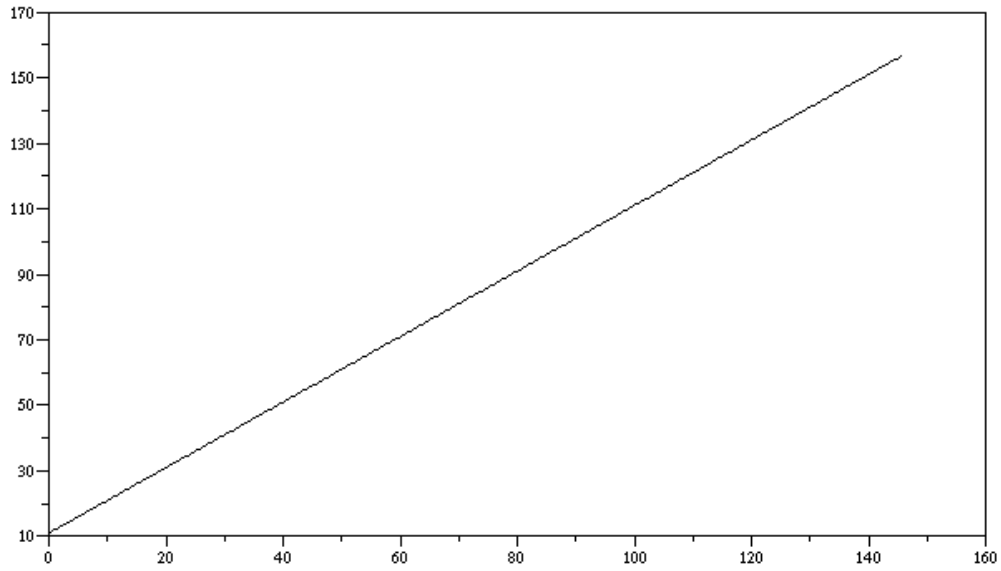
дальнейшем вычисленные параметры будут использованы в командах семейства **plot2d**.

Синтаксис
[xi,xa,np]=graduate(xmi, xma,n1,n2)
[xi,xa,np]=graduate(xmi, xma)
Параметры
xmi, **xma** : действительные скаляры
n1, **n2** : целые числа со значениями по умолчанию n1=3 и n2=10
xi, **xa** : действительные скаляры
np : целое число

Пример.

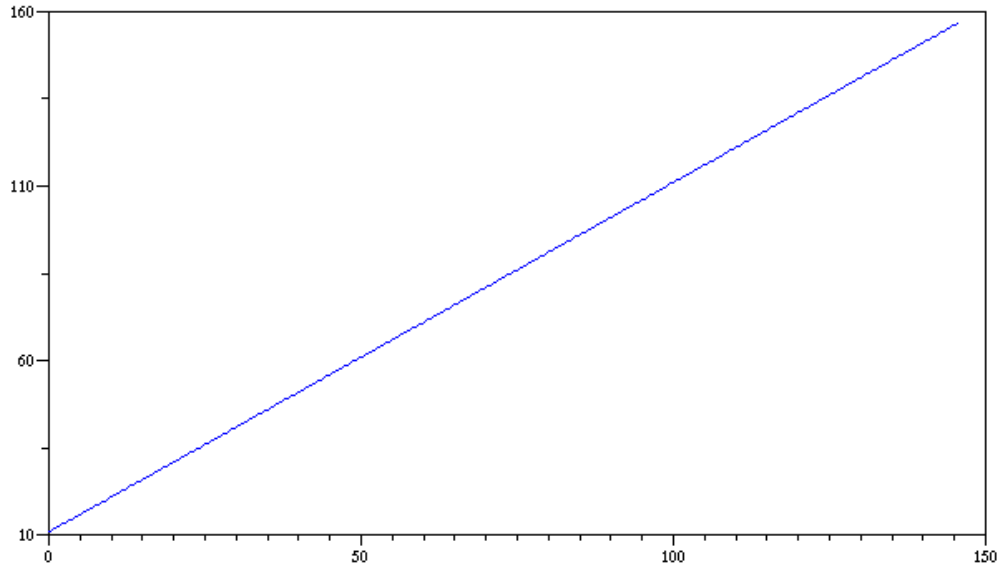
```
x=(0:0.33:145.78);
y=(11:0.33:156.78);
plot2d(x,y);
```

Результат:




```
xbasc();
[xmn,xmx,npx]=graduate(mini(x),maxi(x));
[ymn,ymx,npy]=graduate(mini(y),maxi(y));
rect=[xmn,ymn,xmx,ymx];
plot2d(x,y,2,'011',' ',rect,[10,npx,2,npy])
```

Результат:



Специализированные 2D графики

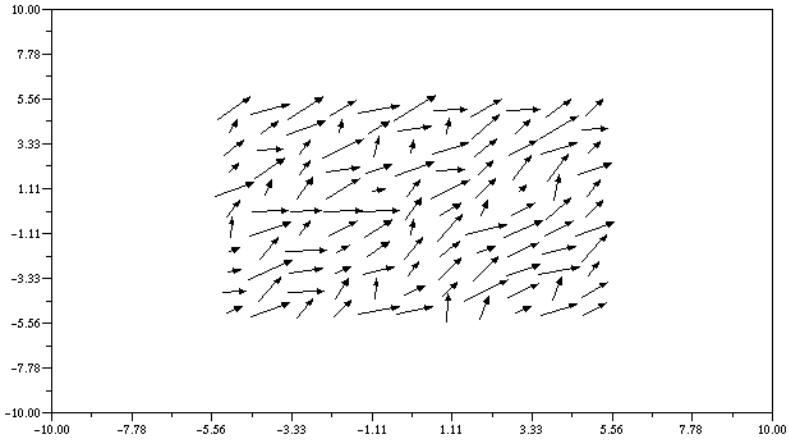
Как сделать изображение в виде векторных полей в двухмерном пространстве R^2?

Способ 1.
С помощью команды **champ**.
При использовании команды **champ** стрелки, изображающие векторные поля, будут черные, а их длина будет зависеть от интенсивности поля. Для отображения интенсивности векторного поля с помощью цветных векторов используйте команду **champ1** (Способ 2).
Синтаксис
champ(x,y,fx,fy,[arfact,rect,strf])
champ(x,y,fx,fy,[opt_args])
Параметры
x, **y** : два вектора, определяющие сетку координат.
fx : матрица, описывающая x-компоненту каждого поля вектора.
fx(i,j) является x-компонентой в точке (x(i),y(j)).
fy : матрица, описывающая y-компоненту каждого поля вектора.
fy(i,j) является y-компонентой в точке (x(i),y(j)).
[opt_args] : изображает последовательность заявлений key1=value1, key2=value2,... где key1, key2,... могут быть одними из следующих: **arfact**, **rect**, **strf**.
Необязательные параметры
arfact : необязательный аргумент типа real, определяющий масштабный фактор для

изображения "головок" стрелок на графике. Значение по умолчанию =1.0.
rect : вектор rect=[xmin,ymin,xmax,ymax] определяет границы используемого графического окна(frame).
strf : строка длины 3 "xyz" , означающая то же, что параметр **strf** в **plot2d**. Первый знак "x" не влияет на результат команды **champ**. (Например, "011" и "111" дадут одинаковый результат).
Пример.

```
champ(-5:5,-5:5,rand(11,11),rand(11,11),1,[-10,-10,10,10],"011")
```

Результат:

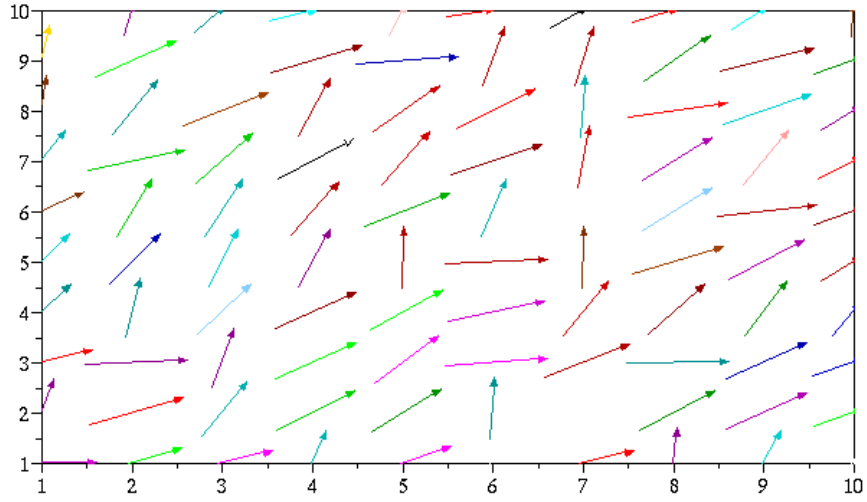


Способ 2.
С помощью **champ1**
Цвет стрелок зависит от интенсивности поля, а длина от нее не зависит. Смотри синтаксис в help.

Пример.

```
champ1(1:10,1:10,rand(10,10),rand(10,10),1.0)
```

Результат:



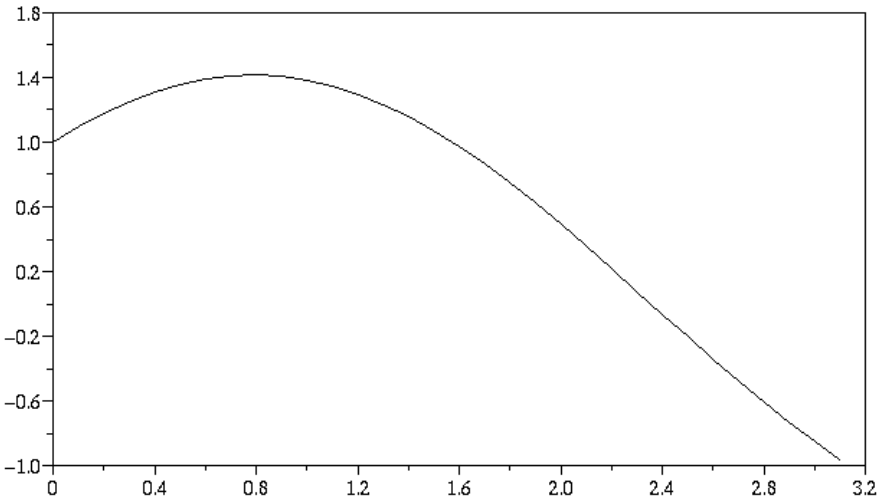
Способ 3.
С помощью **fchamp**, если векторные поля определены в виде внешней функции. Смотри демонстрационный пример **fchamp()**.

Как изобразить график кривой, описываемой внешней функцией?

С помощью команды **fplot2d**. Это аналог команды **plot2d**.
Пример.

```
deff('[y]=f(x)','y=sin(x)+cos(x)'); //Это определение функции y(x)
fplot2d(0:0.1:%pi,f);
```

Результат:



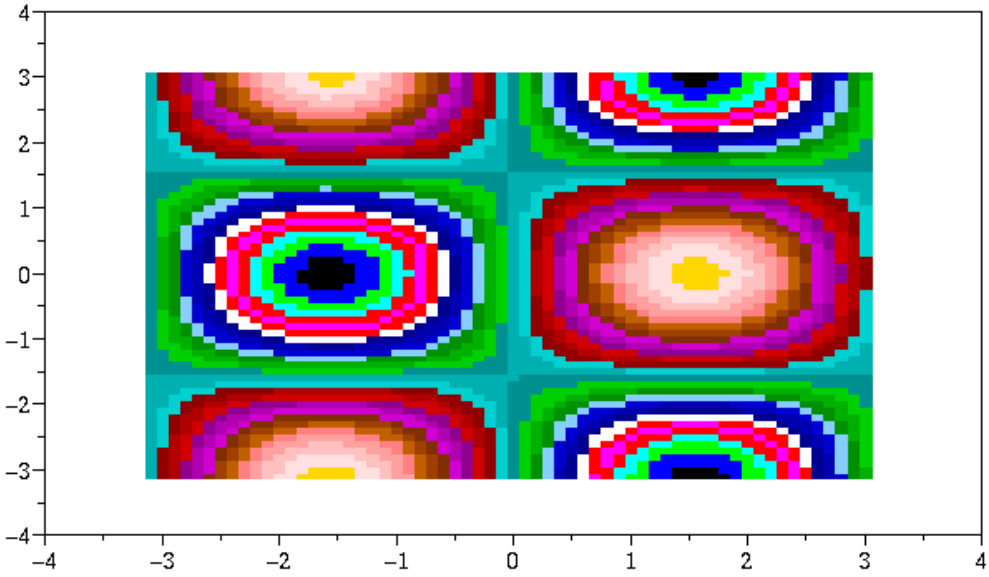
Как изобразить 3D график на плоскости XY, чтобы координата Z задавалась цветом (или градацией серого)?

С помощью команды **grayplot** (если Z задана в самой команде) или с помощью команды **fgrayplot**, если координата Z задана в виде внешней функции.
Для получения такого же, но более сглаженного по цвету (smooth) служат соответствующие команды **Sgrayplot** и **Sfgrayplot**.

Пример.

```
t=-%pi:0.1:%pi;
m=sin(t)'*cos(t);
grayplot(t,t,m);
```

Результат:



Как изобразить на 2D графике ошибки в виде вертикальных отрезков ("ошибки" измерений и вычислений)?

С помощью команды **errbar**.

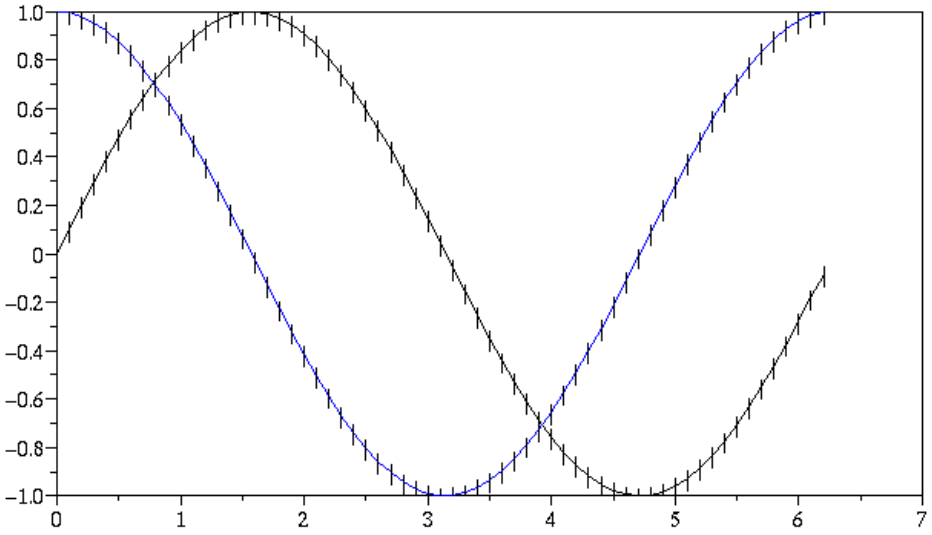
Синтаксис
errbar(x,y,em,ep)

Параметры
x, **y**, **em**, **ep** : четыре матрицы одно и того же размера.

Пример.

```
t=[0:0.1:2*%pi]';
y=[sin(t) cos(t)]; x=[t t];
plot2d(x,y)
errbar(x,y,0.05*ones(x),0.03*ones(x))
```

Результат:

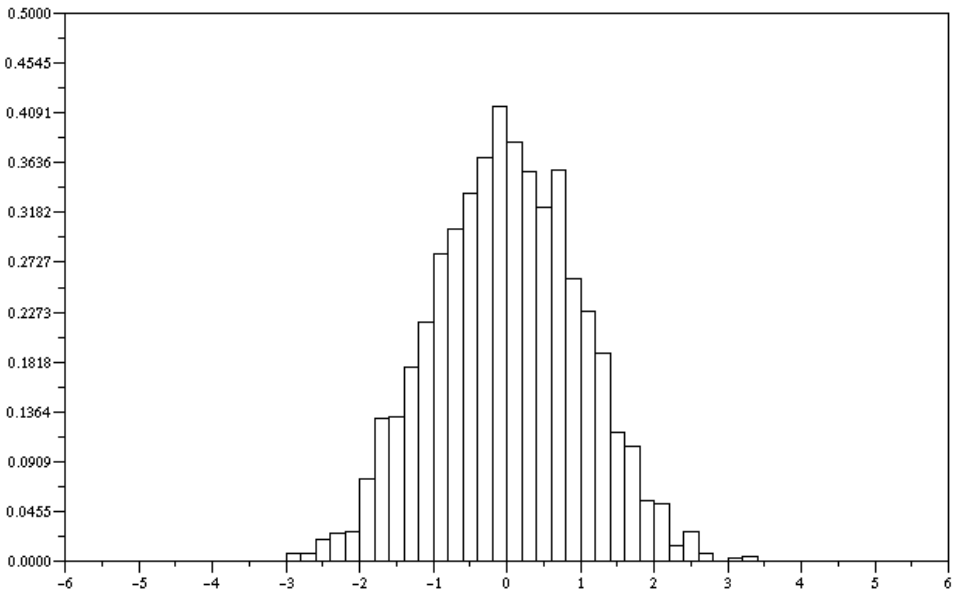


Как нарисовать гистограмму?

С помощью команд **histplot** и **hist3d**.
Пример.

```
histplot([-6:0.2:6],rand(1,2000,'n'),[1,-1],'011',' ',[-
6,0,6,0.5],[2,12,2,11]);
```

Результат:



Изображение некоторых геометрических фигур

Основные команды для изображения объектов из ломаных линий (многоугольников):

xsegs - изображение несвязанных сегментов
xrect, **xfrect**, **xrects** - изображение прямоугольника(ов)
xpoly, **xpolys**, **xfpoly**, **xfpolys** - изображение полилинии(й) или полигона(ов)
xarrows - изображение системы стрелок
xclea - уничтожение (стирание) прямоугольной области

Основные команды для изображения кривых:
xarc, **xfrac**, **xarcs**, **xfarcs** -изображение эллипса(ов) или их части(ей).

Как изображать несвязанные сегменты?

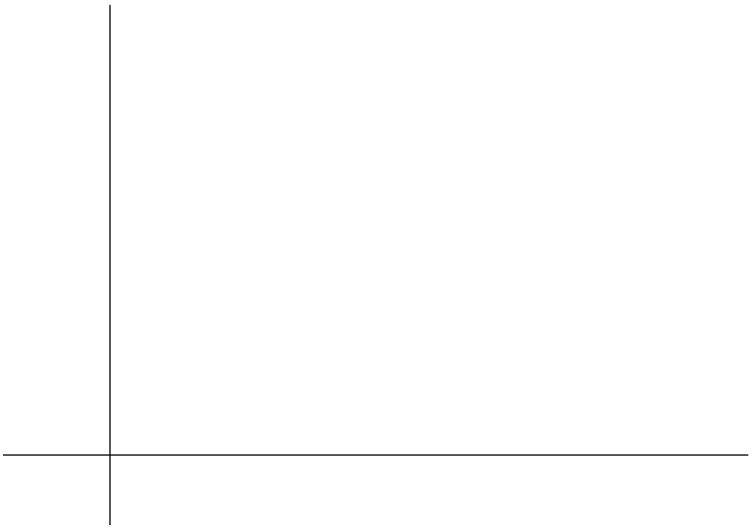
С помощью команды **xsegs**.
Синтаксис
xsegs(xv,yv,[style])
Параметры
xv, **yv** : матрицы одного и того же размера.
style : вектор или скаляр. Если **style** является положительным скаляром, используется пунктирный стиль (dash) для всех сегментов, если **style** - отрицательный скаляр, используется текущий пунктирный стиль. Если это вектор, тогда стиль (стили) определяются для каждого i-того сегмента отдельно.
Отрезки задаются с помощью параметров **xv** и **yv**. Координаты двух точек, определяющих сегмент, даются последовательностями из **xv** и **yv** и соответствием **(xv(i),yv(i))-->(xv(i+1),yv(i+1))**.
Например, для матрицы размером (2,n) сегменты могут быть определены как:
xv=
[xi_1 xi_2 ...;
xf_1 xf_2 ...]

yv=
[yi_1 yi_2 ...;
yf_1 yf_2 ...]

и сегменты (xi_k,yi_k)-->(xf_k,yf_k).
Подробно смотри в **help**. Этот способ, наверное, хорош для изображения параметрических фигур.

Пример 1.
```
xsegs([-1.0,0;1.0,0],[0,-1.0;0,1.0])
```
В результате будут нарисованы две пересекающиеся прямые.

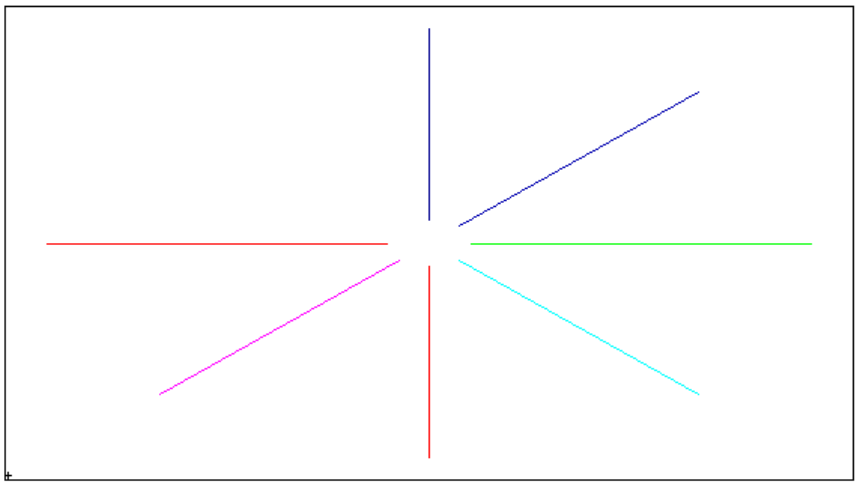
Результат:



Пример 2.

```
x=2*%pi*(0:9)/8;
x1=[sin(x);9*sin(x)];
y1=[cos(x);9*cos(x)];
plot2d([-10,10],[-10,10],[-1,-1],"022");
xsegs(x1,y1,1:10);
```

Результат:

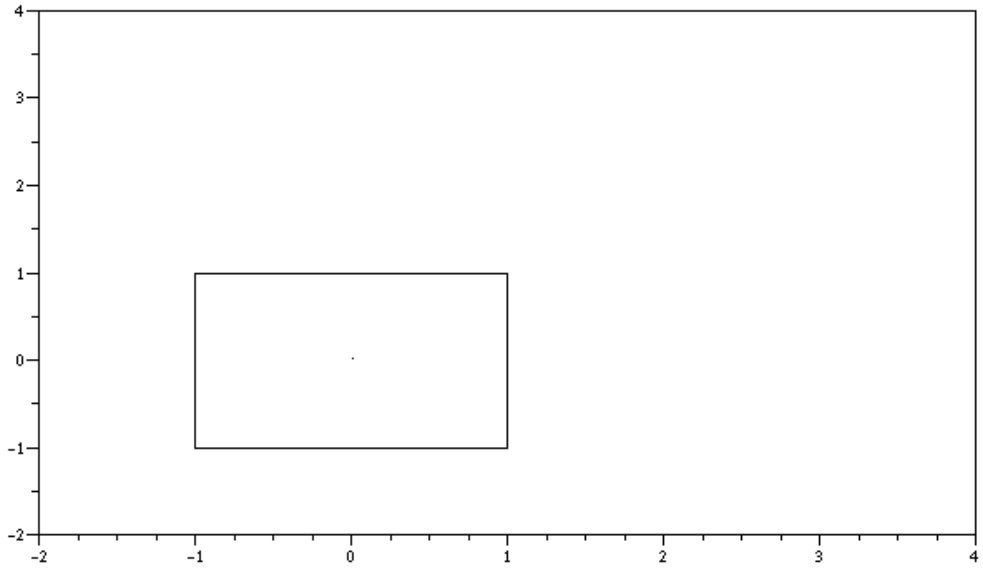


Замечание: Для определения текущего масштаба могут применяться кроме команды **plot2d** и другие команды (**plotframe** и т. д.). Если текущий масштаб не задан, то размеры окна непонятны, и результат непредсказуем. Если изображение окажется вне области экрана, то получим пустое графическое окно без каких-либо предупреждающих сообщений. То же относится и ко всем другим командам изображения геометрических объектов.

Как изобразить прямоугольник?

С помощью команды **xrect**.
Синтаксис
xrect(x,y,w,h)
xrect(rect) ,где **rect =[x,y,w,h]**
Параметры
x, **y**, **w**, **h** : четыре действительные числа, определяющие координаты прямоугольника (координаты верхнего левого угла, ширина, высота), используя текущий масштаб и стиль.

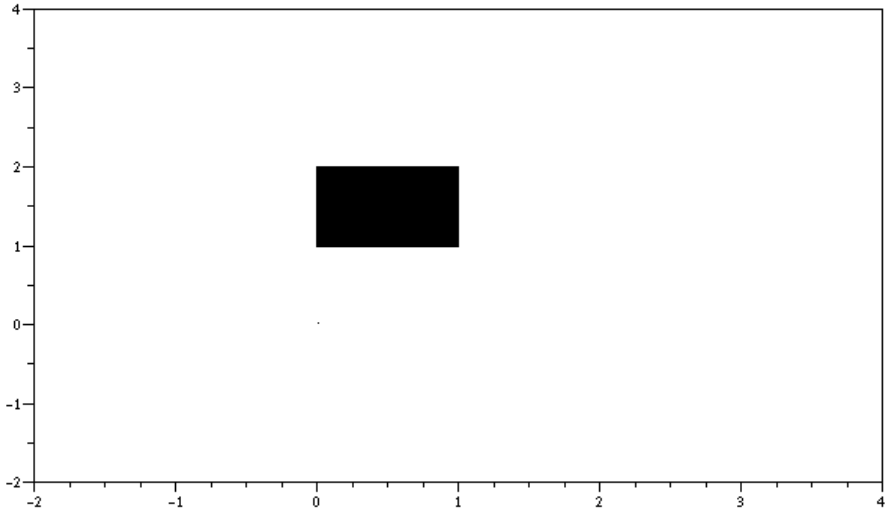
Пример.
```
xset("default");
plotframe([-2,-2,4,4],[4,6,2,6]); //для определения текущего масштаба
xrect(-1,1,2,2)
```
Результат:



Как изобразить закрашенный прямоугольник?

С помощью команды **xfrect**
Синтаксис
xrect(x,y,w,h)
xrect(rect), где **rect =[x,y,w,h]**
Параметры
x, **y**, **w**, **h** : четыре действительные числа, определяющие координаты прямоугольника (координаты верхнего левого угла, ширина, высота), используя текущий масштаб и стиль. Цвет заполнения определяется текущими параметрами.

Пример.
```
plotframe([-2,-2,4,4],[4,6,2,6]); //для определения текущего масштаба
xfrect(0,2,1,1)
```
Результат:

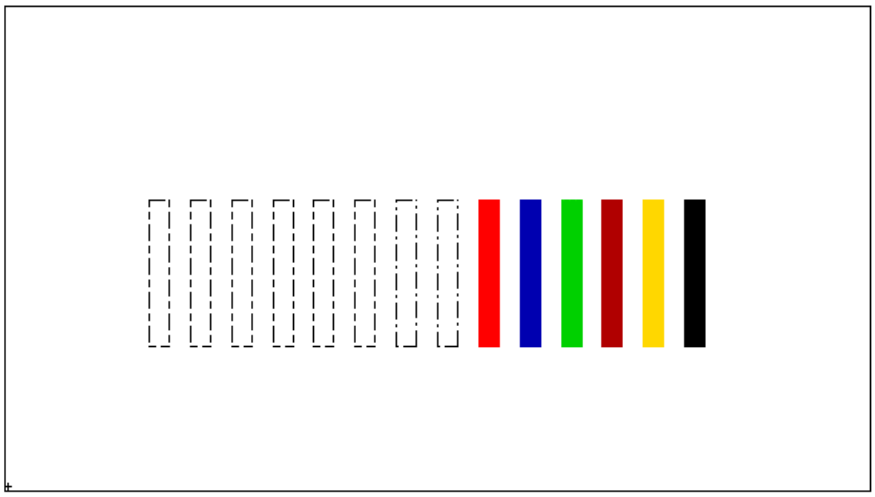


Как изобразить несколько заполненных или незаполненных прямоугольников?

С помощью команды **xrects**. Подробно смотри help.
Пример.

```
plot2d([-100,500],[-50,50],[-1,-1],"022");
cols=[-34,-33,-32,-20:5:20,32,33,34];
step=20;
x=400*(0:14)/14
step = (x(2)-x(1))/2
rects=[x;10*ones(x);step*ones(x);30*ones(x)];
xrects(rects,cols);
```

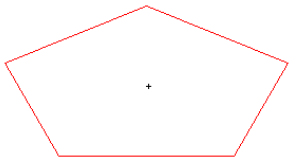
Результат:



Как нарисовать полилинии или полигон?

Способ 1.
С помощью команды **xpoly**.
Полилиния описывается с помощью векторов координат xv и yv.
Синтаксис
xpoly(xv,yv [,dtype [,close]])
Параметры
xv, **yv** :матрицы одного и того же размера (точки полилинии).
dtype : строка (стиль рисования). По умолчанию значение параметра равно **"lines"**. При этом точки полилинии соединяются отрезками. Второе возможное значение параметра **"marks"**. В этом случае изображаются только сами точки полилинии.
close : целое число. Если значение **close**=1, то полилиния замкнута. По умолчанию значение **close**=0.

Пример.
```
x=sin(2*%pi*(0:5)/5);
y=cos(2*%pi*(0:5)/5);
plot2d(0,0,-1,"010"," ",[-2,-2,2,2]); // для задания масштаба
xset("dashes",5);
xpoly(x,y,"lines",1)
```
Результат:



Способ 2.
С помощью команды **xfpoly**, изображающей закрашенный полигон.

Способ 3.
С помощью команды **xpolys**, изображающей группу полигонов.

Способ 4.
С помощью команды **xfpolys**, изображающей несколько закрашенных полигонов.

Как изобразить систему стрелок?

С помощью команды **xarrows**. Изображается несколько стрелок, заданных с помощью матриц. Возможен выбор стиля изображений.

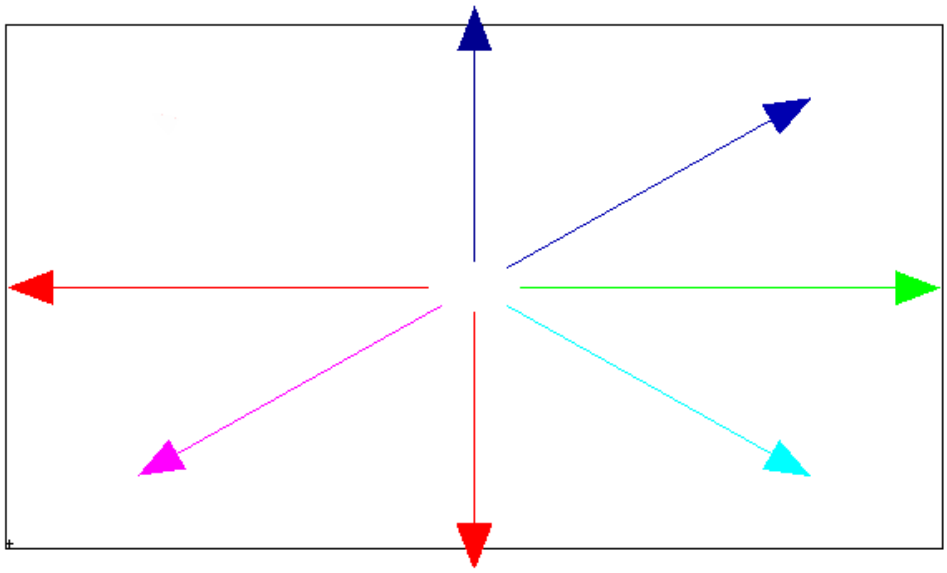
Синтаксис
xarrows(nx,ny,[arsize,style])
Параметры
nx ,**ny** : действительные вектора или матрицы одинакового размера

arsize : действительное число, определяющее размер головки стрелки. Значение по умолчанию можно получить, установив значение параметра **arsize** равным -1.

style : матрица или скаляр. Если **style** положительное число, то стиль "dash" используется для всех стрелок, если отрицательное - то этот стиль используется только для текущей стрелки. Если параметр **style** является вектором, то **style(i)** дает стиль для i-той стрелки.

Команда **xarrows** использует текущий графический масштаб, который вызывается функциями высокого уровня, такими как **plot2d** и т. п.

Пример.
```
x=2*%pi*(0:9)/8;
x1=[sin(x);9*sin(x)];
y1=[cos(x);9*cos(x)];
plot2d([-10,10],[-10,10],[-1,-1],"022")
xarrows(x1,y1,1,1:10)
```
Результат:

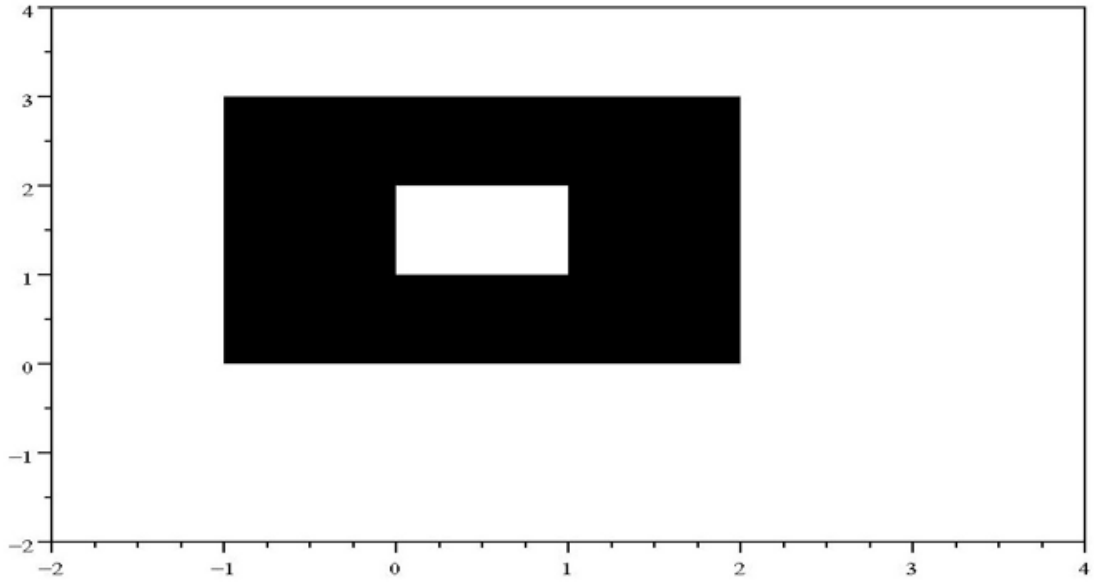


Как уничтожить (стереть) прямоугольную область?

С помощью команды **xclea**
Синтаксис
xclea(x,y,w,h)
Параметры
x, **y**, **w**, **h** : действительные числа, определяющие прямоугольную область. **x** и **y** определяют координаты верхнего левого угла прямоугольника, **w** - ширина, **h** - высота прямоугольной в текущем графическом окне. Масштаб задается предварительно выполненной командой типа **plot2d**.

Пример.
```
plotframe([-2,-2,4,4],[4,6,2,6]); //для определения текущего масштаба
xfrect(-1,3,3,3);
xclea(0,2,1,1)
```

Результат:



Как изобразить эллипс или его дугу?

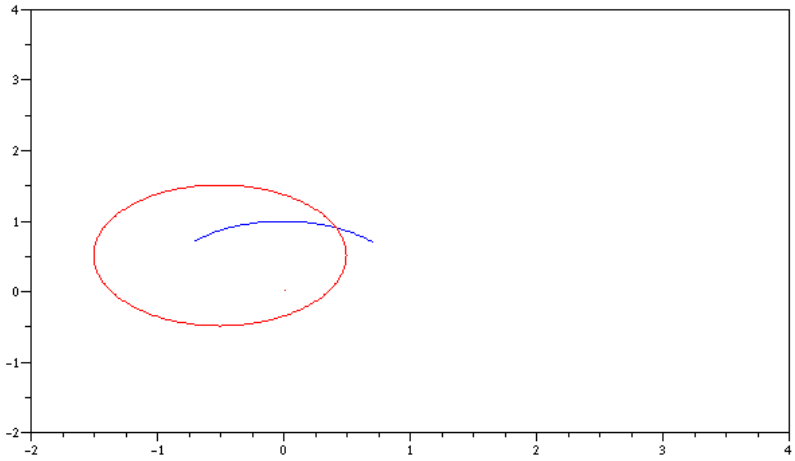
Способ 1.
С помощью команды **xarc**.
Синтаксис
xarc(x,y,w,h,a1,a2)
Параметры
x, **y**, **w**, **h** : четыре действительных числа, определяющие прямоугольник (координаты левого верхнего угла, ширина, высота). В этот прямоугольник будет вписываться эллипс. Если ширина и высота равны, будет получена окружность или ее сегмент (дуга).

a1, **a2** : действительные числа, определяющие сектор.
Сектор определяется следующим образом: начальный угол будет определяться как a1/64 градуса, а конечный угол как (a1+a2)/64 градуса.
При построении используется текущий масштаб и стиль.

Пример.

```
plotframe([-2,-2,4,4],[4,6,2,6]);
xset("dashes",2);
xarc(-1,1,2,2,45*64,90*64);
xset("dashes",5);
```

```
xarc(-1.5,1.5,2,2,0,360*64)
```
Результат:



Способ 2.
С помощью команды **xfarc**.
Изображает закрашенный с помощью текущего стиля сегмент эллипса.

Способ 3.
С помощью команды **xarcs**.
Изображает несколько дуг различных эллипсов с помощью одной команды.

Способ 4.
С помощью команды **xfarcs**.
Изображает несколько закрашенных сегментов различных эллипсов с помощью одной команды.

Изображение текста в графическом окне

Как изобразить текст в графическом окне?

Способ 1.
С помощью команды **xstring**
Синтаксис
xstring(x,y,str,[angle,box])
Параметры
x, **y** : действительные скаляры, определяющие нижний левый угол надписи.
str : текстовая матрица.
angle : действительное число, определяющие угол по часовой стрелке в градусах; по умолчанию =0.
box : целое число, по умолчанию =0.

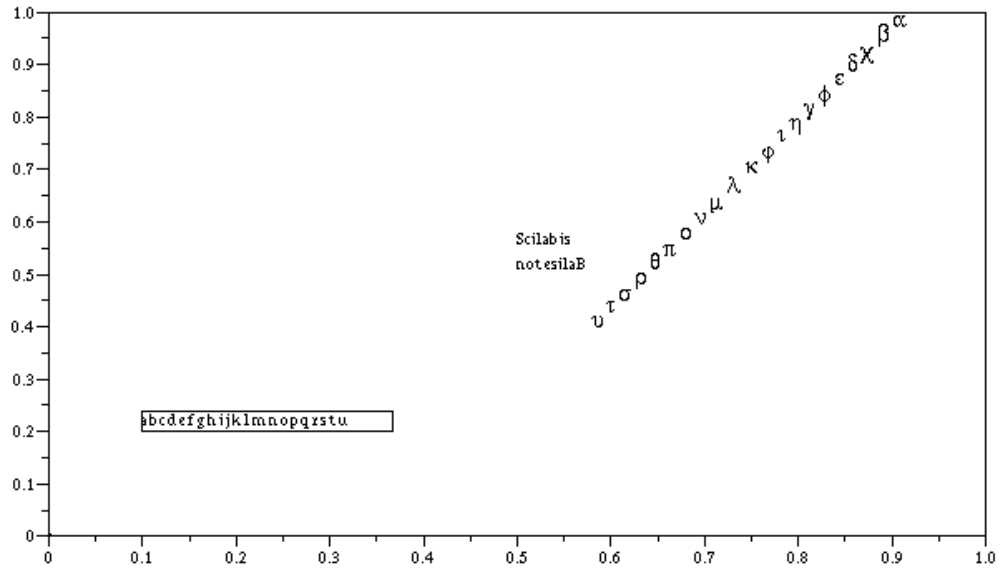
Масштаб определяется текущей графической шкалой. Элементы матрицы **str** выводятся в графическое окно. Знак "точка с запятой " означает перенос на другую строку. Если угол задан, то он определяет наклон изображаемого текста относительно горизонтального.

Если **box** равен 1 и угол равен 0, то вокруг текста будет нарисована рамка, минимально возможного размера. Для задания произвольной рамки пользуйтесь командой **xstringb**.

Пример.

```
xset("default");
plot2d([0;1],[0;1],0)
xstring(0.5,0.5,["Scilab" "is"; "not" "esilaB"])
alphabet=["a" "b" "c" "d" "e" "f" "g" ..
"h" "i" "j" "k" "l" "m" "n" ..
"o" "p" "q" "r" "s" "t" "u"];
xstring(0.1,0.2,alphabet,0,1);
xset("font",1,3); // изменение размера шрифта
xstring(0.9,0.95,alphabet,135)
```

Результат:



Способ 2.

С помощью команды **xstringb**. Текст изображается в рамке (box) заданных размеров. Сама рамка не изображается. Если мы хотим ее видеть, ее следует изобразить отдельной командой.

Синтаксис

xstringb(x,y,str,w,h,[option])

Параметры

x, **y**, **w**, **h** : вектор из 4-х действительных параметров, определяющих размеры рамки (левый верхний угол, ширина, высота в текущем масштабе) .

str : матрица строк.

option : символьная строка

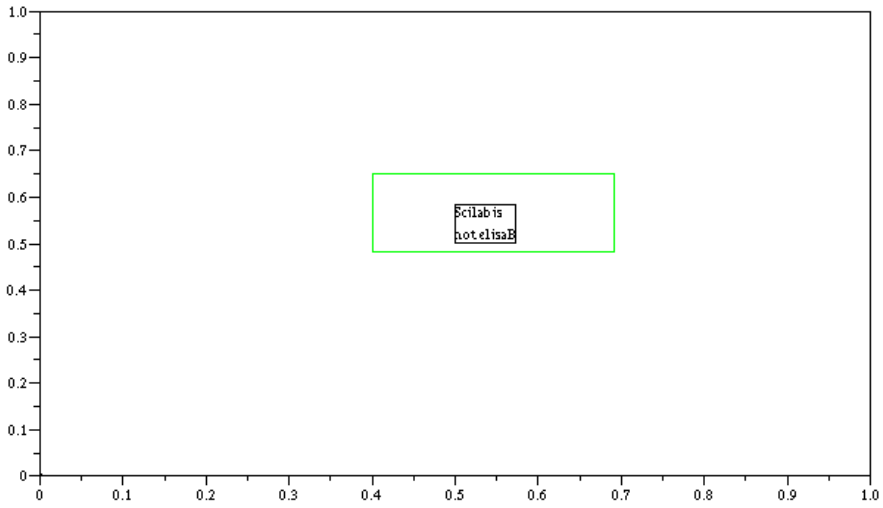
Если значение **option** равно **"fill"**, то значение размеров шрифта принимается максимально возможным для данной рамки.

Пример.

```
plot2d(0,0,[1,-1],'012',' ',[0,0,1,1]);
mat=['Scilab','is';'not','Basile'];
xstringb(0,0.5,mat,1,0.5);
xrect(0,1.0,1,0.5);
```

```
xstringb(0,0,mat,1,0.5,'fill');
xrect(0,0.5,1,0.5);
```

Результат:



Способ 3.
С помощью команды **xnumb**, если мы хотим написать в графическом окне цифры, заданные в числовом, а не в строковом формате.
Синтаксис
xnumb(x,y,nums,[box,angle])
Параметры
x, **y**, **nums** : векторы одинакового размера.
box : целое число. Если =1, то вокруг числа изображается рамка.
angle : необязательный параметр, являющийся вектором того же размера, что и **x**. Определяет угол поворота числа.

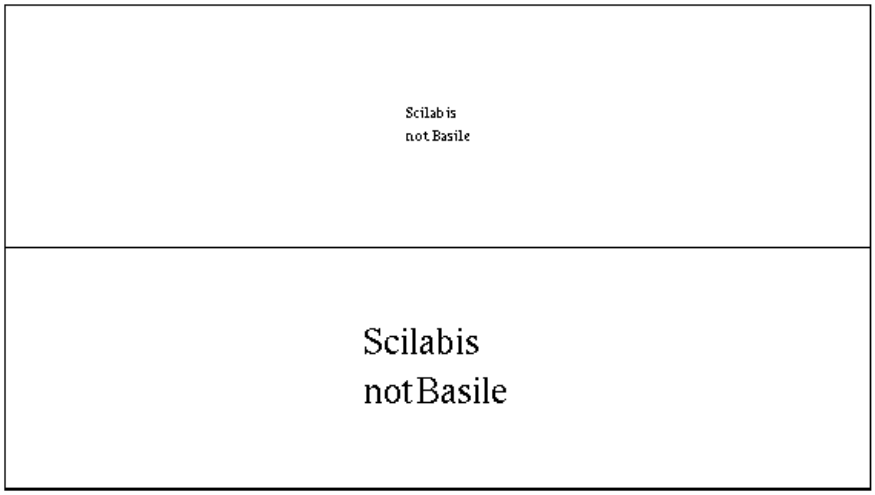
Как узнать размер рамки (box), окружающей текст в команде string?

С помощью команды **xstringl**. Эта команда вычисляет параметры рамки для конкретного заданного текста, является полезным при программировании и использовании драйвера Postscript.
Синтаксис
rect=xstringl(x,y,str)
Параметры
rect : вектор из 4 действительных скаляров, определяющих рамку. **rect=[x,y,w,h]**
str : матрица строк.
x, **y** : координаты верхнего левого угла, w - ширина, h - высота в текущем графическом масштабе матрицы строк str.

Пример.

```
xset("default");
plot2d([0;1],[0;1],0)
str=["Scilab" "is";"not" "elisaB"];
xstring(0.5,0.5,str,0,1);
r=xstringl(0.5,0.5,str);
```

```
xset("dashes",3);
xrect([0.4 0.65 4*r(3) 2*r(4)]')
```
Результат:



Графические функции для решения линейных систем

В Scilab существуют некоторые специализированные графические команды, помогающие в решении систем уравнений. Все они используются в комбинации с командой **syslin**.

Список этих функций:
bode
gainplot
nyquist
m_circle
chart
black
evans
plzr

По-видимому, пакет был изначально приспособлен для решения радиотехнических проблем. Например, **bode** - изображает амплитуду и фазу частоты отклика линейной системы. Это нечто специализированное из радиотехники... Неспециалистам в радиотехнике, думаю, можно не смотреть. Остальные того же сорта: нечто, посвященное линейным системам. В данном пособии эти команды подробно не рассматриваются. В документации пакета в файле **signal.pdf** широко рассматривается применение Scilab для задач обработки сигналов.

Интерактивное редактирование в графическом окне

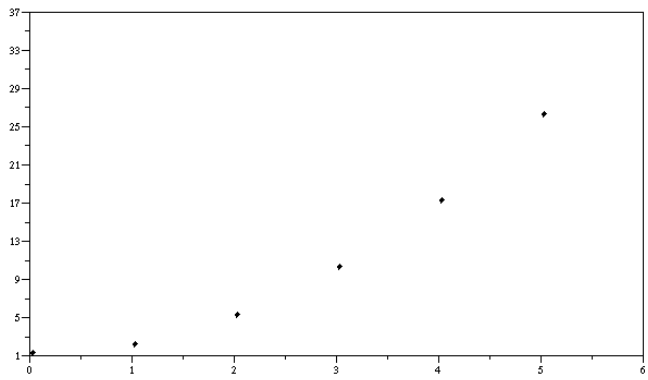
Основные команды интерактивного редактирования:
edit_curv - интерактивный графический редактор (для редактирования кривой)
gr_menu - более простой интерактивный графический редактор (для создания объектов)
locate - создание объектов с помощью мыши

Можно ли интерактивно (с помощью мыши) изменить (добавить или удалить) данные в графическом окне?

С помощью команды **edit_curv**, являющейся интерактивным графическим редактором. Удобное для работы с дискретными данными.
Синтаксис
[x,y,ok,gc] = edit_curv(y)
[x,y,ok,gc] = edit_curv(x,y)
[x,y,ok,gc] = edit_curv(x,y,job)
[x,y,ok,gc] = edit_curv(x,y,job,tit)
[x,y,ok,gc] = edit_curv(x,y,job,tit,gc)
Параметры
x : вектор x координат
y : вектор y координат
job : символьная переменная, которая может принимать одно из следующих значений: 'a','x','y'
tit : вектор из трех строк, определяющий "легенду" для кривой
gc : список переменных графического окна: **gc=list(rect,nax)**
ok : индикатор. Если ok==%t пользователь возвращается по кнопке меню "ok", в противном случае - пользователь возвращается с помощью команды меню графического окна **"abort"**.
Пример.

```
xset("default");
x=[0,1,2,3,4,5,6];
fun=x^2+1;
xbasc();
plot2d(x,fun, -4)
```

Результат:



Теперь выполните:

```
x_new=edit_curv(x)
```

На экране мы получили вспомогательную сетку с выделенной выбранной нами точкой для редактирования x=1 и y=1. С помощью манипулятора "мышь" мы можем передвигать эту точку по экрану. В нашем случае будет меняться только координата "x". Команда меню графического окна **Edit-Undo** отменяет выбранную точку. Команды меню **Edit-Size**

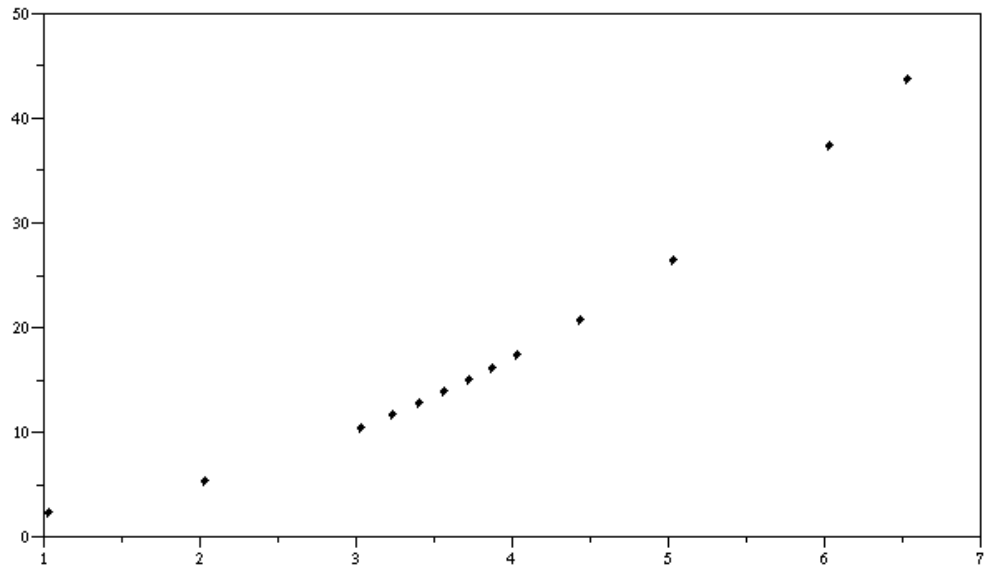
позволяет расширить возможность выбора координат за пределами первоначальной области определения. Посмотрите и другие возможности команд меню. Когда Вас удовлетворит новое положение точки, нажмите правую кнопку мыши, а затем из строки меню графического окна выберите команду **Edit-Ok**. Окно исчезнет (закроется). При этом в окне Scilab мы получим, например, такой результат:

```
xnew =
!  1.         !
!  2.         !
!  3.         !
!  3.2008693  !
!  3.3736949  !
!  3.5364835  !
!  3.6928893  !
!  3.8378936  !
!  4.         !
!  4.4037803  !
!  5.         !
!  6.         !
!  6.4995199  !
!  7.         !
```

Сюда входят координаты "x" и имевшихся до редактирования точек, и координаты "x" новых точек. Заметим, что "y" -координаты точек по никуда не записались. Теперь мы можем пересчитать функцию для нового вектора данных и построить функцию заново:

```
fun=xnew^2+1;
plot2d(xnew,fun, -4)
```

Результат:



Это пример не исчерпывает всех возможностей интерактивного графического редактора.

Как рисовать вручную в графическом окне простые графические объекты (прямоугольники, эллипсы и др.)?

С помощью команды **gr_menu**. Это интерактивный графический редактор для создания простых объектов.
После выполнения команды **gr_menu()** в строке меню графического окна появятся три новые команды **Objects**, **Settings** и **Edit**. На первый взгляд новые возможности похожи на PaintBrush и позволяют вручную рисовать эллипсы, стрелки, прямоугольники, менять стиль, цвет карандаша, цвет заливки и т.д. Для выхода из графического редактора **gr_menu** используйте меню графического окна **Edit-Exit**. Созданные графические объекты могут быть сохранены в параметр типа список (list) в качестве значения выхода.
Синтаксис
[sd1]=gr_menu([sd,flag,no_frame])
Параметры
sd : список (list) (вывод из gr_menu), или вектор из четырех длин [xmin,ymin,xmax,ymax] (границы графика).
sd1 : список (графические объекты, созданные во время сеанса работы **gr_menu**)
flag, **no_frame** : целые со значениями 0 и 1. Используйте **flag=1** для неинтерактивной моды (т.е. для перерисовки графики, сохраненной с помощью редактора **gr_menu**) и **no_frame=1** для создания фрейма вокруг графики редактора **gr_menu**.

[sd]=gr_menu([xmin,ymin,xmax,ymax])
: открывает **gr_menu** во фрейме с заданными размерами [xmin,ymin,xmax,ymax].
[sd]=gr_menu(); : открывает **gr_menu** во фрейме с размерами [0 0 100 100].

Замечание: **gr_menu()** может редактировать и пустое окно, очищенное с помощью **xbasc()**.

[sd]=gr_menu(sd) : перерисовывает графическое окно, сохраненное в переменной типа список **sd** (например, с помощью конструкции [sd]=gr_menu([xmin,ymin,xmax,ymax]) и входит в режим интерактивного редактирования

[sd]=gr_menu(sd,1) : только изображает графику, предварительно сохраненную в переменной sd. При этом возможности графического редактирования и создания простых объектов не возникают.

[sd]=gr_menu(sd,1,1) : рисует графику, сохраненную предварительно в переменной **sd**, и не добавляет в нее фреймов.

Пример использования.
1) Выполните:
```
xmin=0;xmax=300;ymin=0;ymax=500 // это помогает ориентироваться в размерах объекта
sd=gr_menu([xmin,ymin,xmax,ymax])
```
В строке меню графического окна появились новые команды: **Objects**, **Settings** и **Edit**.

2) Выберите команду **Object-frectangle** и при помощи мышки создайте закрашенный прямоугольник. Когда вид прямоугольника Вас удовлетворит, нажмите еще раз на левую кнопку мышки. Теперь прямоугольник создан.

3)Выберите команду **Edit-Exit**. Изображение из графического окна исчезнет, а в главном окне появится, образовавшиеся в результате Ваших манипуляций значения параметра `sd1`.

4) Закройте графическое окно и выполните:
```
gr_menu(sd,1,1)
```

Вы увидите, что изобразилось графическое окно и в нем изображение, все параметры которого были сохранены в переменной `sd`.

Как из графического окна ввести данные в Scilab с помощью манипулятора мышь?

Способ 1.
С помощью команды **locate** - выбор мышью положения точек, которые записываются в заданную переменную.
Синтаксис
x=locate([n,flag])
Параметры
x : матрица размером (2,n1). n1=n если параметр n задан.
n, **flag** : целые числа.

Вводятся координаты одной или более точек с помощью манипулятора мышь. Если n>0, то выбираются n точек и результат присваивается матрице x. Если n<0, точки будут выбираться с помощью правой кнопки мыши до тех пор, пока пользователь не остановит процесс выбора точек с помощью левой кнопки мыши. Последняя точка, на которую кликали манипулятором мышь, не записывается. Если flag=1, то на месте выбранных точек в графическом окне появится отметка, если flag=0, то новые точки в графическом окне не видны..

Пример.
```
rect=[2,25,25,50];
tics=[1,8,1,5];
plotframe(rect,tics) // это для выбора масштаба окна
my_coord=locate(1,3)
```
После этого в графическом окне наведем указатель мыши приблизительно на точку с координатами (4, 40) и нажмем на левую кнопку мыши. Точка выбрана. Поскольку n=1, мы вводим только одну точку. Координаты выбранной точки будут записаны в вектор-строку `my_coord`.
У меня получился результат:
```
my_coord =
! 3.8781362 !
! 39.986188 !
```
Для выбора двух точек и записи их координат следует применить команду `my_coord=locate(2,3)`.
Команды `my_array=locate(-5)` позволит выбрать неограниченный набор точек и записать их в матрицу `my_array`. После выбора местоположения каждой точки следует нажимать правую кнопку мыши, а для прекращения процедуры выбора следует нажать левую кнопку мыши.
Замечание:
Конструкция **x=locate()** эквивалентна конструкции **x=locate(n)**, где n<0, но иногда приводит к ошибкам.

Способ 2.
С помощью команды ожидания нажатия мыши **xclick**.
Эта команда позволяет выбрать и получить координаты только одной выбранной точки в графическом окне. Но зато можно получить дополнительные сведения о номере графического окна, положении кнопки мыши и т. д.

Пример.
```
t=xclick()
```
После выбора точки, получим результат:
```
t = ! 0. 8.9 28.618188 !
```

Первый элемент вектора-строки `t` указывает на то, что была нажата левая кнопка мыши. Ей соответствует число 0.
Полный синтаксис смотрите с помощью команды **help xclick**.

Как узнать текущее положение мыши?

С помощью команды **xgetmouse**.
Синтаксис
rep=xgetmouse([flag])
Параметры
rep : вектор размера 3, [x,y,ibutton].
flag : целое число. Если этот параметр присутствует, очередь событий после нажатия кнопки мыши не очищается, когда вводится команды **xgetmouse**.

Значение параметра **ibutton** указывает на действие кнопки мыши в этой точке.
- если **ibutton** равно -5, -4 или -2 , то активирована соответственно левая, средняя или правая кнопка мыши
- если **ibutton** равно 0, 1 или 2 , то нажата соответственно левая, средняя или правая кнопка мыши.

Если мышь находится вне графического окна, команда **xgetmouse** находится в режиме ожидания.

Пример.
```
plot(1:10)
```
Наведите в графическом окне несколько раз на разные точки. Нажимайте при этом разные кнопки мыши. При этом на самом графическом окне никаких новых отметок не появится. Постарайтесь запомнить, куда Вы нажимали. Для этого упражнения (для простоты проверки) попробуйте приблизительно ставить точки приблизительно по прямой. После этого последовательно выполните команды:
```
rep_1=xgetmouse(0)
rep_2=xgetmouse(0)
rep_3=xgetmouse(0)
```
и Вы получите все положения мыши. Если Вы выбрали всего две точки на графике, то на выполнении третьей команды `rep_3=xgetmouse(0)` Вас вернут в графическое окно.

3D графика

Как изобразить 3D поверхность?

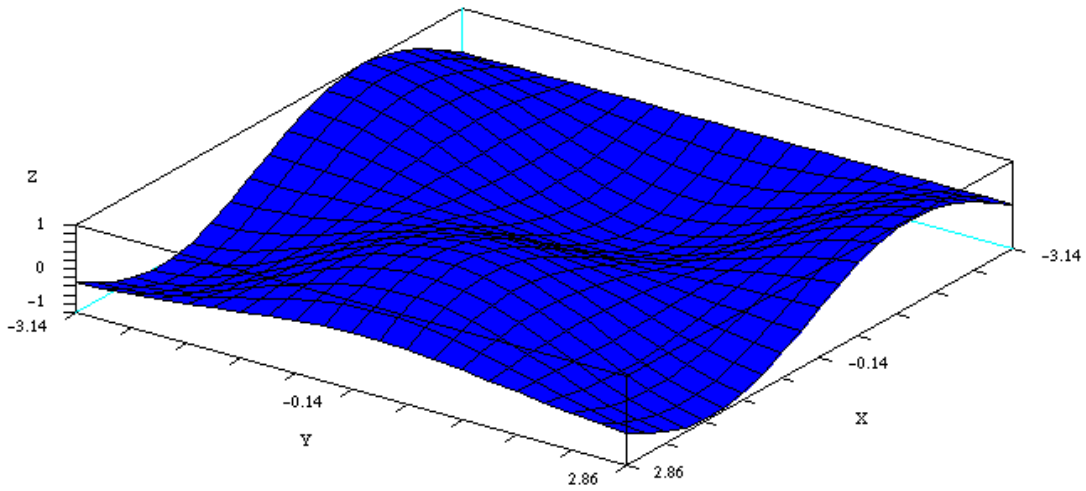
Способ 1.
С помощью команды **plot3d**. Команда создает 3D график по точкам, заданным матрицами x, y и z.

Способ 2.
С помощью команды **plot3d1**. Команда создает 3D график по точкам, заданным матрицами x, y и z с помощью уровней цвета. Вещь в общем избыточная: величина координаты z дополнительно еще и покрашена, в зависимости от принимаемого значения z.

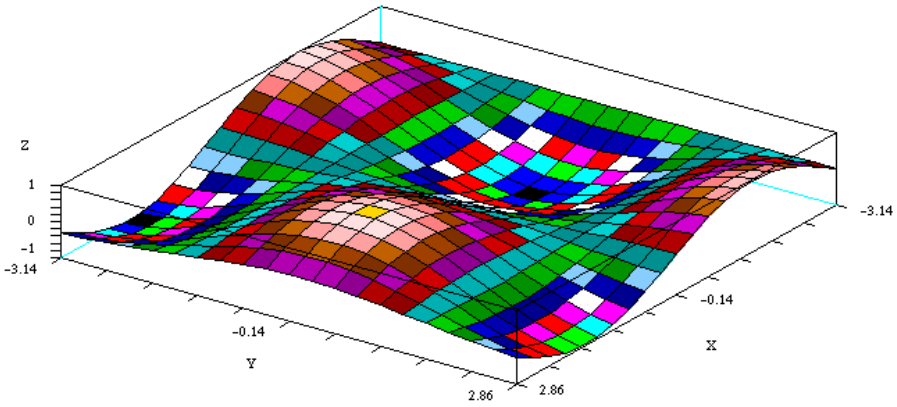
Способ 3.
С помощью команды **fplot3d**. Это аналог команды **fplot3d**, но изображаемая поверхность задана с помощью внешней функции.

Способ 4.
С помощью команды **fplot3d1**. Это аналог команды **plot3d1**, но изображаемая поверхность задана с помощью внешней функции.
Синтаксис этих команд смотри с помощью **help**.

Пример 1.
```
t=-%pi:0.3:%pi;
plot3d(t,t,sin(t)'*cos(t),35,45,'X@Y@Z',[2,2,4]);
```
Результат:



Пример 2.
```
t=-%pi:0.3:%p
i;plot3d1(t,t,sin(t)'*cos(t),35,45,'X@Y@Z',[2,2,4]);
```
Результат:



Пример 3.

```
deff('[z]=surf(x,y)','z=sin(x)*cos(y)');
t=-%pi:0.3:%pi;
fplot3d(t,t,surf,35,45,"X@Y@Z");
```

Пример 4.

```
deff('[z]=surf(x,y)','z=sin(x)*cos(y)');
t=-%pi:0.3:%pi;
fplot3d1(t,t,surf,35,45,"X@Y@Z");
```

Как построить 3D кривую параметрически заданной функции?

Способ 1.
С помощью команды **param3d**.
Синтаксис
param3d(x,y,z,[theta,alpha,leg,flag,ebox])
Параметры
x, **y**, **z** : три вектора одного и того же размера (точки параметрической кривой)

theta, **alpha** : действительные числа, являющиеся сферическими координатами точки.

leg : строка, определяющая заголовки для каждой из осей, разделенные сепараторами @ ("my_X@my_Y@my_Z".

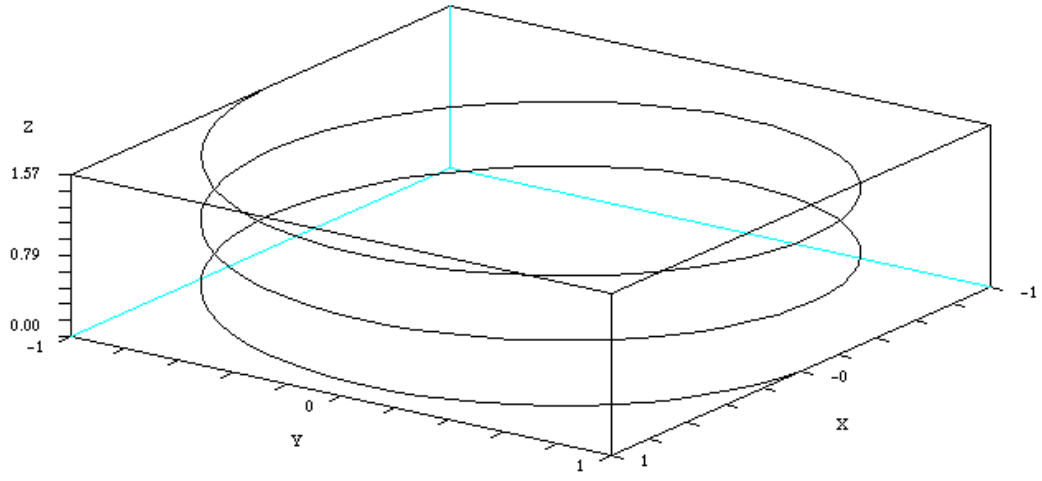
flag=[type,box] ,где параметры **type** и **box** означают то же, что и в команде **plot3d**.
type : целое число. Этот параметр ответственен за масштабирование. Может принимать значения 0, 1, 2, 3, 4, 5, 6. Подробно смотри **help**.
box : целое число, определяющее свойства фрейма вокруг графика). Может принимать значения 0, 1, 2, 3, 4. Подробно смотри **help**.

ebox :вектор. Задает трехмерный объем пространства в котором будет построен график. **ebox=[xmin,xmax,ymin,ymax,zmin,zmax]**. Используется , если в параметре **flag** значение **type** равно 1.

Пример.

```
t=0:0.1:5*%pi;
param3d(sin(t),cos(t),t/10,35,45,'X@Y@Z',[2,4]);
```

Результат:

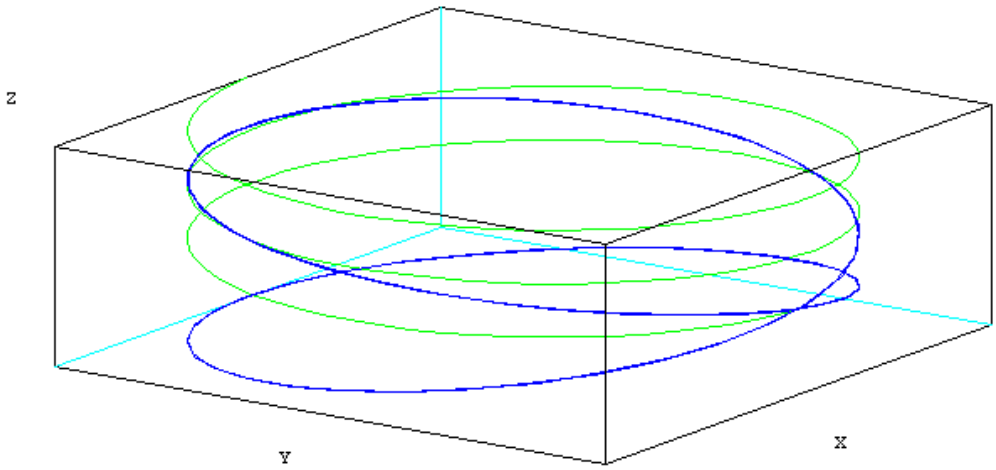


Способ 2.
С помощью команды **param3d1**, если нам нужно изобразить несколько функций от одних и тех же аргументов `x` и `y` с помощью одной команды. Аргумент `z` может являться списком и цветами (тип list(z,colors)). Для одиночной функции будет иметь тот же результат, что и **param3d**.

Пример.

```
t=0:0.1:5*%pi;
t=t';
param3d1([sin(t),sin(2*t)],[cos(t),cos(2*t)],list([t/10,sin(t)],[3,2]),35,45,
"X@Y@Z",[2,3])
```

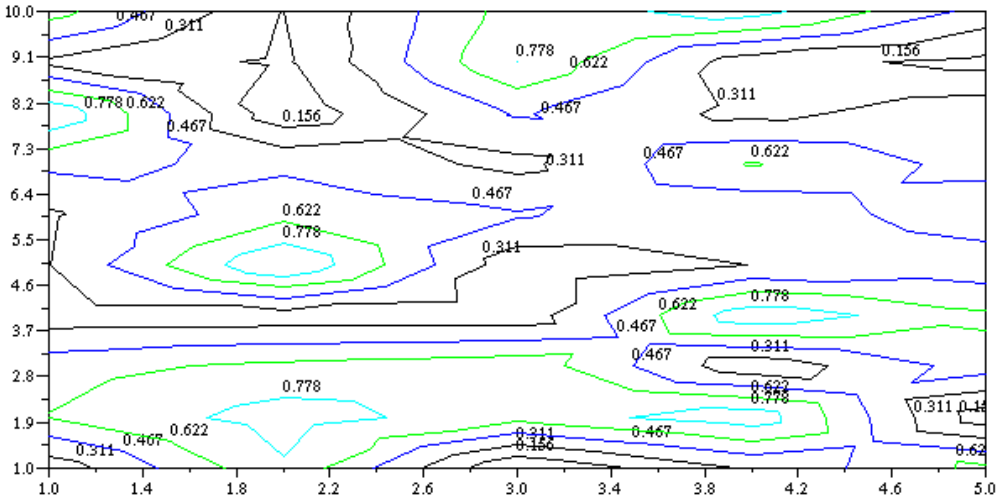
Результат:



Как изобразить пространственную кривую в виде изолиний на плоскости?

Способ 1.
С помощью команды **contour**. Пространственная кривая задается как z=f(x,y).

Пример.
```
contour(1:5,1:10,rand(5,10),5);
```
Результат:



Способ 2.
С помощью команды **contour2d**.

Пример.
```
contour2d(1:10,1:10,rand(10,10),5,1:5,"011"," ",[0,0,11,11]);
```

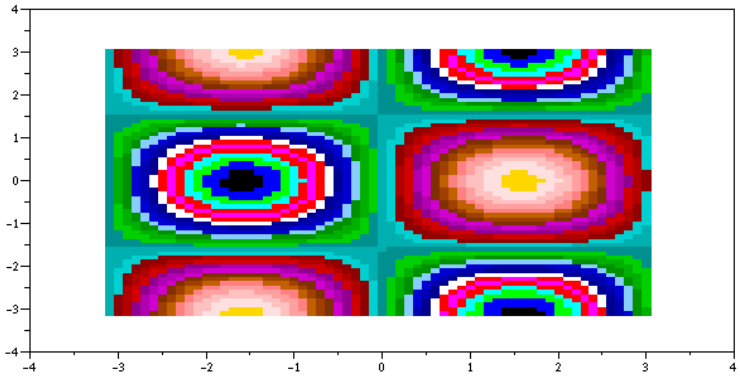
Способ 3.
С помощью команды **fcontour**. Аналог команды **contour**, но пространственная кривая задана внешней функцией.

Пример.
```
deff('[z]=surf(x,y)','z=x**2+y**2');
fcontour(-1:0.1:1,-1:0.1:1,surf,10);
```

Способ 4.
С помощью команды **fcontour2d**. Аналог команды **countour2d**, но пространственная кривая задана внешней функцией.

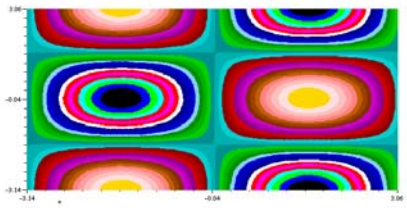
Как изобразить 3D поверхность на 2D плоскости в виде заливки изообластей цветом?

Способ 1.
С помощью команды **grayplot**.
Пример.
```
t=-%pi:0.1:%pi;
m=sin(t)'*cos(t);
grayplot(t,t,m);
```
Результат:



Способ 2.
С помощью команды **fgrayplot**. Аналог **grayplot**, но пространственная кривая задана внешней функцией.
Пример.
```
deff('[z]=surf(x,y)','z=x**2+y**2');
fgrayplot(-1:0.1:1,-1:0.1:1,surf);
```

Способ 3.
С помощью команды **Sgrayplot**. Это то же, что и **grayplot**, но цвета переходят в друг друга постепенно, без резкой границы (smooth).
Пример.
```
t=-%pi:0.1:%pi;
m=sin(t)'*cos(t);
Sgrayplot(t,t,m);
```
Результат:



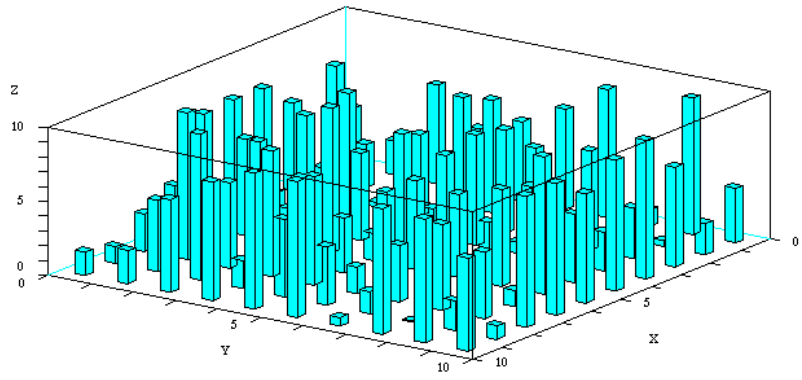
Способ 4.
С помощью команды **Sfgrayplot**. Аналог **Sgrayplot**, но пространственная кривая задана внешней функцией.

Как нарисовать 3D гистограмму?

С помощью команды **hist3d**
Пример.

```
hist3d(10*rand(10,10))
```

Результат:

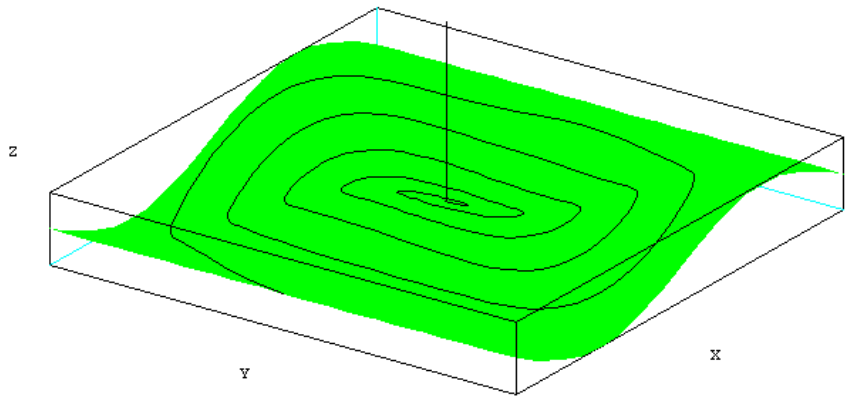


Построение смешанных графиков из 2D и 3D поверхностей

Пример.

```
r=(%pi):-0.01:0;
x=r.*cos(10*r);y=r.*sin(10*r);
deff("[z]=surf(x,y)","z=sin(x)*cos(y)");
t=%pi*(-10:10)/10;
fplot3d(t,t,surf,35,45,"X@Y@Z",[-3,2,3]);
z=sin(x).*cos(y);
[x1,y1]=geom3d(x,y,z);
xpoly(x1,y1,"lines");
[x1,y1]=geom3d([0,0],[0,0],[5,0]);

// xsegs(x1,y1);
```

Замечание: Команда **geom3d** служит для построения проекций 3D изображений на плоскость 2D.
Результат:



Изображение нескольких рисунков в одном графическом окне

С помощью команды **subplot**. Графическое окно разделяется на "матрицу" из отдельных областей ("подокон"), в каждой из которых можно будет изобразить отдельный график.
Синтаксис

```
subplot(m,n,p)
subplot(mnp)
```

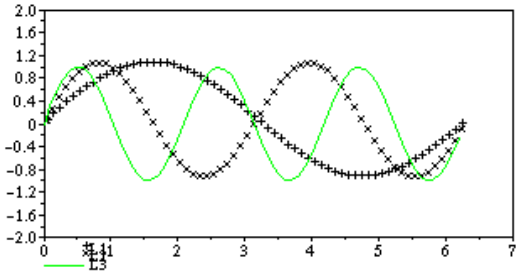
Параметры
m, **n**, **p** : положительные целые числа
mnp : целое число. (Та же конструкция, что и **m**, **n**, **p** , но с пропущенными запятыми)

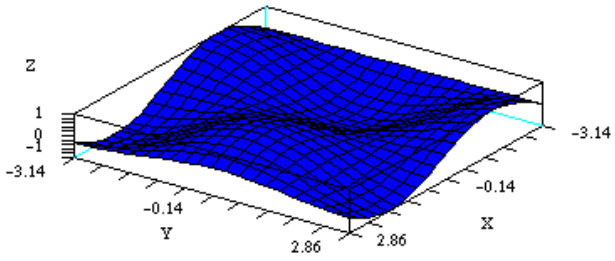
Графическое окно разбивается на матрицу (`m` на `n`) подокон. Текущим выбирается окно с номером `p`.

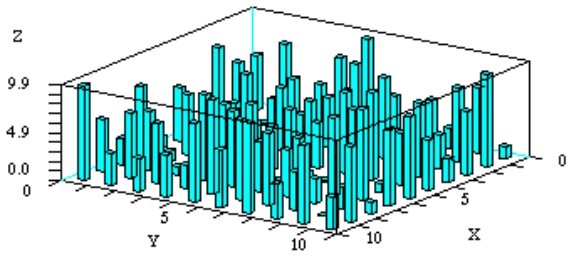
Пример.

```
subplot(221)
plot2d()
subplot(222)
plot3d()
subplot(2,2,4)
hist3d()
```

Замечание: В окне с номером "3" мы специально ничего не нарисовали.
Результат:







Пример сложных построений.

```
// Создается несколько окон с различными рисунками
str_l=list();
//
str_l(1)=['plot3d1();';
'title=[''plot3d1 : z=sin(x)*cos(y)''];';
'xtitle(title,'' '','' '');'];
//
str_l(2)=['contour();';
'title=[''contour ''];';
'xtitle(title,'' '','' '');'];
//
str_l(3)=['champ();';
```

```
'title=[''champ ''];';
'xtitle(title,'' '','' '');'];
//
str_l(4)=['t=%pi*(-10:10)/10;';
'deff(''[z]=surf(x,y)'',''z=sin(x)*cos(y)'');';
'rect=[-%pi,%pi,-%pi,%pi,-5,1];';
'z=feval(t,t,surf);';
'contour(t,t,z,10,35,45,''X@Y@Z'',[1,1,0],rect,-5);';
'plot3d(t,t,z,35,45,''X@Y@Z'',[2,1,3],rect);';
'title=[''plot3d and contour ''];';
'xtitle(title,'' '','' '');'];
//
// Пока все команды записаны в виде текстовых инструкций в переменную str_l.
// Никакие действия не предпринимались.
//
// Цикл запускающий поочередное выполнение каждой инструкции в отдельном окне
for i=1:4,xinit('d7a11.ps'+string(i)');
execstr(str_l(i)),xend();end
```

Выполните этот пример в Scilab самостоятельно, и Вы получите четыре графических окна с разнообразными представлениями одной и той же функции `z=sin(x)*cos(x)`.

Вывод содержимого графического окна на печать

Как получить бумажную версию графического окна?

Утверждается, что из графического окна следует выполнить **File-Print(Scilab)**. У меня на сетевом принтере HPLaser 6L вместо картинки распечатывается какая-то кодовая таблица. Однако, если в качестве принтера установить Acrobat Distiller, то можно создать полноценный **pdf** файл. Затем из Adobe Acrobat полученный файл можно распечатать на принтере.

Как внедрить Scilab графику в документ Latex?

Для внедрения в пакет Latex обычно требуются рисунки в формате **eps** или **ps**. Описание Scilab советует поступать следующим образом:

```
driver('Pos")
xinit("d:\zzz.ps")
t=-%pi:0.3:%pi;
plot3d(t,t,sin(t)'*cos(t),35,45,'X@Y@Z',[2,2,4]);
xend()
```

В результате создается файл d:\zzz.ps, но он не хочет читаться даже с помощью Gostscript.

Еще вариант:

```
driver("Rec")
t=-%pi:0.3:%pi;plot3d(t,t,sin(t)'*cos(t),35,45,'X@Y@Z',[2,2,4]);
xbasimp(0,"d:\f_z.ps")
```

В результате этот файл тоже не читается. По-видимому, здесь имеется ошибка пакета. Однако, можно создать необходимый файл вручную с помощью команд строкового меню графического окна **File-Export**. Полученные файл читаются даже с помощью программы Adobe PhotoShop.