GnuP

СОДЕРЖАНИЕ

GnuP

GnuP – высокоуровневая кроссплатформенная библиотека, позволяющая легко и лаконично интегрировать графику в код программы, написанной на C++ или fortran. Для её использования нужен файл *Gnup.h* (C++) или *Gnup.f95* (fortran) и *gnuplot* - бесплатная программа для построения двух- и трехмерных графиков (http://www.gnuplot.info/).

Установка gnuplot

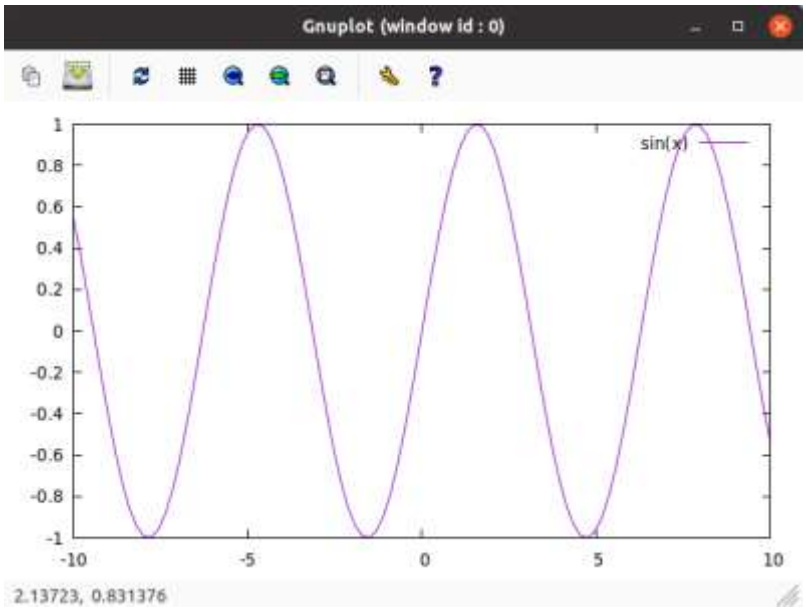
1. Установка на Linux

```
sudo apt install gnuplot
```

Чтобы проверить, что gnuplot установлен, введите в терминале

```
gnuplot
```

После этого запустится терминал программы gnuplot, в котором можно ввести, например, команду `plot sin(x)`. Результат:



2. Установка на Windows

Ссылка для скачивания: https://sourceforge.net/projects/gnuplot/files/gnuplot/

После скачивания нужно добавить gnuplot в PATH. Для этого (Windows 10):

Все параметры -> Система -> Дополнительные параметры системы -> Переменные среды… -> Переменные среды -> два раза кликнуть по Path -> Создать.

После вводим полный путь до папки, где расположен gnuplot. Например:

C:\Program Files\gnuplot\bin

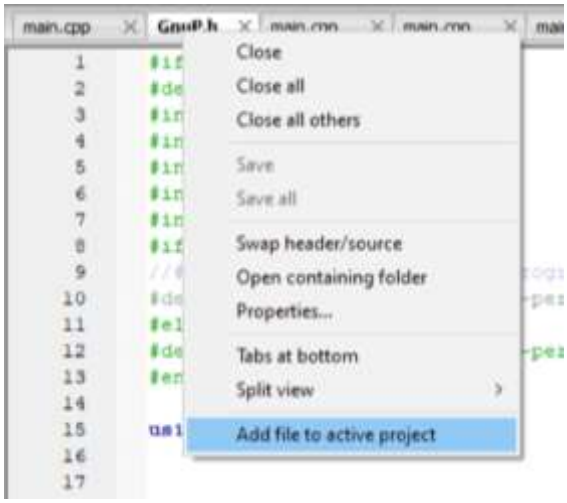
Далее везде нажимаем ОК.

Использование в C++

Для начала работы к своему проекту нужно подключить файл GnuP.h (он должен находиться в одной папке с вашим cpp-файлом):

```
#include "GnuP.h"
```

Если используете Code::Blocks, подключить GnuP.h можно, нажав правой кнопкой мыши на вкладку «GnuP.h» и выбрав «Add file to active project»:



GnuP готов к работе.

Прежде всего нужно создать объект класса GnuP:

```
GnuP p;
```

Теперь можно обращаться к методам этого объекта для построения графиков. Все они будут строиться разом и располагаться на одном полотне.

Методы:

1. **plot**

Этот метод не имеет параметров. После его вызова на экран будут выведены двумерные графики. Вызывать в самом конце, после всех остальных методов.

2. **plot_3d**

Как метод plot, но предназначен для отрисовки трехмерных графиков.

3. **plotArray**

График строится по данным из массивов. Возможные аргументы метода:

- ✓ n, n1, n2, ... – размерности массивов
- ✓ x, x1, x2, ... – массивы с координатами точек по оси абсцисс
- ✓ y, y1, y2, ... – массивы с координатами точек по оси ординат
- ✓ type0, type1, ... – любые числовые типы данных

1) `plotArray(int n, type0 x, type1 y)`

Построение 1 графика.

2) `plotArray (int n, type0 x, type1 y1, type2 y2,…)`

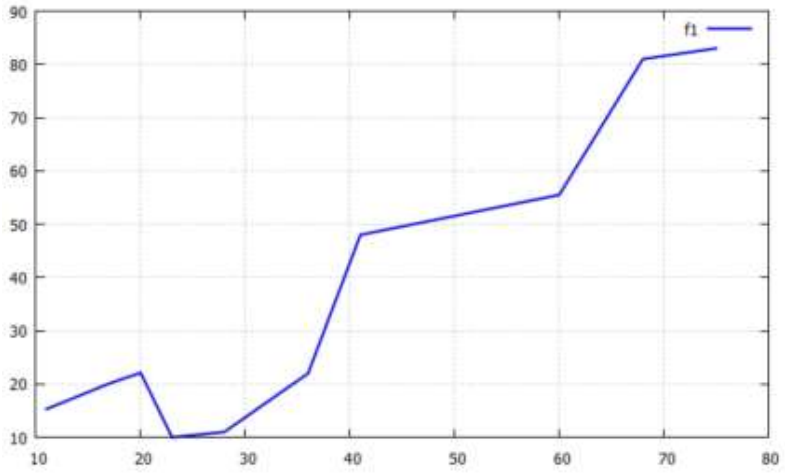
Можно передавать от 1 до 5 различных массивов y1, …, y5 одинакового размера со значениями функций.

3) `plotArray (int n1, type11 x1, type12 y1, int n2, type21 x2, type22 y2, ...)`

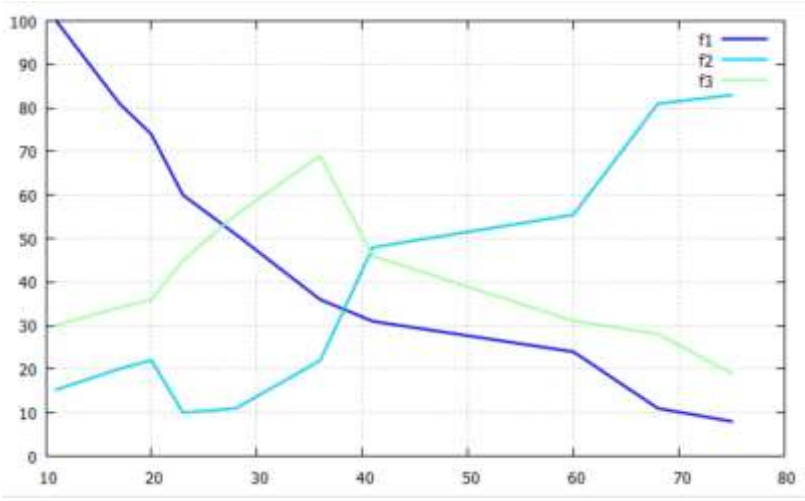
Можно передавать от 1 до 5 наборов данных n_i, x_i, y_i разных размерностей.

Примеры:

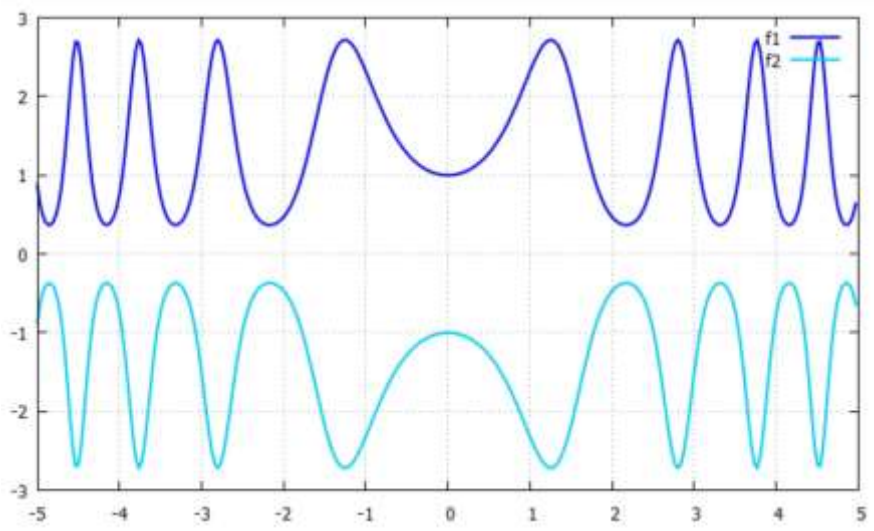
```
int x[10] = {11,17,20,23,28,36,41,60,68,75};
double y[10] = {15.3, 20.1, 22.1, 10, 11, 22, 48, 55.5, 81, 83};
GnuP p;
p.plotArray(10,x,y);
p.plot();
```




```
int x[10] = {11,17,20,23,28,36,41,60,68,75};
int y1[10] = {100,81,74,60,51,36,31,24,11,8};
double y2[10] = {15.3, 20.1, 22.1, 10, 11, 22, 48, 55.5, 81, 83};
float y3[10] = {30, 34.2, 36, 45, 55.5, 69, 46, 31.1, 28.1, 19};
GnuP p;
p.plotArray(10,x,y1,y2,y3);
p.plot();
```




```
double x[300], y1[300], y2[300];
GnuP p;
for (int i=0; i<300; i++){
        x[i] = (i-150)/30.0;
        y1[i] = exp(sin(x[i]*x[i]));
        y2[i] = - exp(sin(x[i]*x[i]));}
p.plotArray(300,x,y1,y2);
p.plot();
```



4. plotFunc

Вариантов использования два. Один из них – аналог plotArray, только график строится по массиву узлов x и пользовательской функции, которая обязательно принимает *один* параметр - точку x, и возвращает *одно числовое* значение.

Возможные аргументы функции:

- ✓ n, n1, n2, ... – размерности массивов
- ✓ x, x1, x2, ... – массивы с координатами точек по оси абсцисс
- ✓ f, f1, f2, ... – указатели на пользовательские функции
- ✓ type0, type1, ... – любые числовые типы данных

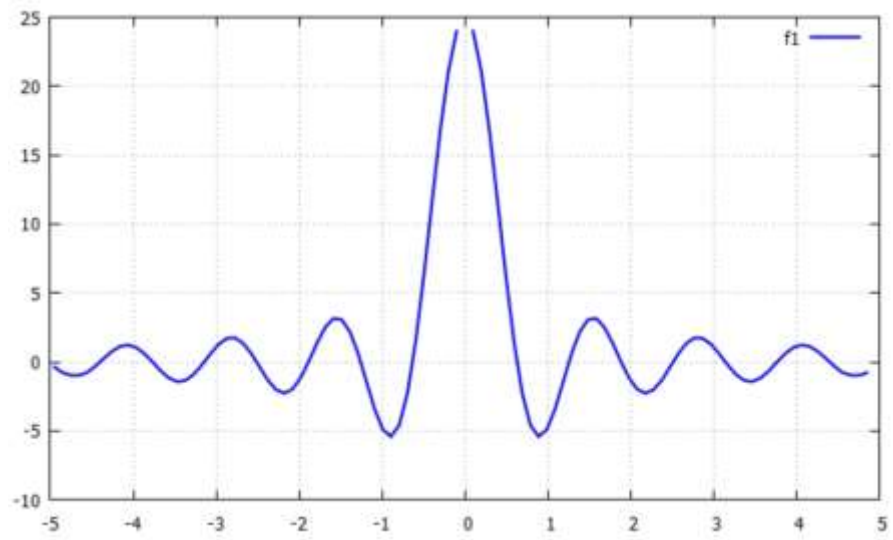
1) `plotFunc (int n, type0 x, type1 f)`

2) `plotFunc (int n, type0 x, type1 f1, … , type5 f5)`

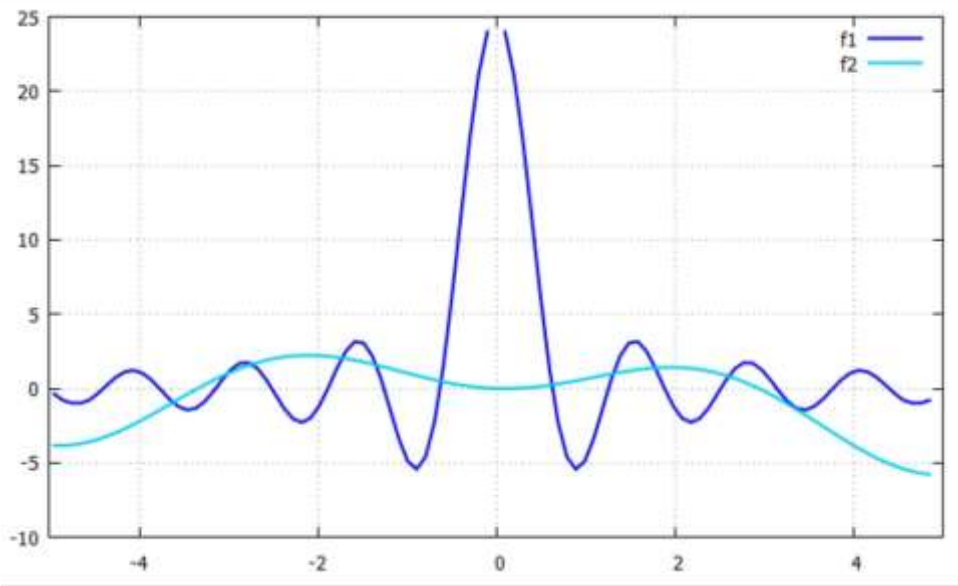
3) `plotFunc (int n1, type11 x1, type12 f1, … , int n5, type51 x5, type52 f5)`

Примеры:

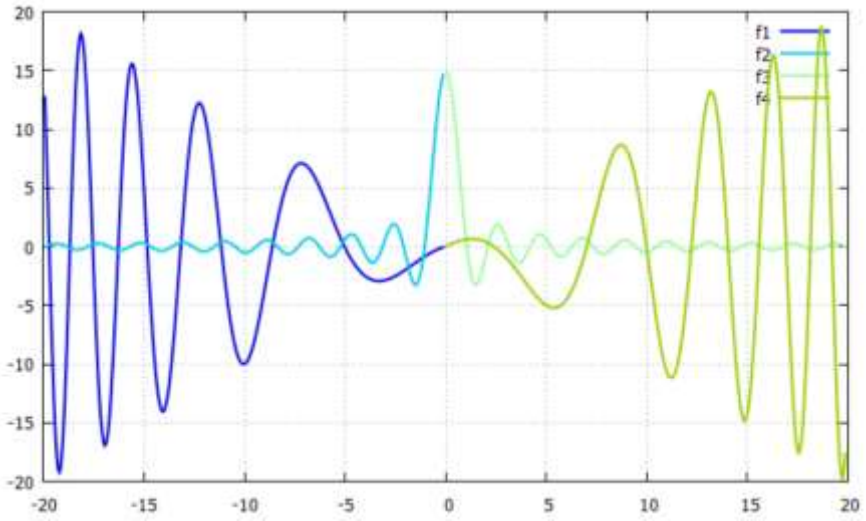
```
double f (double x) {return 5*sin(x*5)/x;}
int main()
{
     double x[100];
     for (int i=0; i<100; i++ )
        x[i] = (i-50)/10.10;
     GnuP p;
     p.plotFunc(100,x,f);
     p.plot();
     return 0; }
```




```
double f1 (double x) {return 5*sin(x*5)/x;}
double f2(double x) { return sin(x)*x - x/5;}
int main()
{
      double x[100];
      for (int i=0; i<100; i++ )
            x[i] = (i-50)/10.10;
      GnuP p;
      p.plotFunc(100,x,f1,f2);
      p.plot();
      return 0;
}
```



```
double f1(double x) { return x*sin(exp(sqrt(-x))/3);}
double f2(double x) { return 5*sin(x*3)/x;}
double f3(double x) { return x*cos(exp(sqrt(x))/3);}
int main()
{    double x1[300], x2[300];
     for (int i=0; i<300; i++){
          x1[i] = (i-150)/15.10 - 10;
          x2[i] = (i-150)/15.10 + 10;}
     GnuP p;
     p.plotFunc(300,x1,f1,300,x1,f2,300,x2,f2,300,x2,f3);
     p.plot();
     return 0; }
```

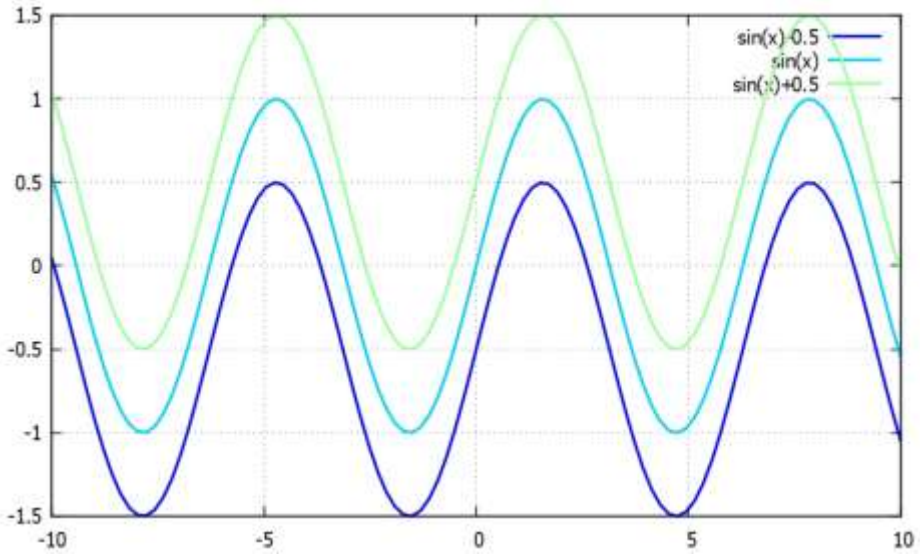


4) `plotFunc (string f1, string f2, … ,string f5)`

В качестве аргументов передаются от 1 до 5 функций, записанных в виде строк. При этом записи функций должны соответствовать синтаксису Gnuplot.

Пример:

```
GnuP p;
p.plotFunc("sin(x)-0.5", "sin(x)","sin(x)+0.5");
p.plot();
```

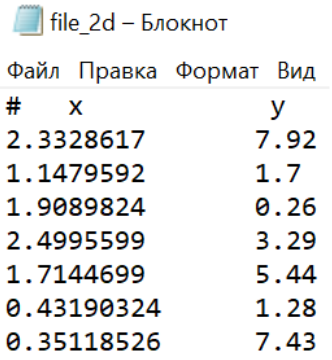


5. plotFile

График строится по данным из файлов.

```
plotFile (string file1, string file2, … , string file5)
```

Файл должен представлять собой набор координат точек графика. В каждой строке два значения через пробел или знак табуляции: координата точки по оси Ox и координата по оси Oy. Пример заполнения на рисунке справа. Если строка начинается со значка "#", то она игнорируется.



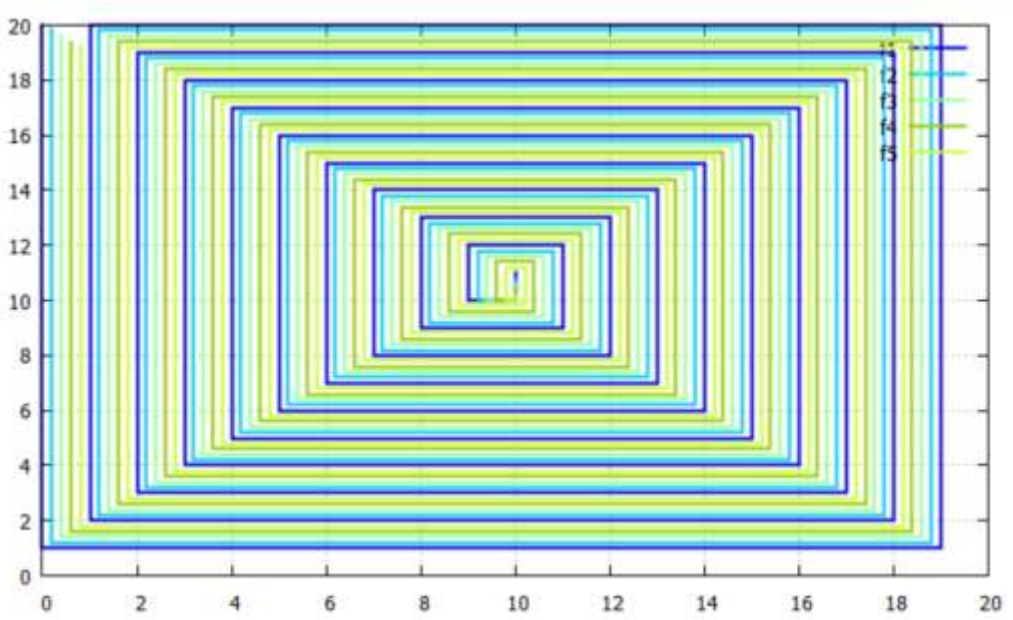
Gnuplot может читать файлы любого формата, для этого необходимо указывать формат файла. Например, *.txt, .dat* и др. За более подробной информацией можно обратиться к документации Gnuplot.

По умолчанию файл для построения будет браться из директории или папки, в которой запускается программа. Можно указать полное имя, например:

```
C:/Users/My/Documents/file_2d.txt
```

Пример:

```
GnuP p;
p.plotFile("f1.txt", "f2.txt", "f3.txt", "f4.txt", "f5.txt");
p.plot();
```



6. plotArrayPar

Метод позволяет построить *один* график и визуально его настроить.

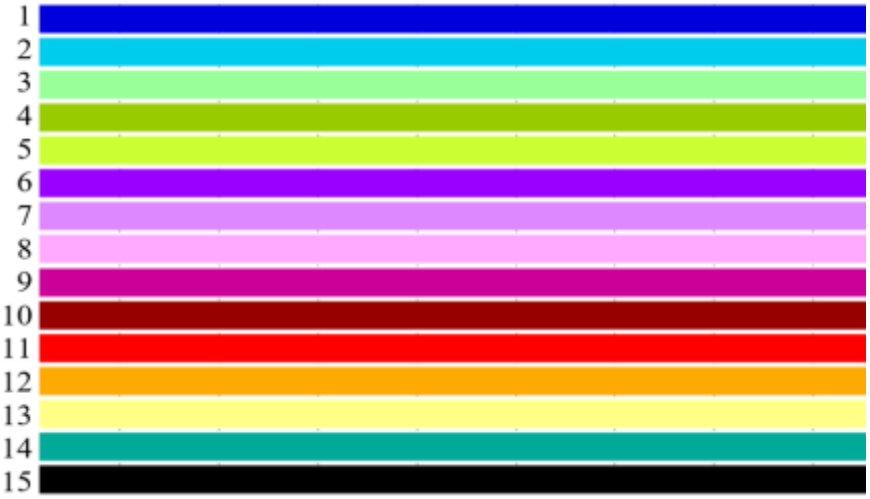
```
plotArrayPar (int n, type0 x, type1 y, int line, int width, int color, string legend)
```

- ✓ n – размерность массивов
- ✓ x, y – массивы с данными
- ✓ line – тип линии
- ✓ width – толщина линии
- ✓ color – цвет линии
- ✓ legend – подпись графика функции

Доступные типы линий

- ✓ 1 – точки
- ✓ 2 – линия (по умолч.)
- ✓ 3 – линия с точкой
- ✓ 0 – поставить по умолчанию

Доступные цвета:



Толщина линии и подпись графика могут быть любыми.

При вызове метода можно опустить параметры для визуальной настройки, тогда они будут установлены по умолчанию:

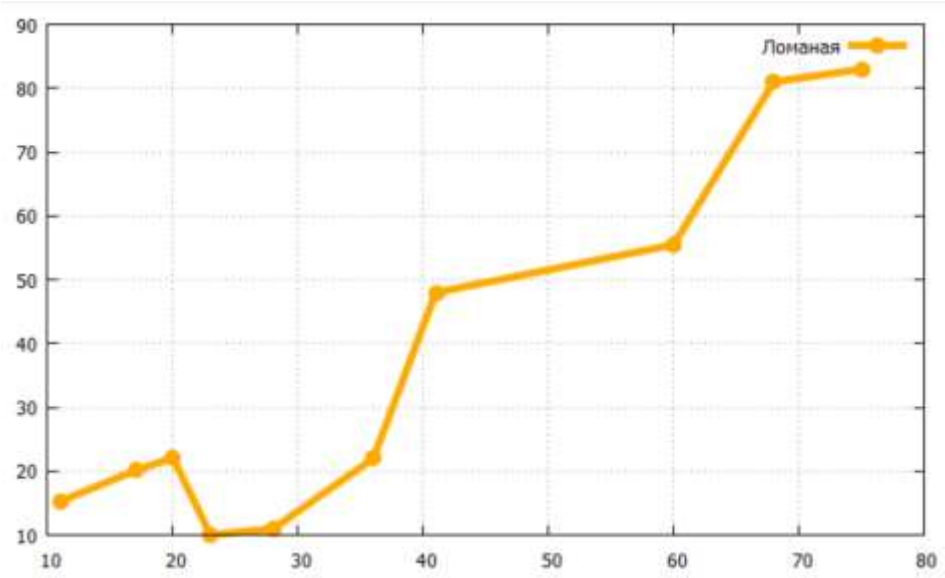
```
plotArrayPar (n, x, y)
```

Если нужно настроить не все параметры, а, например, только цвет линии, то все неинтересующие числовые параметры передаются как нули (line и width), а все параметры, которые идут после интересующего, можно опустить (legend). Они будут так же установлены по умолчанию:

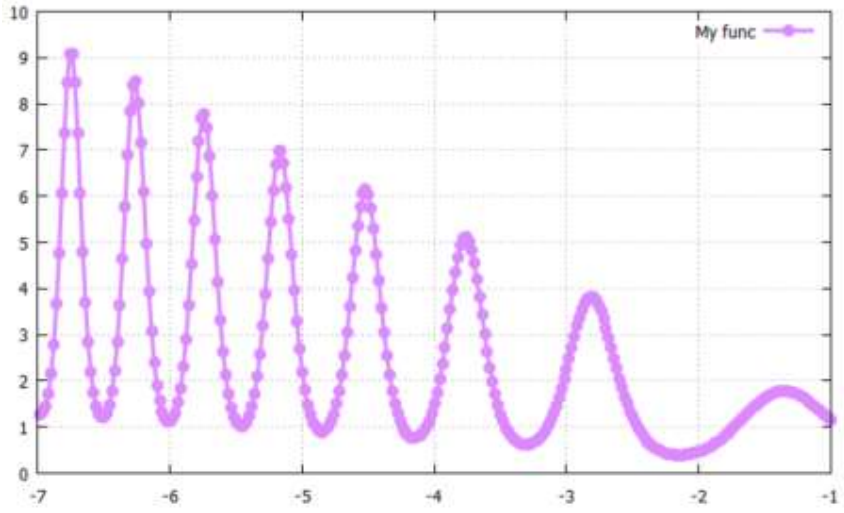
```
plotArrayPar (n, x, y, 0, 0, 9)
```

Пример (тот же, что в п. plotArray, но с визуальной настройкой):

```
int x[10] = {11,17,20,23,28,36,41,60,68,75};
double y[10] = {15.3, 20.1, 22.1, 10, 11, 22, 48, 55.5, 81, 83};
GnuP p;
p.plotArrayPar(10,x,y,3,5,12,"Ломаная");
p.plot();
```




```
double x[300], y[300];
GnuP p;
for (int i=0; i<300; i++){
        x[i] = -(50+i)/50.0;
        y[i] = -x[i]/2*exp(sin(x[i]*x[i])); }
p.plotArrayPar(300,x,y,3,3,7,"My func");
p.plot();
```



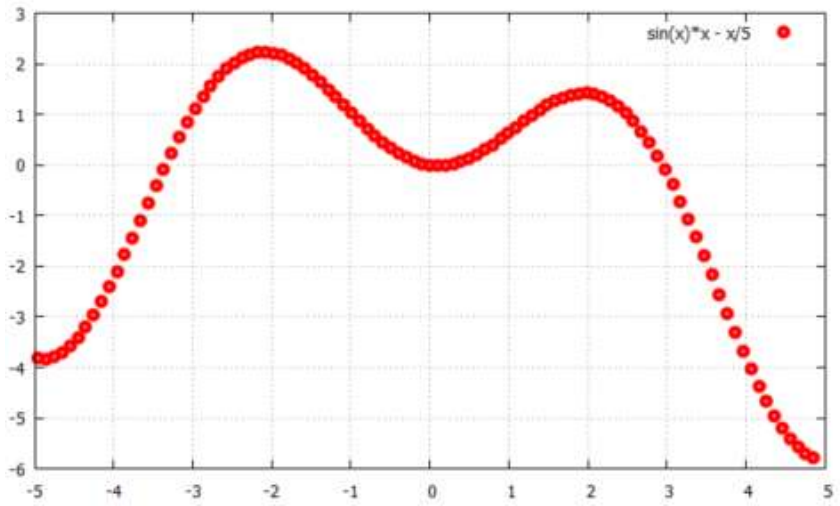
7. plotFuncPar

Аналогично plotArrayPar, только вместо массива значений y передается указатель на пользовательскую функцию f.

```
plotFuncPar (int n, type0 x, type1 f, int line, int width, int color, string legend)
```

Пример:

```
double f(double x) { return sin(x)*x - x/5;}
int main(){
     double x[100];
     for (int i=0; i<100; i++ )
        x[i] = (i-50)/10.10;
     GnuP p;
     p.plotFuncPar(100,x,f,1,4,11,"sin(x)*x - x/5");
     p.plot();
     return 0; }
```

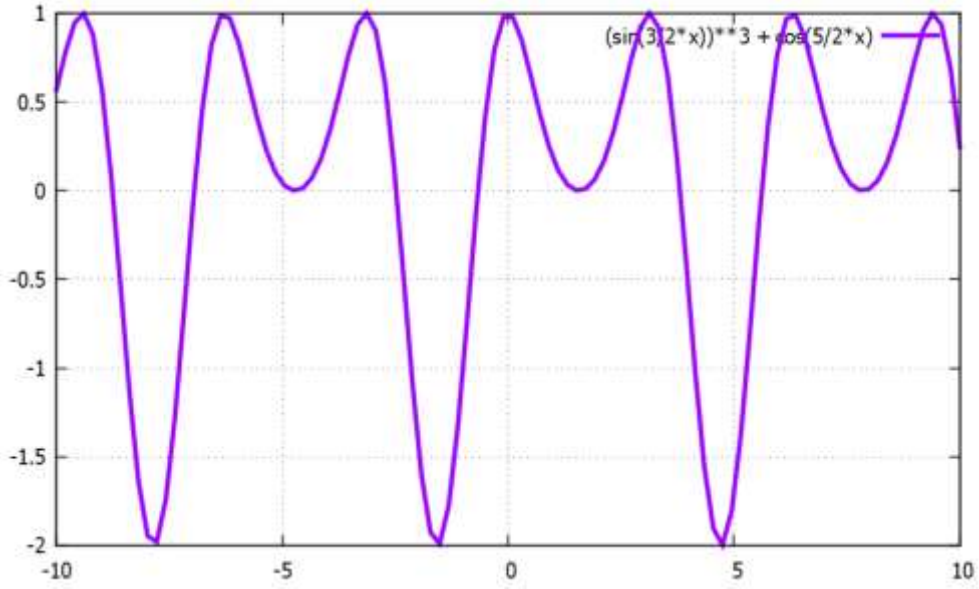


Есть второй вариант использования plotFuncPar, когда в качестве данных передается только функция в виде строки и параметры для настройки, т.е.:

```
plotFuncPar (string f, int line, int width, int color, string legend)
```

Пример:

```
GnuP p;
p.plotFuncPar("(sin(3/2*x))**3 + cos(5/2*x)",2,3,6);
p.plot();
```



Обратите внимание, что запись функции должна соответствовать синтаксису Gnuplot. Например, степень обозначается как "**".

8. plotFuncArg

График строится по массиву узлов x и одной пользовательской функции, которая помимо аргумента x принимает ещё *от 1 до 5* дополнительных аргументов, возвращает *одно* числовое значение.

- ✓ n – размерность массива
- ✓ x – массив узлов
- ✓ f – указатель на пользовательскую функцию с доп.аргументами
- ✓ a, b, c, d, e – дополнительные аргументы функции f, могут быть любого типа

При вызове метода к названию *нужно дописать число от 1 до 5*, обозначающее количество доп.аргументов.

```
1) plotFuncArg1(int n, type0 x, type1 f, a)
2) plotFuncArg2(int n, type0 x, type1 f, a, b)
3) plotFuncArg3(int n, type0 x, type1 f, a, b, c)
4) plotFuncArg4(int n, type0 x, type1 f, a, b, c, d)
5) plotFuncArg5(int n, type0 x, type1 f, a, b, c, d, e)
```

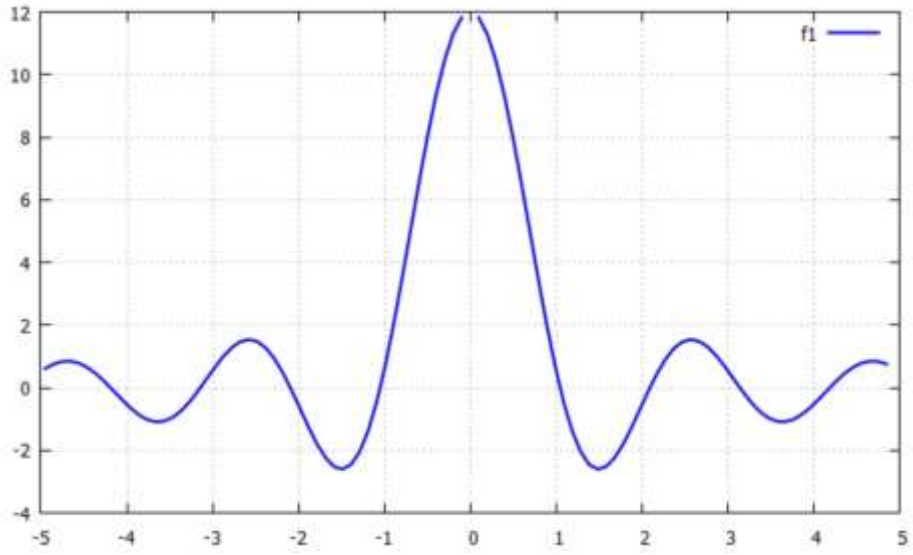
Пользовательская функция при этом должна обязательно первым аргументом принимать одно значение из массива x, а затем все имеющиеся доп.аргументы по порядку:

```
f(x,a,b,c,d,e)
```

Можно настраивать визуально, передавая параметры для настройки после аргументов пользовательской функции. Параметры и способ работы с ними такой же, как в методах plotArrayPar и plotFuncPar.

Пример:

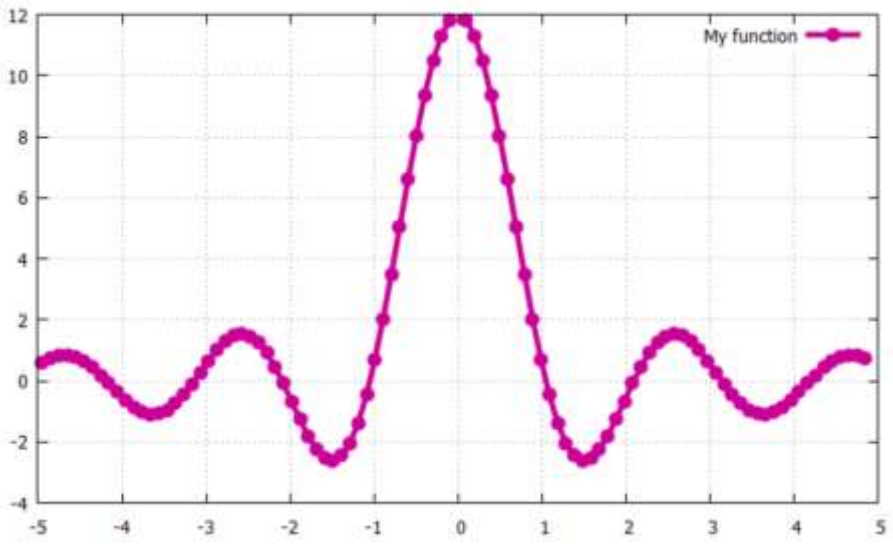
```
double f(double x, int a, string b){
     return b.size()*sin(x*a)/x; }
int main()
{
double x[100];
for (int i=0; i<100; i++ ){
     x[i] = (i-50)/10.10;
string b = "GnuP";
GnuP p;
p.plotFuncArg2(100,x,f,3,b);
p.plot();
return 0;
}
```



Если поменять строку `p.plotFuncArg2(100,x,f,3,b);` на строку

```
p.plotFuncArg2(100,x,f,3,b,3,4,9,"My_function");
```

то:



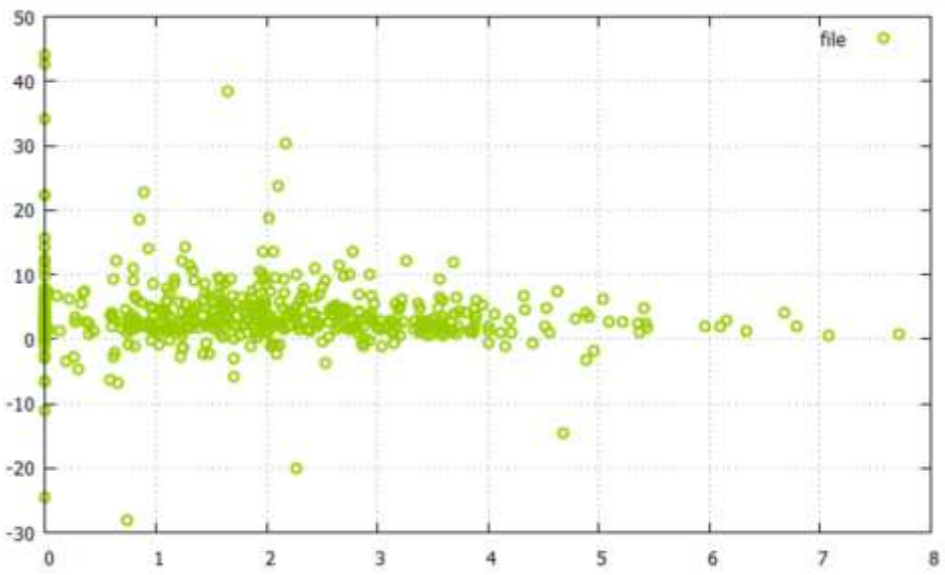
9. plotFilePar

Аналогично plotArrayPar, только вместо массива значений y передается название файла с данными и параметры для настройки. Про тип и структуру файла см. п. 5 – plotFile.

```
plotFilePar (string file, int line, int width, int color, string legend)
```

Пример:

```
GnuP p;
p.plotFilePar("C:/Users/My/Documents/file_2d.txt",1,2,4,"file");
p.plot();
```



10. plotArrayPar_3d

Метод, аналогичный plotArrayPar, но строит *трехмерный* график по точкам с тремя координатами (x, y, z).

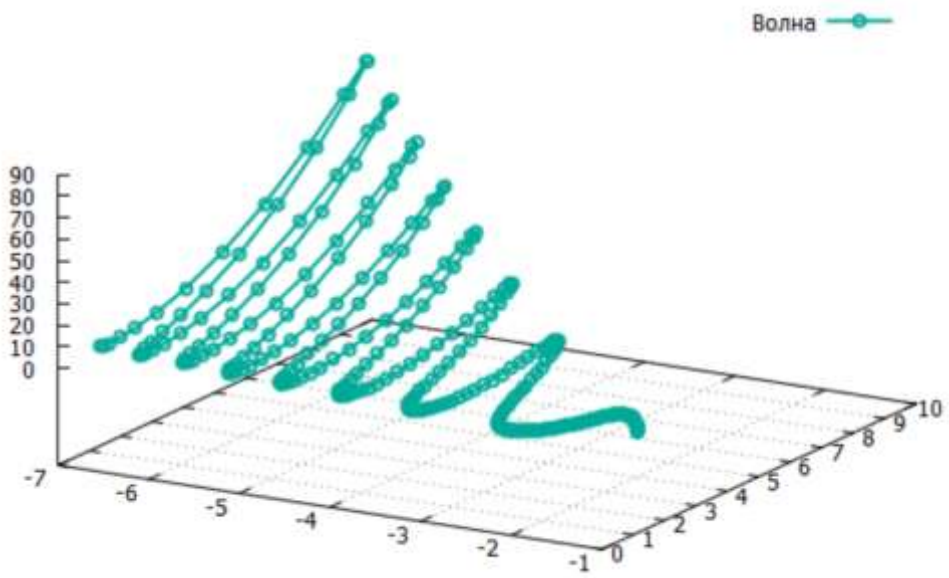
```
plotArrayPar_3d (int n, type0 x, type1 y, type2 z, int
line, int width, int color, string legend)
```

- ✓ x, y, z – массивы с данными

Остальные параметры как у plotArrayPar.

Пример:

```
double x[300], y[300], z[300];
for (int i=0; i<300; i++ ){
        x[i] = -(50+i)/50.0;
        y[i] = -x[i]/2*exp(sin(x[i]*x[i]));
        z[i] = y[i]*y[i];
    }
GnuP p;
p.plotArrayPar_3d(300,x,y,z,3,2,14);
p.plot_3d();
```



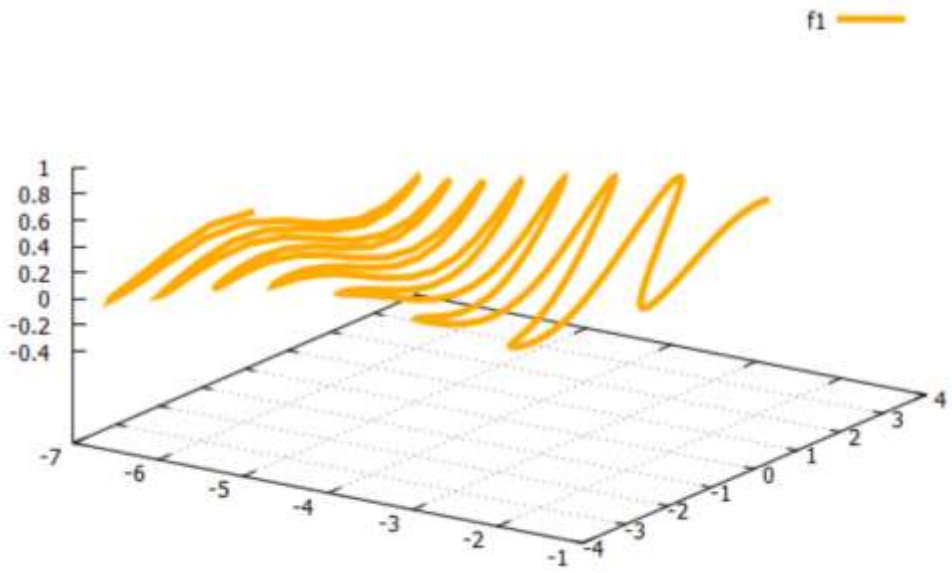
11. plotFuncPar_3d

Аналог plotFuncPar, но в данном случае к параметрам добавляется ещё один массив значений – y; пользовательская функция g должна зависеть от двух аргументов g(x, y).

```
plotFuncPar_3d (int n, type0 x, type1 y, type2 g, int line, int width, int color, string legend)
```

Пример:

```
double g_3d(double x, double y) {return sin(x+y)/(x+y);}
int main(){
     double x[300], y[300];
     for (int i=0; i<300; i++ ){
        x[i] = -(50+i)/50.0;
        y[i] = -(x[i])/2*(cos(x[i]*x[i]));
     }
     GnuP p;
     p.plotFuncPar_3d(300,x,y,g_3d,2,4,12);
     p.plot_3d();
     return 0;
}
```



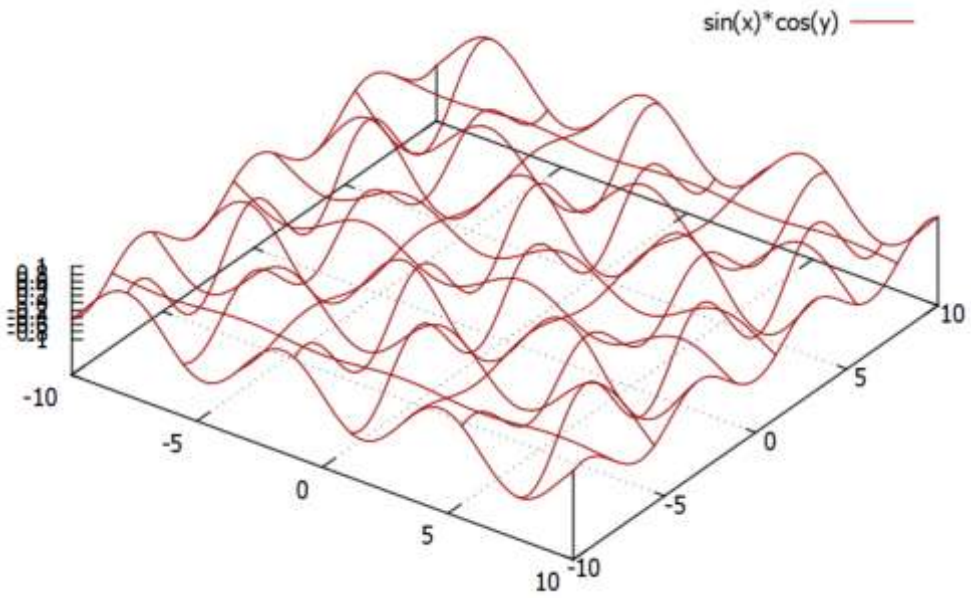
И можно передавать в качестве данных только строку, в которой записано уравнение поверхности для построения, и параметры:

```
plotFuncPar_3d (string g, int line, int width, int color, string legend)
```

Пример:

```
GnuP p;
```

```
p.plotFuncPar_3d("sin(x)*cos(y)",2,0,10);
p.plot_3d();
```



12. plotFilePar_3d

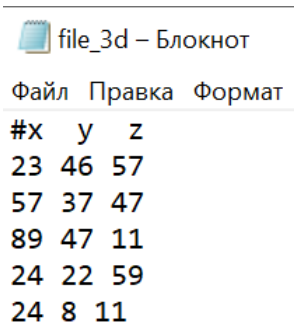
Метод, аналогичный plotFilePar, но строит *трехмерный* график по точкам с тремя координатами (x, y, z) по данным из файла.

```
plotFilePar_3d (string file, int line, int width, int color, string legend)
```

- ✓ file – название файла с данными

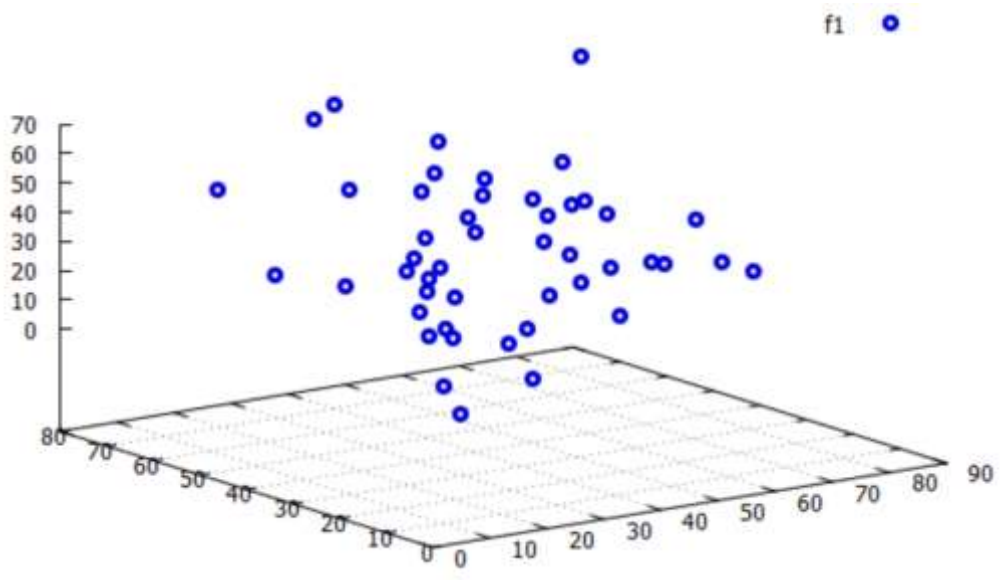
Остальные параметры как у plotFilePar.

Файл с данными должен представлять собой набор координат точек графика. В каждой строке три значения через пробел или знак табуляции: координата точки по оси Ox, по оси Oy и по оси Oz. Подробнее про тип и структуру файла см. п. 5 – plotFile.



Пример:

```
GnuP p;
p.plotFilePar_3d("file_3d.txt", 1, 3);
p.plot_3d();
```



13. setRange

При необходимости можно задать область для построения графиков. В этом поможет setRange.

```
setRange(type1 x1, type2 x2, type3 y1, type4 y2, type5 z1, type6 z2)
```

В качестве аргументов выступают целые или действительные числа, задающие границы $[x_1, x_2]$, $[y_1, y_2]$, $[z_1, z_2]$ по x, y и z соответственно.

Метод применим в случаях 2d и 3d. Можно определить, например, границы для 2d так, чтобы $x \in [1, 5]$, $y \in [-3, 10.5]$ (параметры для оси z просто опускаются):

```
setRange(1, 5, -3, 10.5)
```

Если хочется настроить только одну ось, допустим, чтобы $z \in [-7, 7]$, то границы для других осей, предшествующие задаваемым границам, передаются в качестве 0:

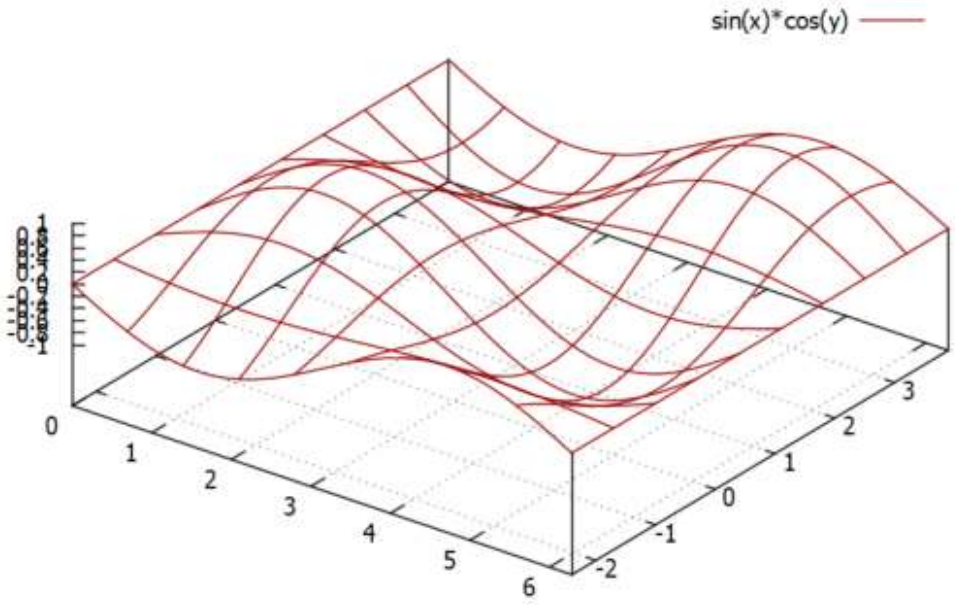
```
setRange(0, 0, 0, 0, -7, 7)
```

Важно: если данные для графика определяются массивами данных или пользовательской функцией, то обязательно должны быть точки, принадлежащие области, получаемой при масштабировании осей.

Пример (дополним пример из п. plotFuncPar_3d) :

```
GnuP p;
p.setRange(0,6.28, -2.4, 3.9);
p.plotFuncPar_3d("sin(x)*cos(y)",2,0,10);
```

```
p.plot_3d();
```



Передавать можно только *четное* количество аргументов (2, 4 или 6). При этом, в каждой паре первый должен быть меньше второго.

Чтобы сбросить установленные параметры, нужно вызвать метод с двумя 0:

```
p.setRange(0,0);
```

14. setParam

Метод для настройки общих параметров. Подходит как для 2d, так и для 3d.

```
setParam(int grid, int shape, int loc_legend_1, int loc_legend_2, string title)
```

- ✓ grid – сетка
- ✓ shape – форма окна вывода
- ✓ loc_legend_1 – расположение легенды
- ✓ loc_legend_2 – расположение легенды
- ✓ title – общий заголовок

Сетка:

- ✓ 1 – вкл. (по умолч.)
- ✓ 2 – выкл.
- ✓ 0 – поставить по умолчанию

Форма окна вывода:

- ✓ 1 – квадратное (только 2d)
- ✓ 2 – прямоугольное (по умолч.)
- ✓ 0 – поставить по умолчанию

Расположение легенды (loc_legend_1):

- ✓ 1 – слева сверху
- ✓ 2 – справа сверху (по умолч.)
- ✓ 3 – слева снизу (только 2d)
- ✓ 4 – справа снизу (только 2d)
- ✓ 0 – поставить по умолчанию

Расположение легенды (loc_legend_2):

- ✓ 1 – внутри окна с графиками (по умолч.)
- ✓ 2 – снаружи окна с графиками
- ✓ 0 – поставить по умолчанию

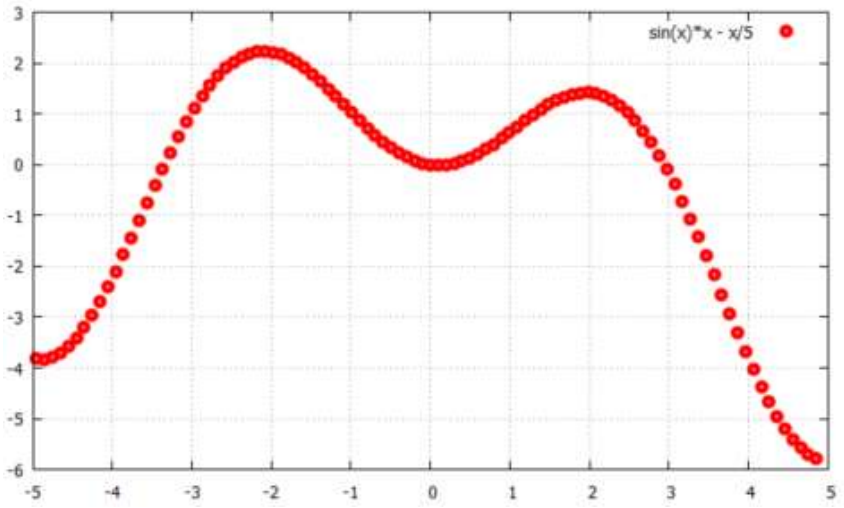
Общий заголовок может быть любой.

Так же, если после всех интересующих параметров имеются ещё какие-либо, их можно опустить. Например, настройка только формы окна:

```
setParam(0, 1)
```

Пример:

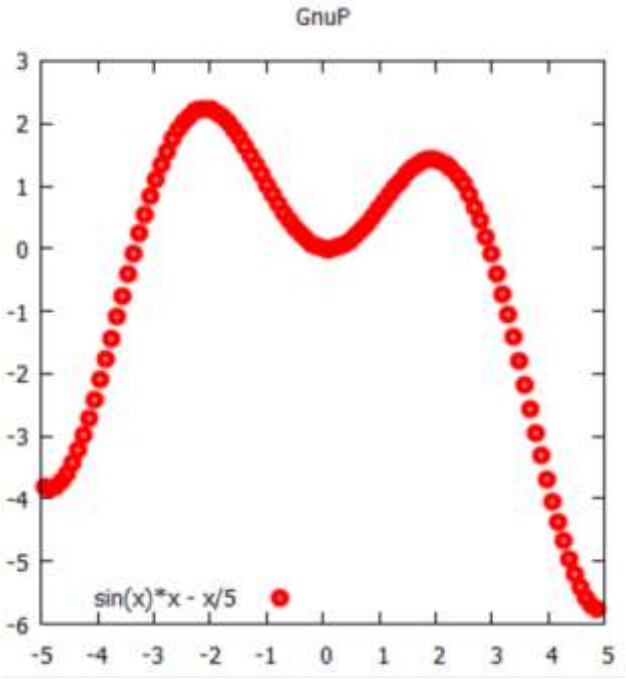
С параметрами по умолчанию (см. код в п. plotFuncPar)



С настроенными параметрами

```
double f(double x) { return sin(x)*x - x/5;}
int main() {
     double x[100];
     for (int i=0; i<100; i++ )
          x[i] = (i-50)/10.10;
     GnuP p;
     p.plotFuncPar(100,x,f,1,4,11,"sin(x)*x - x/5");
     p.setParam(2,1,3,1,"GnuP");
```

```
p.plot();
return 0; }
```



Чтобы сбросить установленные параметры, нужно вызвать метод без параметров:

```
p.setParam();
```

15. setParam_3d

Позволяет определить визуальные настройки 3d графиков.

```
setParam_3d (int hidden, int pm3d, int iso_1, int iso_2)
```

- ✓ hidden – прозрачность поверхности
- ✓ pm3d – цветовая палитра поверхности
- ✓ iso_1 – частота разбиения (по x и y, либо только по x)
- ✓ iso_2 – частота разбиения (по y)

Прозрачность поверхности:

- ✓ 1 – вкл. (по умолч.)
- ✓ 2 – выкл.
- ✓ 0 – поставить по умолчанию

Цветовая палитра поверхности:

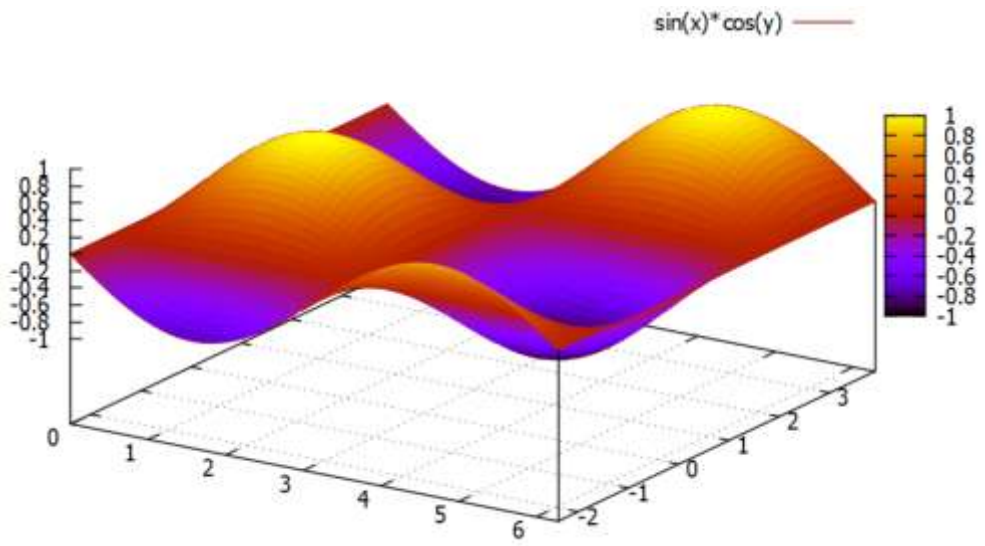
- ✓ 1 – вкл.
- ✓ 2 – выкл. (по умолч.)
- ✓ 0 – поставить по умолчанию

Если график будет представлять собой одну линию, то применить цветовую палитру не получится. Поэтому данный параметр лучше использовать в тандеме с plotFuncPar_3d, задавая функцию строкой.

Частота разбиения – количество точек на оси, в которых будет вычисляться значение функции. Может задаваться одним целым числом (параметр iso_1). В этом случае оно будет определять сразу и разбиение по x, и по y. Если же ввести оба параметра, то iso_1 будет определять разбиение по x, а iso_2 – по y.

Пример (дополним пример из п. setRange):

```
GnuP p;
p.setRange(0,6.28, -2.4, 3.9);
p.setParam_3d(1,1,60,60);
p.plotFuncPar_3d("sin(x)*cos(y)",2,0,10);
p.plot_3d();
```



Чтобы сбросить установленные параметры, нужно вызвать метод без параметров:

```
p.setParam_3d();
```

16. clearData

Очищает данные о 2d графиках, которые были сохранены до вызова clearData.

```
p.clearData();
```

17. clearData_3d

Очищает данные о 3d графиках, которые были сохранены до вызова clearData_3d.

```
p.clearData_3d();
```

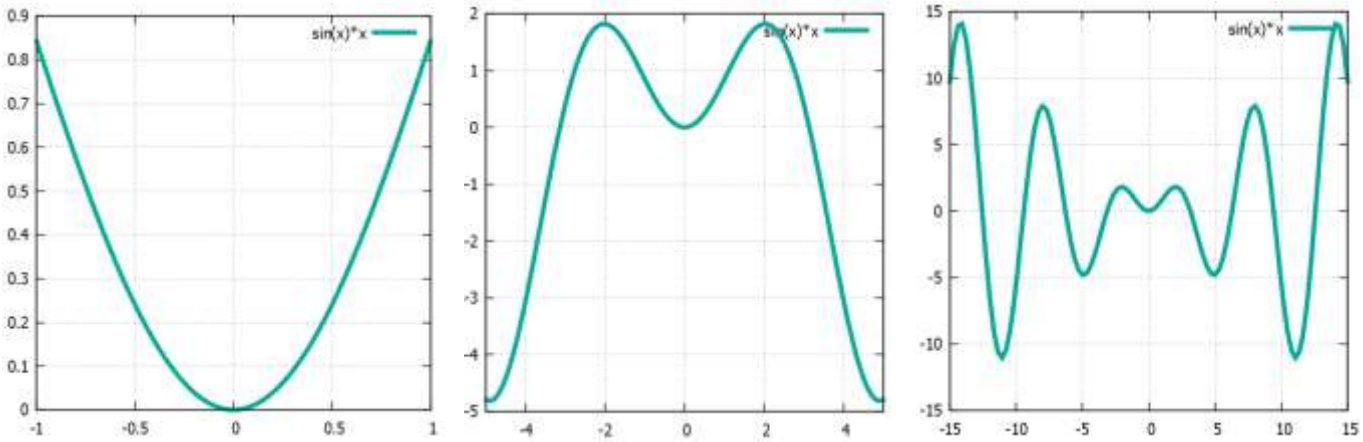
18. Повторный вызов plot и plot_3d

В одной программе можно несколько раз вызвать методы plot и plot_3d.

Допустим, хотим посмотреть, как выглядит график функции x·sin(x), но изначально не знаем, на каком отрезке лучше строить. Поэтому задаем цикл из 3 итераций, в котором будем вводить и устанавливать значения a и b: x∈[a, b], а потом вызывать plot, чтобы увидеть график.

```
GnuP p;
p.plotFuncPar("sin(x)*x",2,4,14);
p.setParam(0,1);
double a, b;
for(int i=0;i<3;i++){
     cout<<"a = "; cin>>a;
     cout<<"b = "; cin>>b;
     p.setRange(a,b);
     p.plot();
}
```

На первой итерации введем a = -1 и b = 1, на второй – a = -5 и b = 5, на третьей – a = -15 и b = 15. После каждого ввода на экране будет появляться квадратное окно с графиком, где x∈[a, b]. Переход к следующей итерации будет возможен после закрытия окна с графиком.

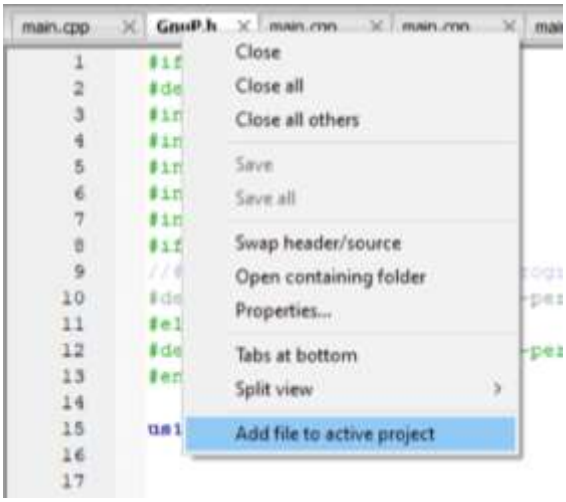


Использование в fortran

Для начала работы к своему проекту нужно подключить файл GnuP.95 (он должен находиться в одной папке с вашим файлом основной программы):

```
use GnuP
```

Если используете Code::Blocks, подключить GnuP.f95 можно, нажав правой кнопкой мыши на вкладку «GnuP.f95» и выбрав «Add file to active project»:



GnuP готов к работе.

Прежде всего нужно создать объект класса GnuP_f:

```
type(GnuP_f) :: p
```

Перед началом **обязательно** нужно вызвать метод GnuPP для инициализации:

```
call GnuPP(p)
```

Теперь можно обращаться к методам этого объекта для построения графиков. Все они будут строиться разом и располагаться на одном полотне.

Введем некоторые обозначения, которые понадобятся далее для определения методов.

Таблица 1

	Название	x	y или f(x)
1	INT	`integer`	`integer`
2	REAL	`real`	`real`
3	MIX_1	`integer`	`real`
4	MIX_2	`real`	`integer`

Абсолютно каждый метод первым параметром принимает **объект класса**.

Методы:

1. **plot**

После его вызова на экран будут выведены двумерные графики. Вызывать в самом конце, после всех остальных методов.

2. plot_3d

Как метод plot, но предназначен для отрисовки трехмерных графиков.

3. plotArray

График строится по данным из массивов. Возможные аргументы метода:

- ✓ n, n1, n2, ... – размерности массивов
- ✓ x, x1, x2, ... – массивы с координатами точек по оси абсцисс
- ✓ y, y1, y2, ... – массивы с координатами точек по оси ординат
- ✓ type0, type1, ... – числовые типы данных

1) `plotArray (GnuP_f p, int n, type0 x, type1 y)`

Построение 1 графика.

Доступны все комбинации типов для x и y из таблицы 1.

2) `plotArray (GnuP_f p, int n, type0 x, type1 y1, type1 y2,…)`

Можно передавать от 1 до 5 различных массивов y1, …, y5 одинакового размера со значениями функций.

Доступны все комбинации типов для x и y_i из таблицы 1, y_i должны быть одного типа. Например, x – integer, y1, y2, y3 – real (MIX_1).

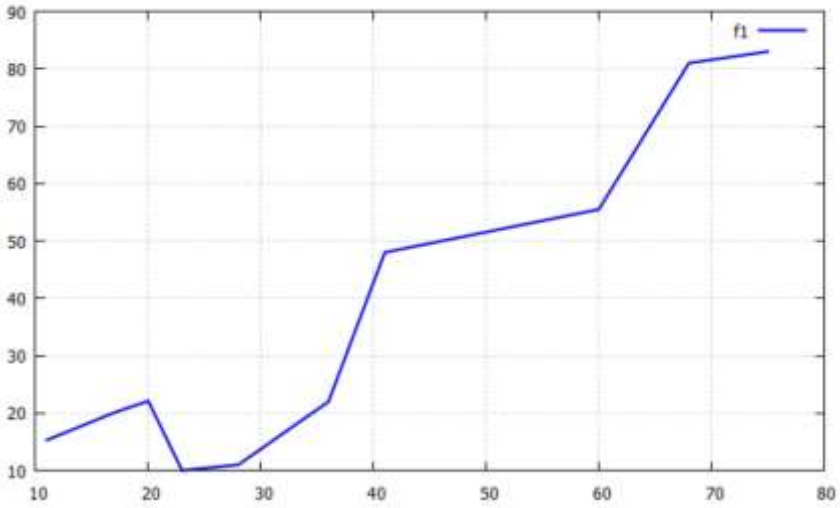
3) `plotArray (GnuP_f p, int n1, type x1, type y1, int n2, type x2, type y2, ...)`

Можно передавать от 1 до 5 наборов данных n_i, x_i, y_i разных размерностей. Все массивы должны быть либо типа integer, либо real.

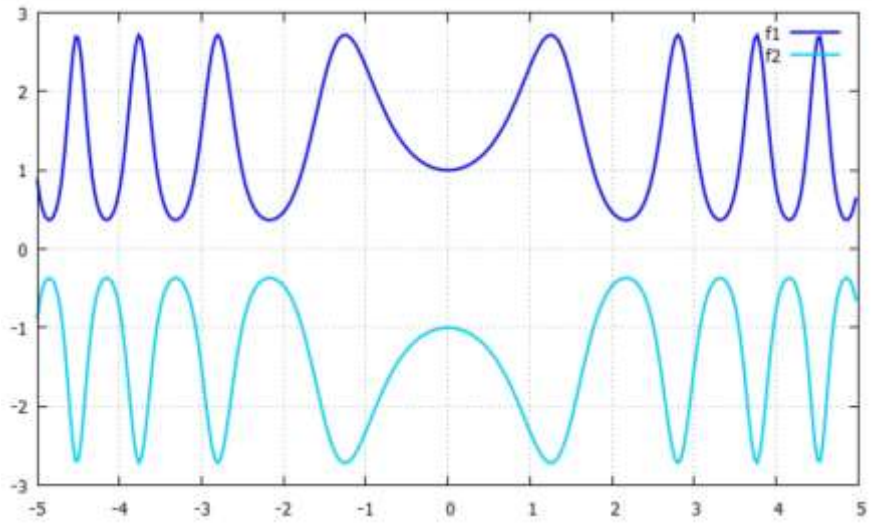
Примеры:

```
integer, dimension(0:9):: x = (/11,17,20,23,28,36,41,60,68,75/)
real, dimension(0:9):: y = (/15.3, 20.1, 22.1, 10.0, 11.0, 22.0, 48.0, 55.5, 81.0, 83.0/)
type(GnuP_f) :: p
call GnuPP(p)
call plotArray(p,10,x,y)
```

```
call plot(p)
```

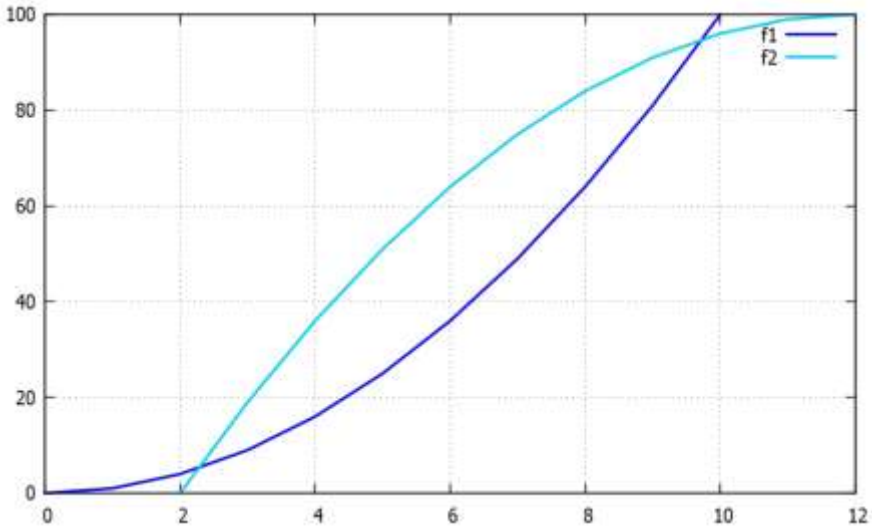



```
real, dimension(0:300) :: x, y1, y2
type(GnuP_f) :: p
do i=0,300
      x(i) = (i-150)/30.0
      y1(i) = exp(sin(x(i)*x(i)))
      y2(i) = - exp(sin(x(i)*x(i)))
end do
call GnuPP(p)
call plotArray(p,301,x,y1,y2)
call plot(p)
```




```
integer, dimension(0:10):: x1, x2, y1, y2
type(GnuP_f) :: p
do i=0,10
```

```
        x1(i) = i
        x2(i) = i+2
        y1(i) = i*i
        y2(i) = i*20 - i*i
    end do
    call GnuPP(p)
    call plotArray(p,11,x1,y2,11,x2,y2)
    call plot(p)
```



4. plotFunc

Аналог plotArray, только график строится по массиву узлов x и пользовательской функции, которая обязательно принимает *один* параметр - точку x, и возвращает *одно числовое* значение.

Возможные аргументы функции:

- ✓ n – размерность массива x
- ✓ x – массив с координатами точек по оси абсцисс
- ✓ m – размерность массива f
- ✓ f – массив указателей на пользовательские функции
- ✓ type0, type1 – числовые типы данных

```
plotFunc (GnuP_f p, int n, type0 x, int m, type1 f)
```

Чтобы можно было передать функции в качестве параметров, нужно создать массив функций, тип которого соответствует одному из специальных типов:

- ✓ func_INT– аргумент функции и возвращаемое значение типа integer
- ✓ func_REAL – аргумент функции и возвращаемое значение типа real
- ✓ func_IR – аргумент функции типа integer, возвращаемое значение – real
- ✓ func_RI – аргумент функции типа real, возвращаемое значение – integer

Сами функции могут быть как внешними, так и внутренними, они должны соответствовать следующему интерфейсу:

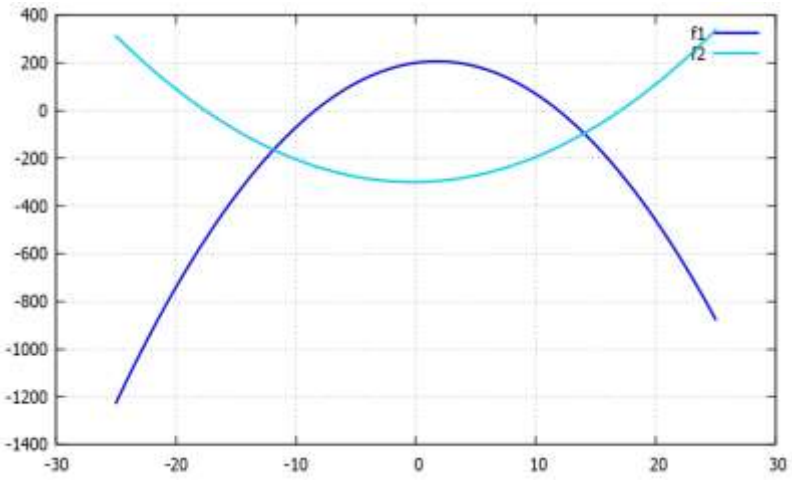
```
type1 function f(x)
      type0 :: x
end function
```

Примеры:

```
program main
     use GnuP
     implicit none
     external f_2 ! внешняя функция
     integer f_2
     real, dimension(0:100):: x
     ! массив указателей на функции
     type(func_RI), dimension(1:2) :: mass
     type(GnuP_f) :: p

     do i = -50,50
         x(i+50) = i*0.5
     end do
     mass(1)%f_pointer => f_1
     mass(2)%f_pointer => f_2
     call GnuPP(p)
     call plotFunc(p,101,x,2,mass)
     call plot(p)
contains
    integer function f_1(x)
        real :: x
        f_1 = -x*x*2 + 7*x + 200
        return
    end function
end
integer function f_2(x)
        real :: x
        f_2 = x*x + 0.5*x - 300
        return
```

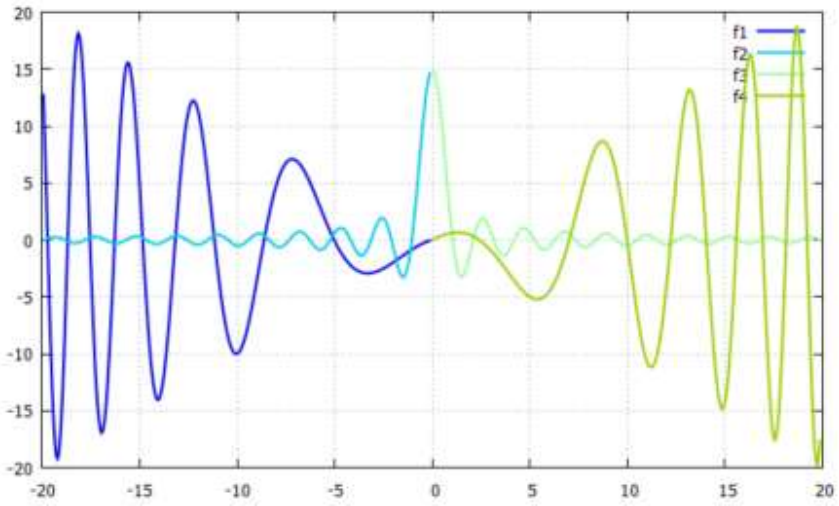
```
end function
```




```
program main
     use GnuP
     implicit none
     real, dimension(0:300) :: x1, x2
     type(func_REAL), dimension(1:2) :: mass_1, mass_2
     type(GnuP_f) :: p
     mass_1(1)%f_pointer => f_1
     mass_1(2)%f_pointer => f_2
     mass_2(1)%f_pointer => f_2
     mass_2(2)%f_pointer => f_3
     do i=0,300
          xxx1(i) = (i-150)/15.10 - 10
          xxx2(i) = (i-150)/15.10 + 10
     end do
     call GnuPP(p)
     call plotFunc(p,301,xxx1,2,mass_1)
     call plotFunc(p,301,xxx2,2,mass_2)
     call plot(p)
contains
     real function f_1(x)
        real :: x
        f_1 = x*sin(exp(sqrt(-x))/3)
        return
    end function
```

```
    real function f_2(x)
        real :: x
        f_2 = 5*sin(x*3)/x
        return
    end function

    real function f_3(x)
        real :: x
        f_3 = x*cos(exp(sqrt(x))/3)
        return
    end function
end
```



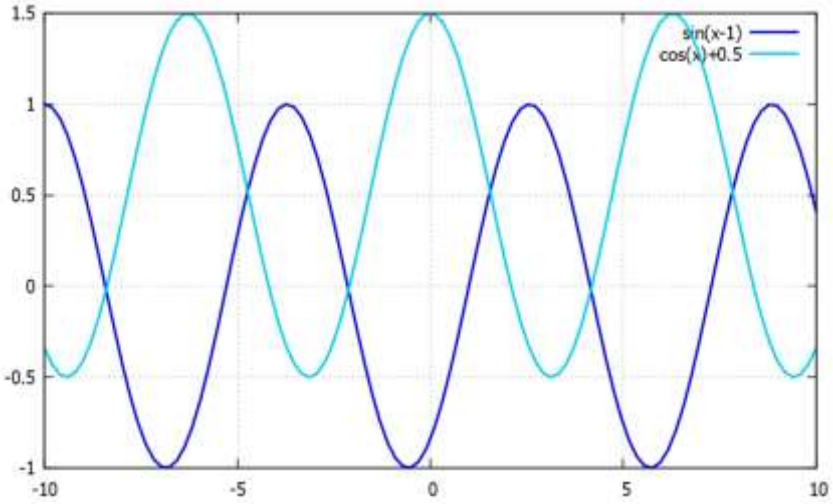
Второй вариант использования:

```
plotFunc (GnuP_f p, string f1, string f2, … , string f5)
```

В качестве аргументов передаются от 1 до 5 функций, записанных в виде строк. При этом записи функций должны соответствовать синтаксису Gnuplot.

Пример:

```
type(GnuP_f) :: p
call GnuPP(p)
call plotFunc(p, "sin(x-1)", "cos(x)+0.5")
call plot(p)
```



5. plotArrayPar

Метод позволяет построить *один* график и визуально его настроить.

```
plotArrayPar (GnuP_f p, int n, type0 x, type1 y, int line, int width, int color, string legend)
```

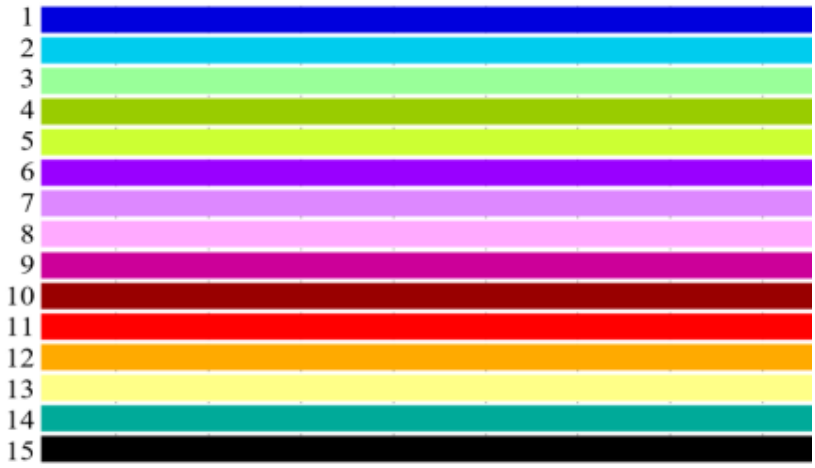
- ✓ n – размерность массивов
- ✓ x, y – массивы с данными
- ✓ line – тип линии
- ✓ width – толщина линии
- ✓ color – цвет линии
- ✓ legend – подпись графика функции

Доступны все комбинации типов для x и y из таблицы 1.

Доступные типы линий

- ✓ 1 – точки
- ✓ 2 – линия (по умолч.)
- ✓ 3 – линия с точкой
- ✓ 0 – поставить по умолчанию

Доступные цвета:



Толщина линии и подпись графика могут быть любыми.

При вызове метода можно опустить параметры для визуальной настройки, тогда они будут установлены по умолчанию:

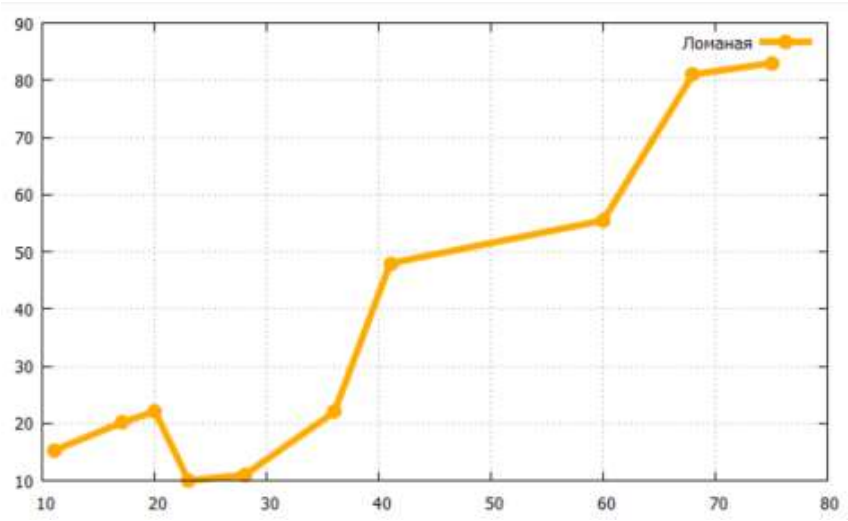
```
plotArrayPar (p, n, x, y)
```

Если нужно настроить не все параметры, а, например, только цвет линии, то все неинтересующие числовые параметры передаются как нули (line и width), а все параметры, которые идут после интересующего, можно опустить (legend). Они будут так же установлены по умолчанию:

```
plotArrayPar (p, n, x, y, 0, 0, 9)
```

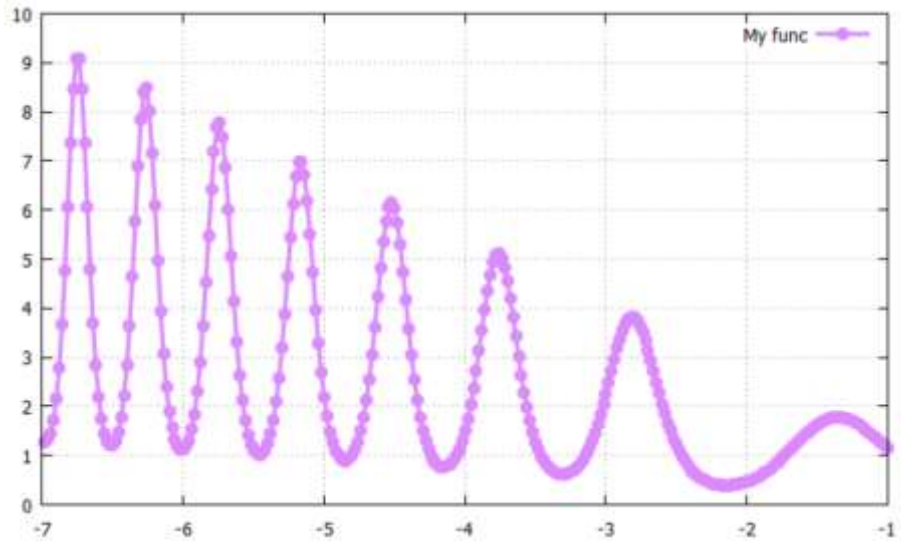
Пример (тот же, что в п. plotArray, но с визуальной настройкой):

```
integer, dimension(0:9):: x = (/11,17,20,23,28,36,41,60,68,75/)
real, dimension(0:9):: y = (/15.3, 20.1, 22.1, 10.0, 11.0, 22.0, 48.0, 55.5, 81.0, 83.0/)
type(GnuP_f) :: p
call GnuPP(p)
call plotArrayPar(p,10,x,y,3,5,12,"Ломаная")
call plot(p)
```




```
real, dimension(0:300) :: x, y
type(GnuP_f) :: p
do i=0,300
     x(i) = -(50+i)/50.0
     y(i) = -x(i)/2*exp(sin(x(i)*x(i)))
end do
```

```
call GnuPP(p)
call plotArrayPar(p,301,x,y,3,3,7,"My func")
call plot(p)
```



6. plotFuncPar

Аналогично plotArrayPar, только вместо массива значений y передается пользовательская функция f, которая принимает одно значение x и возвращает значение f(x).

```
plotFuncPar (GnuP_f p, int n, type0 x, type1 f, int line, int width, int color, string legend)
```

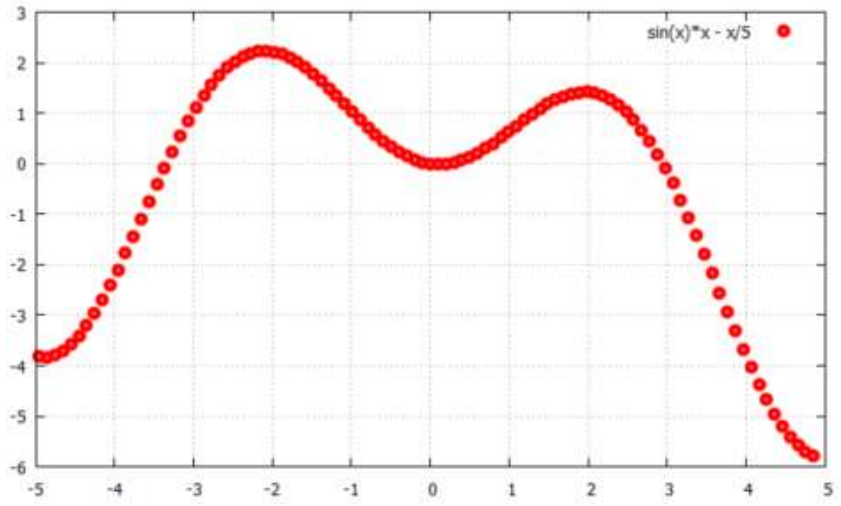
Функция f обязательно должна быть внешней (или модульной) и должна соответствовать следующему интерфейсу:

```
type1 function f(x)
     type0 :: x
end function
```

Пример:

```
program main
     use GnuP
     implicit none
     external f_R
     real f_R
     real, dimension(0:100):: x
     type(GnuP_f) :: p
     do i=0,100
        x(i) = (i-50)/10.10
```

```
    end do
    call GnuPP(p)
    call plotFuncPar(p,100,x,f,1,4,11,"sin(x)*x - x/5")
    call plot(p)
end
real function f(x)
    real :: x
    f = sin(x)*x - x/5
    return
end function
```

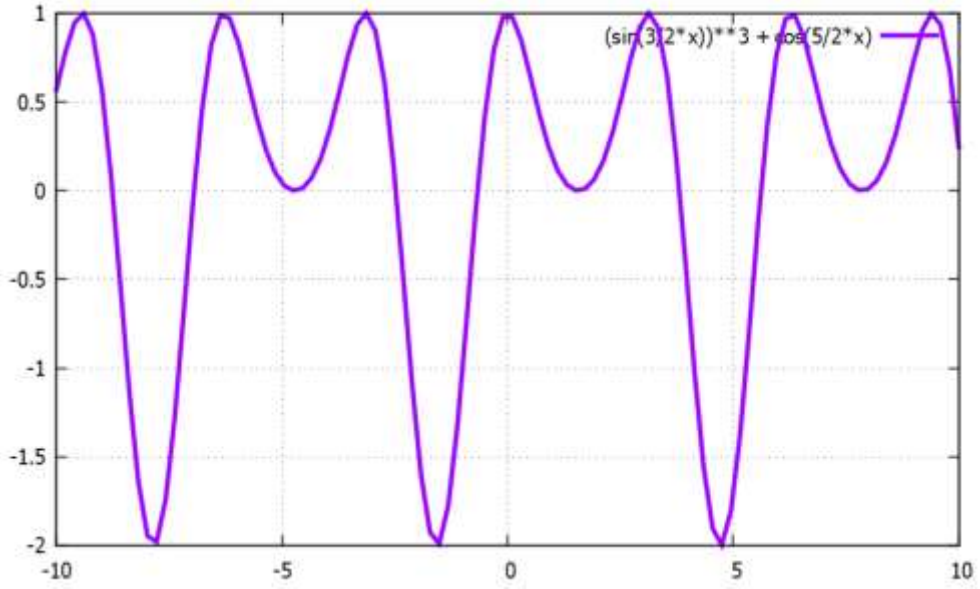


Есть второй вариант использования plotFuncPar, когда в качестве данных передается только функция в виде строки и параметры для настройки, т.е.:

```
plotFuncPar (GnuP_f p, string f, int line, int width, int color, string legend)
```

Пример:

```
type(GnuP_f) :: p
call GnuPP(p)
call plotFuncPar(p, "(sin(3/2*x))**3 + cos(5/2*x)",2,3,6)
call plot(p)
```



Обратите внимание, что запись функции должна соответствовать синтаксису Gnuplot. Например, степень обозначается как "**".

7. plotArrayPar_3d

Метод, аналогичный plotArrayPar, но строит *трехмерный* график по точкам с тремя координатами (x, y, z).

```
plotArrayPar_3d (GnuP_f p, int n, type0 x, type0 y, type1 z, int line, int width, int color, string legend)
```

- ✓ x, y, z – массивы с данными

Остальные параметры как у plotArrayPar.

Комбинации типов массивов x, y и z должны соответствовать таблице 2. Массивы x и y обязательно должны быть одного типа.

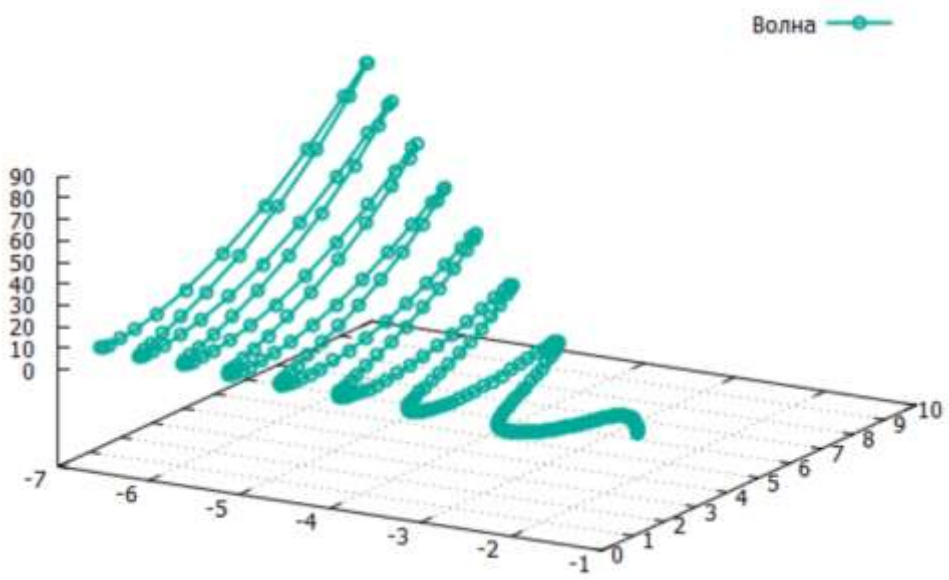
Таблица 2

	Название	x	y	z или g(x, y)
1	INT	`integer`	`integer`	`integer`
2	REAL	`real`	`real`	`real`
3	MIX_1	`integer`	`integer`	`real`
4	MIX_2	`real`	`real`	`integer`

Пример:

```
real, dimension(0:300) :: x, y, z
type(GnuP_f) :: p
do i=0,300
```

```
        x(i) = -(50+i)/50.0
        y(i) = -x(i)/2*exp(sin(x(i)*x(i)))
        z(i) = y(i)*y(i)
    end do
    call GnuPP(p)
    call plotArrayPar_3d(p,300,x,y,z,3,2,14)
    call plot_3d(p)
```



8. plotFuncPar_3d

Аналог plotFuncPar, но в данном случае к параметрам добавляется ещё один массив значений – y; пользовательская функция f должна зависеть от двух аргументов f(x, y).

```
plotFuncPar_3d (GnuP_f p, int n, type0 x, type0 y, type1 f, int line, int width, int color, string legend)
```

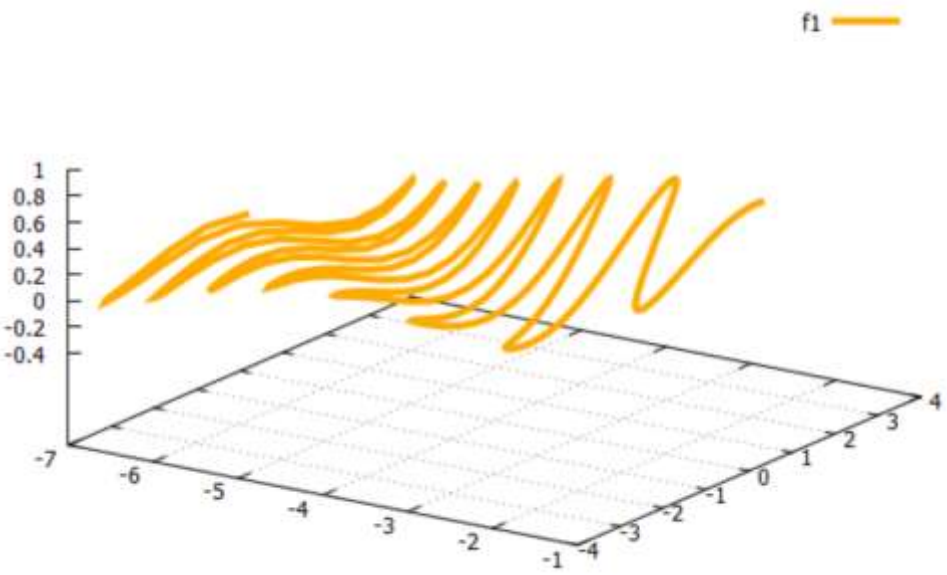
Как и в двумерном случае, функция f обязательно должна быть внешней (или модульной) и должна соответствовать интерфейсу:

```
type1 function f(x,y)
    type0 :: x,y
end function
```

Пример:

```
program main
    use GnuP
    implicit none
```

```
      external f_3d
      real f_3d
      real, dimension(0:300) :: x, y
      type(GnuP_f) :: p
      do i=0,300
          x(i) = -(50+i)/50.0
          y(i) = -x(i)/2*(cos(x(i)*x(i)))
      end do
      call GnuPP(p)
      call plotFuncPar_3d(p,300,x3d,y3d,f_REAL_3d,2,4,12)
      call plot_3d(p)
end
real function f_3d(x,y)
    real :: x,y
    f_3d = sin(x+y)/(x+y)
    return
end function
```



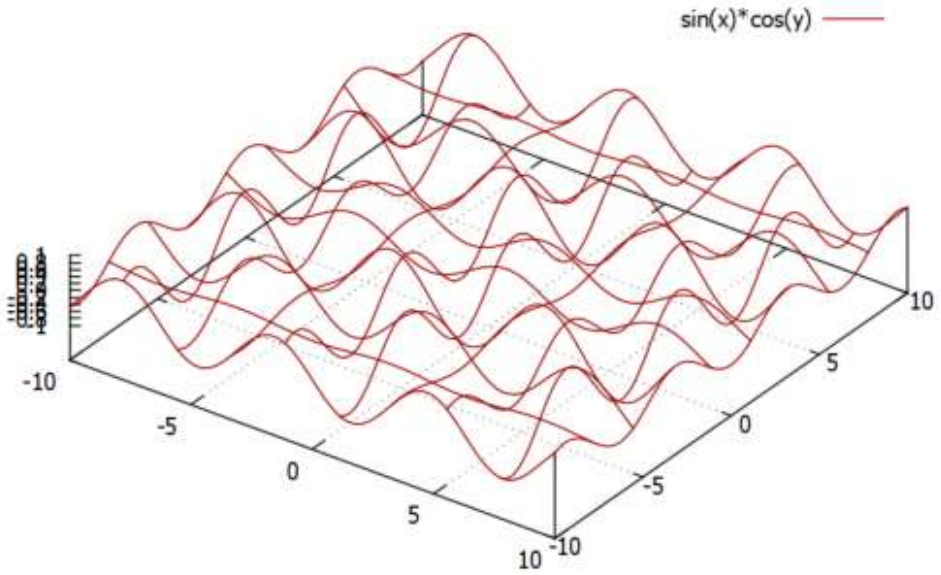
И можно передавать в качестве данных только строку, в которой записано уравнение поверхности для построения, и параметры:

```
plotFuncPar_3d (GnuP_f p, string f, int line, int width,
int color, string legend)
```

Пример:

```
type(GnuP_f) :: p
```

```
call GnuPP(p)
call plotFuncPar_3d(p,"sin(x)*cos(y)",2,0,10)
call plot_3d(p)
```



9. setRange

При необходимости можно задать область для построения графиков. В этом поможет setRange.

```
setRange(GnuP_f p, type x1, type x2, type y1, type y2, type z1, type z2)
```

В качестве аргументов выступают параметры, задающие границы $[x_1, x_2]$, $[y_1, y_2]$, $[z_1, z_2]$ по x, y и z соответственно. Все они должны быть одного типа (либо integer, либо real).

Метод применим в случаях 2d и 3d. Можно определить, например, границы для 2d так, чтобы $x \in [1, 5]$, $y \in [-3, 10.5]$ (параметры для оси z просто опускаются):

```
setRange(p, 1.0, 5.0, -3.0, 10.5)
```

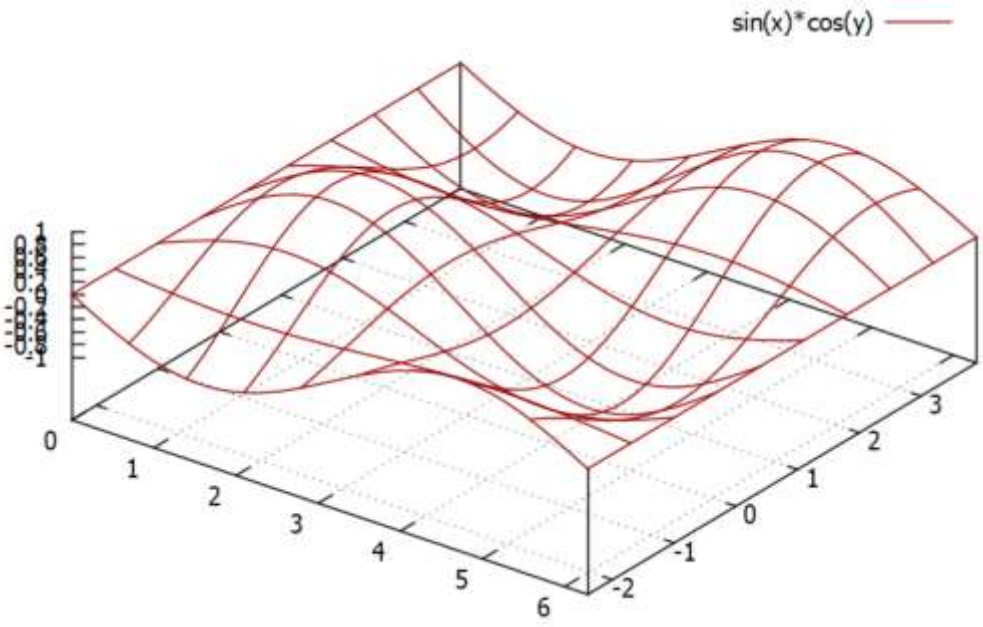
Если хочется настроить только одну ось, допустим, чтобы $z \in [-7, 7]$, то границы для других осей, предшествующие задаваемым границам, передаются в качестве 0:

```
setRange(p, 0, 0, 0, 0, -7, 7)
```

Важно: если данные для графика определяются массивами данных или пользовательской функцией, то обязательно должны быть точки, принадлежащие области, получаемой при масштабировании осей.

Пример (дополним пример из п. plotFuncPar_3d) :

```
type(GnuP_f) :: p
call GnuPP(p)
call setRange(p, 0.0 ,6.28, -2.4, 3.9)
call plotFuncPar_3d(p,"sin(x)*cos(y)",2,0,10)
call plot_3d(p)
```



Передавать можно только *четное* количество аргументов (2, 4 или 6). При этом, в каждой паре первый должен быть меньше второго.

Чтобы сбросить установленные параметры, нужно вызвать метод с двумя 0:

```
call setRange(p,0,0)
```

10. setParam

Метод для настройки общих параметров. Подходит как для 2d, так и для 3d.

```
setParam(GnuP_f p, int grid, int shape, int loc_legend_1,
int loc_legend_2, string title)
```

- ✓ grid – сетка
- ✓ shape – форма окна вывода
- ✓ loc_legend_1 – расположение легенды
- ✓ loc_legend_2 – расположение легенды
- ✓ title – общий заголовок

Сетка:

- ✓ 1 – вкл. (по умолч.)
- ✓ 2 – выкл.
- ✓ 0 – поставить по умолчанию

Форма окна вывода:

- ✓ 1 – квадратное (только 2d)
- ✓ 2 – прямоугольное (по умолч.)
- ✓ 0 – поставить по умолчанию

Расположение легенды (loc_legend_1):

- ✓ 1 – слева сверху
- ✓ 2 – справа сверху (по умолч.)
- ✓ 3 – слева снизу (только 2d)
- ✓ 4 – справа снизу (только 2d)
- ✓ 0 – поставить по умолчанию

Расположение легенды (loc_legend_2):

- ✓ 1 – внутри окна с графиками (по умолч.)
- ✓ 2 – снаружи окна с графиками
- ✓ 0 – поставить по умолчанию

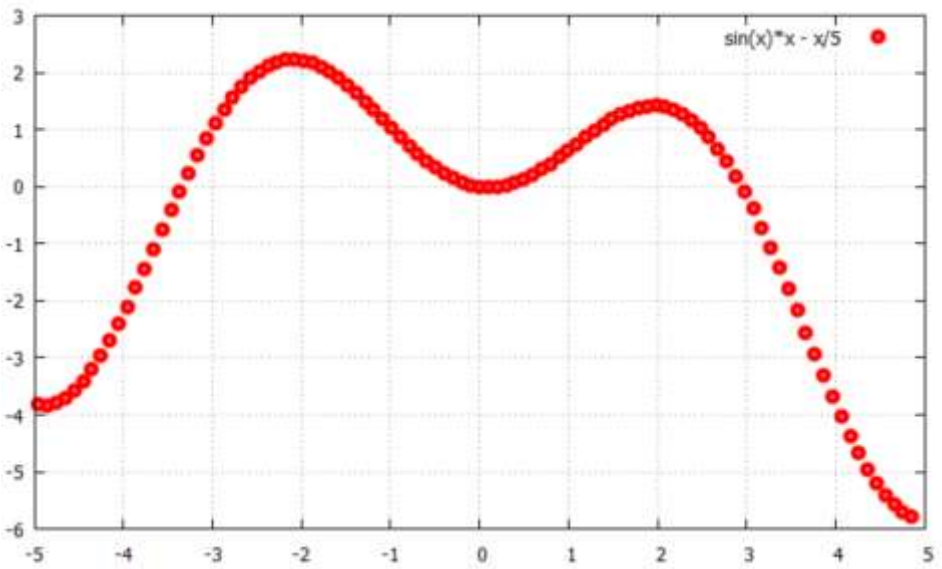
Общий заголовок может быть любой.

Так же, если после всех интересующих параметров имеются ещё какие-либо, их можно опустить. Например, настройка только формы окна:

```
setParam(p,0,1)
```

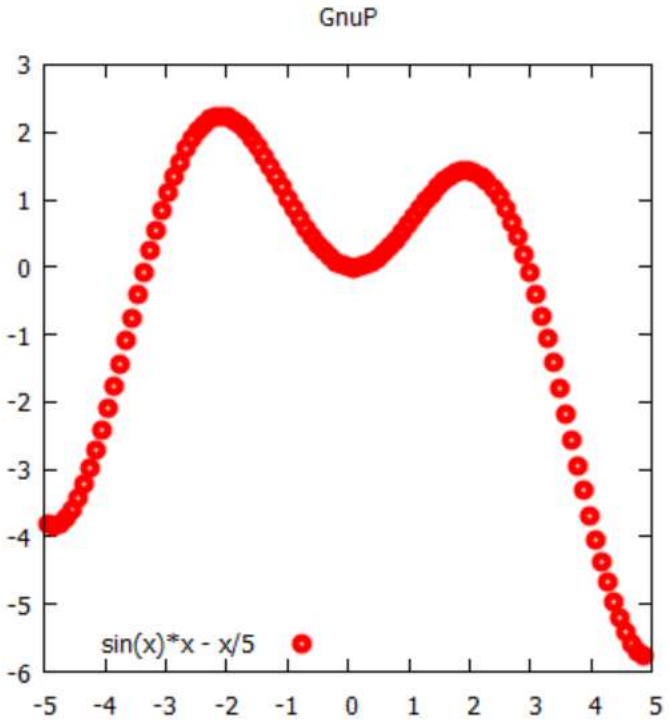
Пример:

С параметрами по умолчанию (см. код в п. plotFuncPar)



С настроенными параметрами – вставим в код после строки `call GnuPP(p)` *строку:*

```
call setParam(p,2,1,3,1,"GnuP")
```



Чтобы сбросить установленные параметры, нужно вызвать метод без параметров (только с объектом класса):

```
call setParam(p)
```

11. setParam_3d

Позволяет определить визуальные настройки 3d графиков.

```
setParam_3d (GnuP_f p, int hidden, int pm3d, int iso_1, int iso_2)
```

- ✓ hidden – прозрачность поверхности
- ✓ pm3d – цветовая палитра поверхности
- ✓ iso_1 – частота разбиения (по x и y, либо только по x)
- ✓ iso_2 – частота разбиения (по y)

Прозрачность поверхности:

- ✓ 1 – вкл. (по умолч.)
- ✓ 2 – выкл.
- ✓ 0 – поставить по умолчанию

Цветовая палитра поверхности:

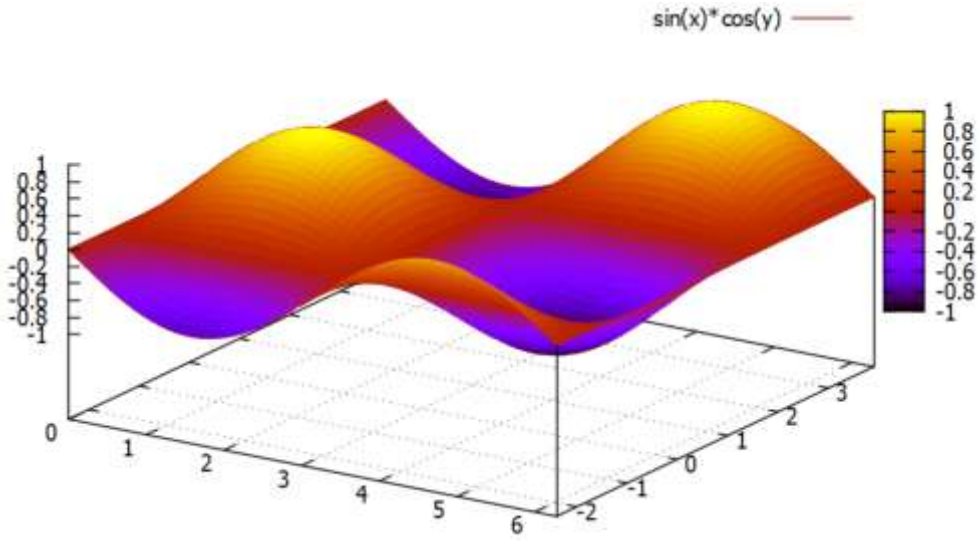
- ✓ 1 – вкл.
- ✓ 2 – выкл. (по умолч.)
- ✓ 0 – поставить по умолчанию

Если график будет представлять собой одну линию, то применить цветовую палитру не получится. Поэтому данный параметр лучше использовать в тандеме с plotFuncPar_3d, задавая функцию строкой.

Частота разбиения – количество точек на оси, в которых будет вычисляться значение функции. Может задаваться одним целым числом (параметр iso_1). В этом случае оно будет определять сразу и разбиение по x, и по y. Если же ввести оба параметра, то iso_1 будет определять разбиение по x, а iso_2 – по y.

Пример (дополним пример из п. setRange):

```
type(GnuP_f) :: p
call GnuPP(p)
call setRange(p, 0.0 ,6.28, -2.4, 3.9)
p.setParam_3d(1,1,60,60);
call plotFuncPar_3d(p,"sin(x)*cos(y)",2,0,10)
call plot_3d(p)
```



Чтобы сбросить установленные параметры, нужно вызвать метод без параметров (только с объектом класса):

```
call setParam_3d(p)
```

12. clearData

Очищает данные о 2d графиках, которые были сохранены до вызова clearData.

```
call clearData(p)
```

13. clearData_3d

Очищает данные о 3d графиках, которые были сохранены до вызова clearData_3d.

```
call clearData_3d(p)
```

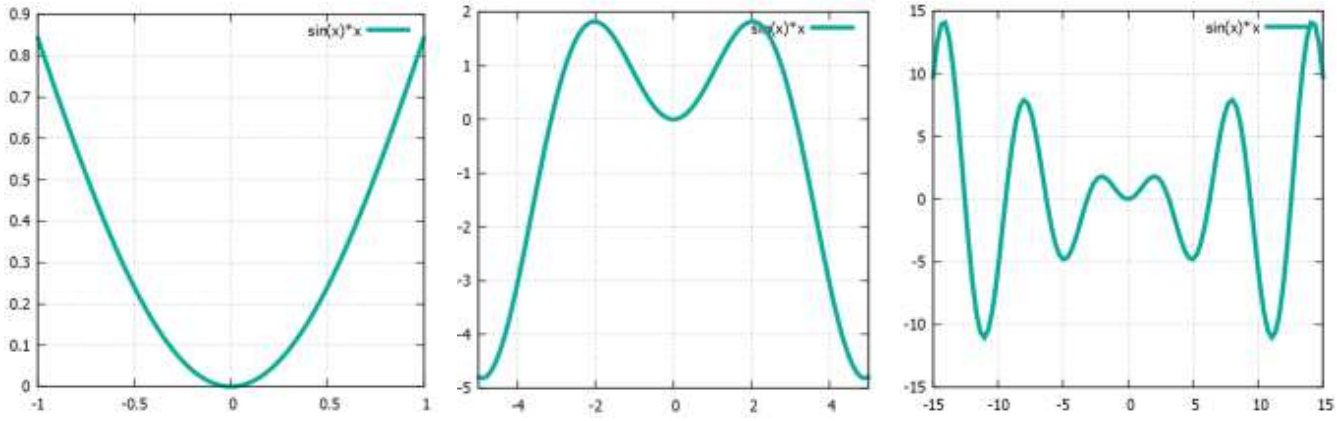
14. Повторный вызов plot и plot_3d

В одной программе можно несколько раз вызвать методы plot и plot_3d.

Допустим, хотим посмотреть, как выглядит график функции x·sin(x), но изначально не знаем, на каком отрезке лучше строить. Поэтому задаем цикл из 3 итераций, в котором будем вводить и устанавливать значения a и b: x∈[a, b], а потом вызывать plot, чтобы увидеть график.

```
integer :: a, b
type(GnuP_f) :: p
call GnuPP(p)
call plotFuncPar(p,"sin(x)*x",2,4,14)
call setParam(p,0,1)
do i=1,3
     print '("a = "$)'
     read *, a
     print '("b = "$)'
     read *, b
     call setRange(p,a,b)
     call plot(p)
end do
```

На первой итерации введем a = -1 и b = 1, на второй – a = -5 и b = 5, на третьей – a = -15 и b = 15. После каждого ввода на экране будет появляться квадратное окно с графиком, где x∈[a, b]. Переход к следующей итерации будет возможен после закрытия окна с графиком.



15. Компиляция и сборка

Чтобы собрать программу с GnuP, необходимо выполнить следующие команды:

Linux:

```
gfortran -c GnuP.f95
gfortran -g -c main.f95 -o main.o
gfortran -o a.out GnuP.o main.o
./a.out
```

Windows:

```
gfortran -c GnuP.f95
gfortran -g -c main.f95 -o main.o
gfortran -o a.exe GnuP.o main.o
./a.exe
```

Здесь:

- ✓ GnuP.f95 – библиотека
- ✓ main.f95 – главная программа

1) Компилируется модуль GnuP.f95, в результате чего создаётся объектный «GnuP.o» и промежуточный файл модуля «gnup.mod».

2) Компилируется программа main.f95, в результате чего создаётся объектный файл «main.o».

3) Вызывается компоновщик для объединения объектных файлов GnuP.o и main.o в исполняемый файл программы – с именем по умолчанию «a.out» или «a.exe» (.out для Linux, .exe для Windows). Если вы хотите другое имя исполняемого файла, используйте опцию компилятора «-o».

4) Запуск исполняемого файла программы.

Если у Вас имеются вопросы или предложения, можете писать по адресу polinka.sestra@mail.ru