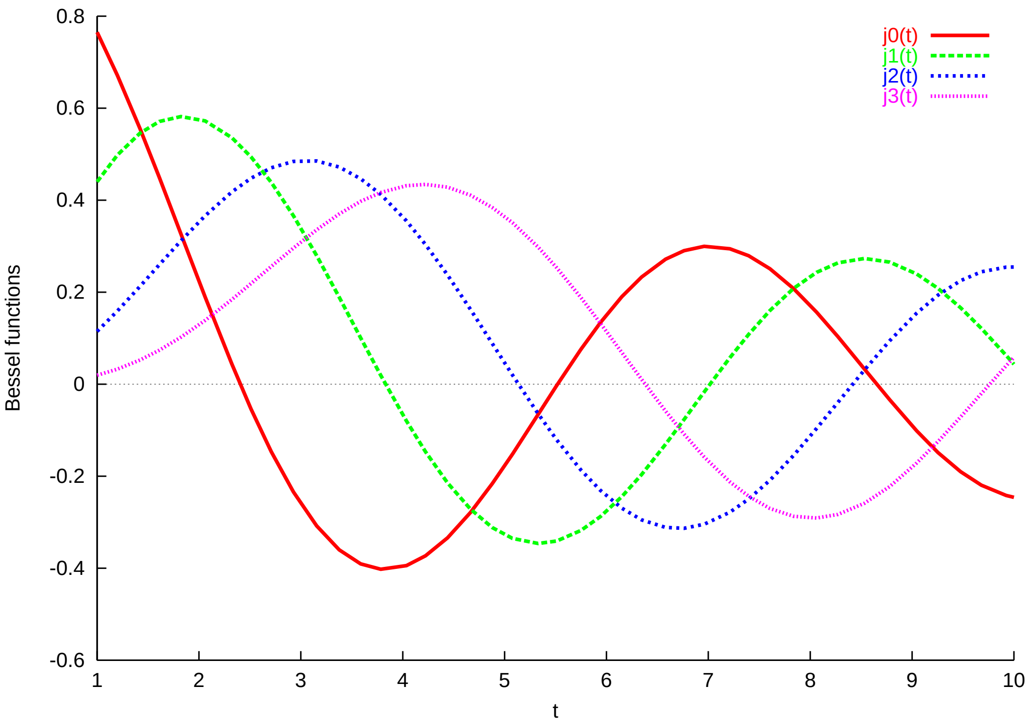
SoftIntegration®

SoftIntegration Graphical Library

Version 2.9

User's Guide



How to Contact SoftIntegration

Mail SoftIntegration, Inc.
216 F Street, #68
Davis, CA 95616
Phone + 1 530 297 7398
Fax + 1 530 297 7392
Web http://www.softintegration.com
Email info@softintegration.com

Revision 2.9.0, April 2010

SoftIntegration, Inc. makes no representations, expressed or implied, with respect to this documentation, or the software it describes, including without limitations, any implied warranty merchantability or fitness for a particular purpose, all of which are expressly disclaimed. Users should be aware that included in the terms and conditions under which SoftIntegration is willing to license the software as a provision that SoftIntegration, and their distribution licensees, distributors and dealers shall in no event be liable for any indirect, incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of the software, and that liability for direct damages shall be limited to the amount of purchase price paid for the software.

Ch, SoftIntegration, and One Language for All are either registered trademarks or trademarks of SoftIntegration, Inc. in the United States and/or other countries. Other product or brand names are trademarks or registered trademarks of their respective holders.

Typographical Conventions

The following list defines and illustrates typographical conventions used as visual cues for specific elements of the text throughout this document.

- Interface components are window titles, button and icon names, menu names and selections, and other options that appear on the monitor screen or display. They are presented in boldface. A sequence of pointing and clicking with the mouse is presented by a sequence of boldface words.

  Example: Click OK

  Example: The sequence Start->Programs->Ch6.3->Ch indicates that you first select Start. Then select submenu Programs by pointing the mouse on Programs, followed by Ch6.3. Finally, select Ch.

- Keycaps, the labeling that appears on the keys of a keyboard, are enclosed in angle brackets. The label of a keycap is presented in typewriter-like typeface.

  Example: Press `<Enter>`

- Key combination is a series of keys to be pressed simultaneously (unless otherwise indicated) to perform a single function. The label of the keycaps is presented in typewriter-like typeface.

  Example: Press `<Ctrl><Alt><Enter>`

- Commands presented in lowercase boldface are for reference only and are not intended to be typed at that particular point in the discussion.

  Example: “Use the install command to install...”

  In contrast, commands presented in the typewriter-like typeface are intended to be typed as part of an instruction.

  Example: “Type `install` to install the software in the current directory.”

- Command Syntax lines consist of a command and all its possible parameters. Commands are displayed in lowercase bold; variable parameters (those for which you substitute a value) are displayed in lowercase italics; constant parameters are displayed in lowercase bold. The brackets indicate items that are optional.

  Example: **ls [-aAbcCdfFgilLmnopqrRstux1] [*file* ...]**

- Command lines consist of a command and may include one or more of the command’s possible parameters. Command lines are presented in the typewriter-like typeface.

  Example: `ls /home/username`

- Screen text is a text that appears on the screen of your display or external monitor. It can be a system message, for example, or it can be a text that you are instructed to type as part of a command (referred to as a command line). Screen text is presented in the typewriter-like typeface.

  Example: The following message appears on your screen

  ```
  usage:  rm [-fiRr] file ...
  ```

```
ls [-aAbcCdfFgilLmnopqrRstux1] [file ... ]
```

- Function prototype consists of return type, function name, and arguments with data type and parameters. Keywords of the Ch language, typedefed names, and function names are presented in boldface. Parameters of the function arguments are presented in italic. The brackets indicate items that are optional.

  Example: double derivative(double (**func*)(double), double *x*, ... [double **err*, double *h*]);

- Source code of programs is presented in the typewriter-like typeface.

  Example: The program hello.ch with code

  ```
  int main() {
      printf("Hello, world!\n");
  }
  ```

  will produce the output `Hello, world!` on the screen.

- Variables are symbols for which you substitute a value. They are presented in italics.

  Example: module *n* (where *n* represents the memory module number)

- System Variables and System Filenames are presented in boldface.

  Example: startup file /home/username/.chrc or .chrc in directory `/home/username` in Unix and C:\ >_chrc or _chrc in directory C:\ > in Windows.

- Identifiers declared in a program are presented in typewriter-like typeface when they are used inside a text.

  Example: variable `var` is declared in the program.

- Directories are presented in typewriter-like typeface when they are used inside a text.

  Example: Ch is installed in the directory `/usr/local/ch` in Unix and `C:/Ch` in Windows.

- Environment Variables are the system level variables. They are presented in boldface.

  Example: Environment variable PATH contains the directory /usr/ch.

Table of Contents

1 Installation and Compilation **1**

- 1.1 System Requirements . . . 1
  - 1.1.1 System Requirement for Windows 95/98/Me/NT/2000/XP . . . 1
  - 1.1.2 System Requirement for Unix and Mac OS X . . . 1
- 1.2 Install and Build Executables with SIGL in Windows . . . 2
  - 1.2.1 Install SIGL in Windows . . . 2
  - 1.2.2 Windows Environment Settings . . . 3
  - 1.2.3 Uninstall SIGL in Windows . . . 3
  - 1.2.4 Build Executables in Windows . . . 3
    - 1.2.4.1 Build Executables in Windows Using Visual .NET . . . 3
    - 1.2.4.2 Build Executables Using Borland C/C++ Compiler . . . 4
- 1.3 Install and Build Executables with SIGL in Unix . . . 4
  - 1.3.1 Install SIGL in Unix . . . 4
  - 1.3.2 Uninstall SIGL in Unix . . . 4
  - 1.3.3 Build Executables in Unix . . . 4
- 1.4 Install and Build Executables with SIGL in Max OS X . . . 6
  - 1.4.1 Install SIGL in Mac OS X . . . 6
- 1.5 Setup for Plotting Using AquaTerm . . . 6
  - 1.5.1 Uninstall SIGL in Mac OS X . . . 6
  - 1.5.2 Build Executables in Mac OS X . . . 6

2 Two and Three-Dimensional Plotting **8**

- 2.1 A Class for Plotting . . . 8
  - 2.1.1 Data for Plotting . . . 8
  - 2.1.2 Annotations . . . 13
  - 2.1.3 Multiple Data Sets and Legends . . . 16
  - 2.1.4 Subplots . . . 22
  - 2.1.5 Export Plots . . . 22
  - 2.1.6 Print Plots . . . 24
    - 2.1.6.1 Printing Plots in Windows . . . 24
    - 2.1.6.2 Printing Plots in Unix . . . 27
- 2.2 2D Plotting . . . 27
  - 2.2.1 Plot Types, Line Styles, and Markers . . . 27
  - 2.2.2 Polar Plot . . . 32
- 2.3 3D Plotting . . . 32
  - 2.3.1 Plot Types . . . 32
  - 2.3.2 Contour Plots . . . 38

2.3.3 Plotting in Different Coordinate Systems 39

2.4 Dynamic Web Plotting 39

3 Distribution of Applications with SIGL 47

3.1 Build an Application for Distribution 47

3.2 Distribution in Windows 47

3.3 Distribution in Unix and Mac OS X 48

4 Reference for Class CPlot 49

CPlot Class 49

arrow 55

autoScale 59

axis 60

axisRange 62

axes 66

barSize 68

border 69

borderOffsets 71

boxBorder 72

boxFill 73

boxWidth 76

changeViewAngle 77

circle 79

colorBox 81

contourLabel 83

contourLevels 85

contourMode 88

coordSystem 91

data 96

data2DCurve 100

data3DCurve 101

data3DSurface 104

dataFile 106

dataSetNum 109

deleteData 110

deletePlots 111

dimension 112

displayTime 112

enhanceText 113

func2D 118

func3D 119

funcp2D 121

funcp3D 122

getLabel 124

getOutputType 125

getSubplot 126

getTitle 130

grid 131

isUsed . . . 134
label . . . 134
legend . . . 136
legendLocation . . . 138
legendOption . . . 139
line . . . 142
lineType . . . 145
margins . . . 149
origin . . . 150
outputType . . . 151
plotType . . . 160
plotting . . . 189
point . . . 189
pointType . . . 191
polarPlot . . . 194
polygon . . . 196
rectangle . . . 200
removeHiddenLine . . . 202
scaleType . . . 205
showMesh . . . 206
size . . . 209
size3D . . . 210
sizeOutput . . . 211
sizeRatio . . . 212
smooth . . . 214
subplot . . . 215
text . . . 216
tics . . . 217
ticsDay . . . 218
ticsDirection . . . 220
ticsFormat . . . 221
ticsLabel . . . 223
ticsLevel . . . 225
ticsLocation . . . 227
ticsMirror . . . 229
ticsMonth . . . 230
ticsPosition . . . 232
ticsRange . . . 233
title . . . 234
fplotxy . . . 236
fplotxyz . . . 238
plotxy . . . 241
plotxyf . . . 243
plotxyz . . . 245
plotxyzf . . . 248

5 Appendix A: Differences between Ch and C++ for Graphical Plotting **251**

6 Appendix B: Source Code for the Figure on the Cover Page 252

7 Porting Code to the Latest Version 253
7.1 Porting Code to SIGL Version 2.5 253

Index 255

Chapter 1

Installation and Compilation

This chapter describes the system requirement and installation of the SoftIntegration Graphical Library (SIGL) as well as its compilation with application programs in both Windows and Unix.

1.1 System Requirements

1.1.1 System Requirement for Windows 95/98/Me/NT/2000/XP

To install and use SIGL for Windows, the following minimum requirements should be met:

- Operating System: Windows 95/98/Me/2000/XP/Windows NT workgroup or Server 4.0 with SP3 or above
- Supported compilers: Visual Studio .NET VC++ 2005 Service Pack 1 or higher, Borland C++ Compiler, Borland C++ Builder
- CPU: with a 486 or higher microprocessor
- Memory: a minimum of 16 Megabytes of RAM
- Disk Space: 6 Mb

1.1.2 System Requirement for Unix and Mac OS X

For Unix, the supported Operating System is

- Intel Linux 2.4.20-8 or above with gcc/g++ 3.2.2 or above.
- Mac OS X 10.3 or above
- Sparc Solaris 2.6 or above
- HP-UX 10.20 or above

The hardware requirement for the Intel Linux platform is

- Pentium/90Mhz or above
- A minimum of 16 Megabytes of RAM
- Disk Space Requirement. 6 Mb

1.2 Install and Build Executables with SIGL in Windows

1.2.1 Install SIGL in Windows

Before starting the installation, close all running applications. If you have installed an older version SIGL before, uninstall it off the system first. Note that SILIB_HOME is not the string "SILIB_HOME". Rather, it is the Windows file system path under which SIGL is installed. For instance use C:\sigl for SILIB_HOME in Windows. Make sure SILIB_HOME is different from the home directory CHHOME for the Ch language environment.

To start the installation process from a CD:

1. **Login to the computer with an Administrator privilege under NT/2000/XP, or login to the computer in Windows 95/98/Me.**
2. **Insert the SIGL setup CD into the CD-ROM drive.**
3. **On Windows 95/98/Me and Windows NT/2000/XP, the setup process starts automatically if AutoPlay for CDs is enabled. Click** `Next` **to continue.**

   If AutoPlay for CDs is not enabled, use Windows Explorer to navigate from the root directory of the CD. Then, double-click the `Setup.exe` **file.**
4. **Read and accept the SoftIntegration license agreement.**
5. **Enter the product Serial Number**
6. **Accept default folder names.**
7. **Accept the typical installation and press** `Next`
8. **Follow the instructions of the setup program to install SIGL on your computer.**
9. **Click** `Finish` **to complete the installation**

Note: You are able to quit the installation at any time by pressing the `<Cancel>` **button displayed in every dialog box during the installation. You can also move back and forth to review your settings by clicking the** `<Back>` **and** `<Next>` **buttons.**

The compilation and runtime libraries are located in SILIB_HOME/lib. If you computer has VC++ 2005 installed, SILIB_HOME/lib contains the library for VC++ 2005. Otherwise, it contains the library for VC++ 2008.

If you computer has VC++ 2005 installed while the SIGL is installed, and later you upgrade your compiler from VC++ 2005 to VC++ 2008, (a) you need to copy files in the diretory SILIB_HOME/lib/VC2008 to SILIB_HOME/lib; (b) you need to copy the dynamical loaded library file SILIB_HOME/lib/VC2008/libchplot.dll to the directory C:/Windows/System32 for Windows 32-bit or C:/Windows/SysWoW64 for Windows 64-bit.

If you install SIGL in a machine first, then install VC++ 2005, (a) you need to copy files in the diretory SILIB_HOME/lib/VC2005 to SILIB_HOME/lib; (b) you need to copy the dynamical loaded library file SILIB_HOME/lib/VC2005/libchplot.dll to the directory C:/Windows/System32 for Windows 32-bit or C:/Windows/SysWoW64 for Windows 64-bit.

To avoid the issues related to different compiler versions of VC++ and related manifest files, one may use the static SIGL library SILIB_HOME/lib/VC2005/libchplot_a.lib.

1.2.2 Windows Environment Settings

For Windows, SIGL will create and set SILIB_HOME in its registry. The value of C:\silib is the default directory where you have SIGL installed.

1.2.3 Uninstall SIGL in Windows

Stop all the applications using the SIGL.

Click Control Panel in My Computer. Click Add/Remove Programs, select SoftIntegration Graphical Library 1.0. then Click Add/Remove .... Press Yes if you are asked to completely remove SIGL and all of its components.

1.2.4 Build Executables in Windows

The header file chplot.h for SIGL is located in `SILIB_HOME/include/chplot.h`**. The library libchplot.lib is located in** `SILIB_HOME/lib/libchplot.lib`**. The dynamically loaded library libchplot.dll is installed in the Windows system directory during installation of SIGL. You can build applications using SIGL accordingly with a C++ compiler. Directory** `SILIB_HOME/demos` **contains source code described in the next two chapters and** `Makefiles` **for compiling these sample code using Microsoft Visual C++. These sample code use macro** `M_PI` **for π, which is defined in header file** `math.h` **in Visual C++. By default, this mathematical macro is not available. To use this macro, the program needs to be compiled with the macro _USE_MATH_DEFINES defined. If the programs are compiled using integrated development environment, make sure macro _USE_MATH_DEFINES is defined in the environment or the macro** `M_PI` **for π is defined inside a C++ program with the following statement.**

```
#ifndef M_PI
#define M_PI  3.14159265358979323846
#endif
```

1.2.4.1 Build Executables in Windows Using Visual .NET

Assume SIGL is installed in the directory `C:\silib`**. To compile the code using SIGL in Visual .NET, follow the procedures below.**

- **Create a Win32 project or Win32 console project.**
- **Header file chplot.h is located in the directory** `C:/silib/include`**. Add the directory** `C:/silib/include` **to the header file search path. Click** `Project => Properties ...`**, it will pop up an Properties pages window. From left panel, click** `Configuration Properties => C/C++ => General`**, select** `Additional Include Directories` **from right panel, Add the include path** `C:\silib\include`.
- **The application program should be linked with SIGL Ch library libchplot.lib located in the directory** `C:/silib/lib`**, To add this directory, in the library search path as follows. Under the same Configuration Properties Window, select** `Linker`**, select** `General`**, then,** `Additional Library Directories` **from right panel, Add** `C:\silib\lib`**, then**

click
`Apply` **to finalize the settings in Property Window. Click** `Command Line` **under** `Linker`**, then select** `Additional Options`**, Add** `libchplot.lib`**. Finally, click** `OK` **to finalize the settings in Project Settings.**

The source code for Visual .NET project file `data2D.vcproj` **for Porgram 2.1 described in sections 2.1.1 can be found in the distribution of SIGL for Windows in the directory** `SILIB_HOME/demos/VC/VC.NET`**. It can be opened from Visual .NET directly.**

1.2.4.2 Build Executables Using Borland C/C++ Compiler

To build executables using Borland C/C++ compiler, use header file chplot.h located in `SILIB_HOME/include/chplot.h`**. The library libchplot_bc.lib is located in** `SILIB_HOME/lib/libchplot_bc.lib`**. The dynamically loaded library libchplot_ch.dll is installed in the Windows system directory during installation of SIGL. You can build applications using SIGL accordingly with a C++ compiler. Directory** `SILIB_HOME/demos/Borland` **contains source code and sample Makefile for building applications using Borland C/C++ compiler.**

1.3 Install and Build Executables with SIGL in Unix

1.3.1 Install SIGL in Unix

If you have installed an older version SIGL, uninstall that version off the system first. Note that SILIB_HOME is not the string "SILIB_HOME". Rather, it is the Unix file system path under which SIGL is installed. Under Unix, the default directory for installing SIGL is /usr/local/silib under root, or HOME/silib under a general user. Make sure SILIB_HOME is different from the home directory CHHOME for the Ch language environment.
If you have the CD with you, install using the following steps.

1. **Insert the SIGL setup CD into the CD-ROM drive. Depending on how your operating system is configured, your CD drive may be mounted automatically. If the CD drive is not mounted, you must mount it before continuing.**
2. **Go to your CD-ROM directory where the CD-ROM is mounted.**
3. **Run the following command.**

```
sh ./install.sh
```

1.3.2 Uninstall SIGL in Unix

Take the following steps:

- **Remove all components of SIGL from the SILIB_HOME directory where you installed it.**

1.3.3 Build Executables in Unix

The header file chplot.h for SIGL is located in `SILIB_HOME/include/chplot.h`**. The directory** `SILIB_HOME/lib` **contains both static and dynamical libraries for SIGL. Only one of libraries needs to be linked for your applications. You can build applications using SIGL accordingly with a C++**

compiler. Directory `SILIB_HOME/demos` contains source code described in the next two chapters and `Makefiles` for compiling these sample code.

To use SIGL, environment variable SILIB_HOME needs to be setup. In the following description, we assume that SIGL is installed in /usr/local/silib. If SIGL is not installed in this directory, change the code accordingly.

If you use Ch shell, add the following command to the startup file `.chrc` in your home directory,

```
putenv("SILIB_HOME=/usr/local/silib");
```

If you use 'csh' or 'tcsh' shell, you need to add the following command to the startup file `.cshrc` or `.tcshrc` in your home directory, respectively.

```
setenv SILIB_HOME /usr/local/silib
```

If you use 'sh', 'ksh', or 'bash' shell, you need to add the following command to the startup file `.bashrc` in your home directory.

```
export SILIB_HOME=/usr/local/silib
```

If the dynamically loaded library of SIGL is used on for Solaris or Linux, the environment variable `LD_LIBRARY_PATH` shall include the path `SILIB_HOME/lib` to load the library at runtime. The environment variable can be updated in a startup file under different Unix shells. Assume `/usr/local/silib` is the home for `SILIB_HOME`, under a Ch shell, use the following code in its startup file `.chrc` in your home directory.

```
if(getenv("LD_LIBRARY_PATH")!=NULL)
    putenv(stradd("LD_LIBRARY_PATH=/usr/local/silib:",
                  getenv("LD_LIBRARY_PATH")));
else
    putenv("LD_LIBRARY_PATH=/usr/local/silib");
```

Under a 'csh' or 'tcsh' shell, use the following code its statup file `.cshrc` or `.tcshrc` in your home directory, respectively.

```
if ($?LD_LIBRARY_PATH) then
   setenv LD_LIBRARY_PATH /usr/local/silib:${LD_LIBRARY_PATH}
else
    setenv LD_LIBRARY_PATH /usr/local/silib
endif
```

Under a 'sh', 'ksh', or 'bash' shell, use the following code its statup file `.bashrc` in your home directory.

```
if test "${LD_LIBRARY_PATH}" = ""
then
   export LD_LIBRARY_PATH=/usr/local/silib
else
   export LD_LIBRARY_PATH=/usr/local/silib:${LD_LIBRARY_PATH}
fi
```

Alternatively, in Linux, you may build your applications with the following link option to set the runtime library explicitly.

```
 -Wl,-R${SILIB_HOME}/lib -lchplot
```

1.4 Install and Build Executables with SIGL in Max OS X

1.4.1 Install SIGL in Mac OS X

If you have installed an older version SIGL, uninstall that version off the system first. Note that SILIB_HOME is not the string "SILIB_HOME". Rather, it is the Unix file system path under which SIGL is installed. Under Max OS X, the default directory for installing SIGL is /usr/local/silib under root, or HOME/silib under a general user.
If you have the CD with you, install using the following steps.

1. **Insert the SIGL setup CD into the CD-ROM drive. Depending on how your operating system is configured, your CD drive may be mounted automatically. If the CD drive is not mounted, you must mount it before continuing.**
2. **Go to your CD-ROM directory where the CD-ROM is mounted.**
3. **Run the following command.**

```
sudo ./install.sh
```

1.5 Setup for Plotting Using AquaTerm

Plots in SIGL Edition can be displayed using either X11 or AquaTerm.

Plots in SIGL are displayed using X11 by default. Installation instructions for X11 can be found by searching for "X11 install" on Apple Computer's web site http://www.apple.com.

AquaTerm is an open source application for Mac OS X that provides a GUI interface for plotting programs. To use AquaTerm for displaying plots in SIGL, follow the instructions below to set it up.

1. **Downloaded AquaTerm from the internet at http://aquaterm.sourceforge.net and install it.**
2. **Setup the environment variable `GNUTERM` with the value `aqua`.**

1.5.1 Uninstall SIGL in Mac OS X

Take the following steps:

- **Remove all components of SIGL from the SILIB_HOME directory where you installed it.**
- **Remove the package receipt file "/Library/Receipts/sigl-X.Y.Z.pkg" where X.Y.Z stands for SIGL version number such as 1.0.0.**

1.5.2 Build Executables in Mac OS X

The header file chplot.h for SIGL is located in `SILIB_HOME/include/chplot.h`. The directory `SILIB_HOME/lib` contains both static and dynamical libraries for SIGL. Only one of libraries needs to be linked for your applications. You can build applications using SIGL accordingly with a C++ compiler. Directory `SILIB_HOME/demos` contains source code described in the next two chapters and `Makefiles` for compiling these sample code.

To use SIGL, environment variable SILIB_HOME needs to be setup. In the following description, we assume that SIGL is installed in /usr/local/silib. If SIGL is not installed in this directory, change the code accordingly.

If you use Ch shell, add the following command to the startup file `.chrc` in your home directory,

```
putenv("SILIB_HOME=/usr/local/silib");
```

If you use 'csh' or 'tcsh' shell, you need to add the following command to the startup file .cshrc or .tcshrc in your home directory, respectively.

```
setenv SILIB_HOME /usr/local/silib
```

If you use 'sh', 'ksh', or 'bash' shell, you need to add the following command to the startup file `.bashrc` **in your home directory.**

```
export SILIB_HOME=/usr/local/silib
```

If the dynamically loaded library of SIGL is used on Mac OS X, the environment variable `DYLD_LIBRARY_PATH` **shall include the path** `SILIB_HOME/lib` **to load the library at runtime. The environment variable can be updated in a startup file under different Unix shells. Assume** `/usr/local/silib` **is the home for** `SILIB_HOME`, **under a Ch shell, use the following code in its startup file** `.chrc` **in your home directory.**

```
if(getenv("DYLD_LIBRARY_PATH")!=NULL)
    putenv(stradd("DYLD_LIBRARY_PATH=/usr/local/silib:",
                  getenv("DYLD_LIBRARY_PATH")));
else
    putenv("DYLD_LIBRARY_PATH=/usr/local/silib");
```

Under a 'csh' or 'tcsh' shell, use the following code.

```
if ($?DYLD_LIBRARY_PATH) then
   setenv DYLD_LIBRARY_PATH /usr/local/silib:${DYLD_LIBRARY_PATH}
else
    setenv DYLD_LIBRARY_PATH /usr/local/silib
endif
```

Under a 'sh', 'ksh', or 'bash' shell, use the following code.

```
if test "${DYLD_LIBRARY_PATH}" = ""
then
   export DYLD_LIBRARY_PATH=/usr/local/silib
else
   export DYLD_LIBRARY_PATH=/usr/local/silib:${DYLD_LIBRARY_PATH}
fi
```

For Mac OS X version 10.2 or higher, the environment variables DYLD_FORCE_FLAT_NAMESPACE shall be set to 1 and DYLD_INSERT_LIBRARIES to /usr/lib/libncurses.dylib. For example, this can be accomplished in Ch shell by commands.

```
putenv("DYLD_FORCE_FLAT_NAMESPACE=1");
putenv("DYLD_INSERT_LIBRARIES=/usr/lib/libncurses.dylib");
```

Under a 'csh' or 'tcsh' shell, use the following code.

```
setenv DYLD_FORCE_FLAT_NAMESPACE 1
setenv DYLD_INSERT_LIBRARIES /usr/lib/libncurses.dylib
```

Under a 'sh', 'ksh', or 'bash' shell, use the following code.

```
export DYLD_FORCE_FLAT_NAMESPACE=1
export DYLD_INSERT_LIBRARIES=/usr/lib/libncurses.dylib
```

Chapter 2

Two and Three-Dimensional Plotting

Two and three dimensional plottings can be conveniently accomplished using the SoftIntegration Graphical Library. Plots can be generated from data arrays or files, and can be displayed on a screen, saved in a large number of different file formats, or generated as a stdout stream in png file format for display in a Web browser through a Web server. This chapter describes how to write C++ programs, compatible with Ch, to generate plots in two and three dimensional spaces.

2.1 A Class for Plotting

The plotting class CPlot enables high-level creation and manipulation of plots for applications in C++ or in the Ch language environment. Member functions of class CPlot are listed on page 49. Detailed description of each function can be found in the reference for CPlot class. In the following subsections, features applicable to both 2D and 3D plotting will be presented.

2.1.1 Data for Plotting

A data set is necessary for creating a plot. The data for a plot can be stored in the memory of the program or in a file. The simplest form of data used for a two-dimensional plot has two arrays, one for x-axis and the other for y-axis as shown in Program 2.1. Figure 2.1 displays the plot produced by Program 2.1. There are two member functions used in Program 2.1. Function CPlot::data2DCurve() adds data for plotting. At the point where function CPlot::plotting() is called, a plot is generated.

The data for plotting of 2D curve can also be added to an instance of CPlot class by the member function

```
int CPlot::data2DCurve(double x[], double y[], int n);
```

Both one-dimensional arrays `x` **and** `y` **have the same number of elements of size** `n`**. Data points for array** `y` **of value** NaN are internally removed before plotting occurs. The “holes” in a data set can be constructed by manually setting elements of `y` to this value.

The data for plotting of 3D curve can be added to an instance of **CPlot** class by the member function

```
int CPlot::data3DCurve(double x[], double y[], double z[], int n);
```

One-dimensional arrays `x`, `y`, and `z` have the same number of elements of size `n`. Program 2.2 with corresponding plot in Figure 2.2 illustrates how a spatial curve can be generated.

A set of data for 3D surface plotting can be added to an instance of **CPlot** class by the member function

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
      x[i] = -M_PI + i*2*M_PI/(NUM-1);  // linspace(x, -PI, PI)
      y[i] = sin(x[i]);
    }
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Program 2.1: A simple program using **CPlot** class.

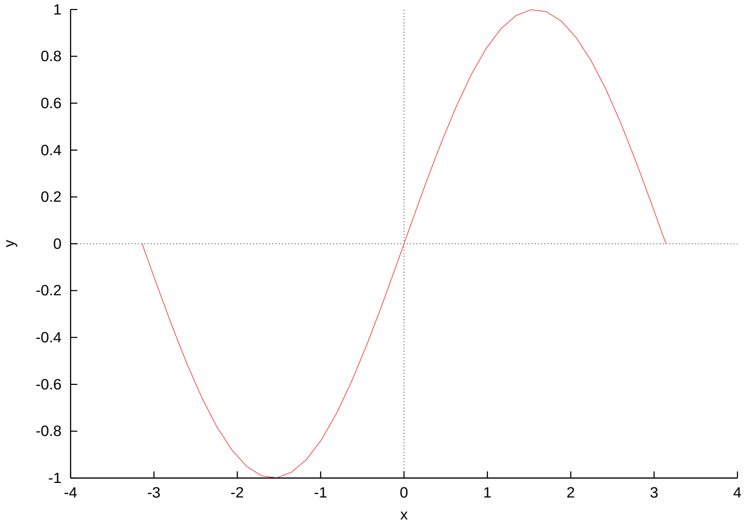


Figure 2.1: A very simple plot.

```
#include <math.h>
#include <chplot.h>

#define NUM 360
int main() {
    int i;
    double x[NUM], y[NUM], z[NUM];
    class CPlot plot;

    for(i=0; i< NUM; i++) {
      x[i] = 0 + i*360.0/(NUM-1);   // linspace(x, 0, 360)
      y[i] = sin(x[i]*M_PI/180);
      z[i] = cos(x[i]*M_PI/180);
    }
    plot.data3DCurve(x, y, z, NUM);
    plot.plotting();
    return 0;
}
```

Program 2.2: A plotting program for a 3D curve.

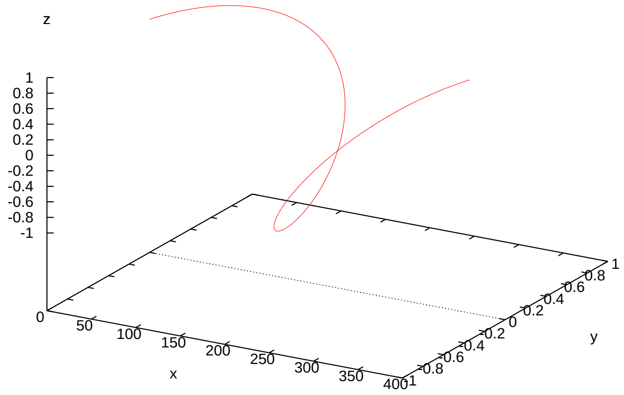


Figure 2.2: A plot with a 3D curve.

```
int CPlot::data3DSurface(double x[], double y[], double z[],
                              int n, int m);
```

If one-dimensional array x has the number of elements of size n, and *y* has size m, *z* shall be a one-dimensional array of size $n_z = n \cdot m$. In a Cartesian coordinate system, arrays *x*, *y*, and *z* represent values in X-Y-Z coordinates, respectively. In a cylindrical coordinate system, arrays *x*, *y*, and *z* represent θ), z, and r coordinates, respectively. In a spherical coordinate system, arrays *x*, *y*, and *z* represent θ), ϕ, and r coordinates, respectively.

For a 3D grid, the ordering of the z data is important. For calculation of the z values, the *x* value is held constant while *y* is cycled through its range of values. The *x* value is then incremented and *y* is cycled again. This is repeated until all the data is calculated. So, for a 10x20 grid the data shall be ordered as follows:

x1	y1	z1
x1	y2	z2
.		
.		
.		
x1	y19	z19
x1	y20	z20
x2	y1	z21
x2	y2	z22
.		
.		
.		
x2	y19	z29
x2	y20	z30
x3	y1	z31
x3	y2	z32
.		
.		
.		
x10	y18	z198
x10	y19	z199
x10	y20	z200

A 3D-plot in Figure 2.3 is produced by Program 2.3. Unlike Program 2.2, the number of elements (600) for array z in Program 2.3 is the product of the number of elements (20) for array x and that (30) for array y. The color box with the gradient of the smooth color between the maximum and minimum values of the color palette for a 3D plot can be removed by the member function **CPlot**::**colorBox**().

The data for plotting can also be stored in a file first and then obtained by function

```
int CPlot::dataFile(char *filename);
int CPlot::dataFile(char *filename, char *option);
```

Each data file corresponds to a single data set. The data file should be formatted with each data point on a separate line. 2D data is specified by two values per point. An empty line in a 2D data file causes a break of the curve in the plot. Multiple curves can be plotted in this manner, however the plot style will be the same for all curves. For example, Program 2.4 will generate a plot shown in Figure 2.1.

3D data is specified by three values per data point. For 3D grid or surface data, each row is separated in the data file by a blank line. For example, a 3 x 3 grid would be represented as follows:

```
#include <chplot.h>
#include <math.h>

#define NUMX 20
#define NUMY 30
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double r;
    int i, j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
      x[i] = -10 + i*20.0/(NUMX-1);  // linspace(x, -10, 10)
    }
    for(i=0; i<NUMY; i++) {
      y[i] = -10 + i*20.0/(NUMY-1);  // linspace(y, -10, 10)
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[30*i+j] = sin(r)/r;
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.plotting();
    return 0;
}
```

Program 2.3: A plotting program for a 3D grid.

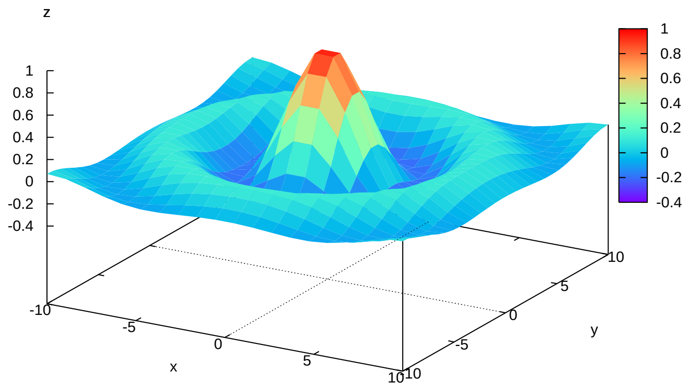


Figure 2.3: A plot with a 3D grid.

```
#include <chplot.h>
#include <math.h>

int main() {
    char *filename;
    int i;
    class CPlot plot;
    FILE *out;

    filename = tmpnam(NULL);       //Create a temporary file.
    out=fopen (filename,"w");      //Write data to the file.
    for (i=-180;i<=180;i++)
       fprintf(out,"%i %f \n",i,sin(i*M_PI/180));
    fclose(out);
    plot.dataFile(filename);
    plot.plotting();
    remove(filename);
    return 0;
}
```

Program 2.4: A plotting program using data from a file.

```
x1  y1  z1
x1  y2  z2
x1  y3  z3

x2  y1  z4
x2  y2  z5
x2  y3  z6

x3  y1  z7
x3  y2  z8
x3  y3  z9
```

Two empty lines in the data file will cause a break in the plot. Multiple curves or surfaces can be plotted in this manner, however, the plot style will be the same for all curves or surfaces. Member function **CPlot**::**dimension**() with the value of 3 as the argument must be called before a 3D data file can be added.

2.1.2 Annotations

A plot can be annotated with a title and labels on axes using corresponding member functions

```
void CPlot::title(char *title);
```

and

```
void CPlot::label(int axis, char *label);
```

respectively. The argument *axis* of member function **CPlot**::**label**() is the axis to be set. The valid macros for *axis* are listed in Table 2.1. Figure 2.4 displays the plot produced by Program 2.5 using member functions **CPlot**::**title**() and **CPlot**::**label**(). By default, no title is displayed and the coordinate axes are labeled with symbols x, y, and z.

Table 2.1: The macros for axes.

PLOT_AXIS_X	Select the x axis only.
PLOT_AXIS_Y	Select the y axis only.
PLOT_AXIS_Z	Select the z axis only.
PLOT_AXIS_XY	Select the x and y axes.
PLOT_AXIS_XYZ	Select the x, y, and z axes.

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;
    char *title="Sine Wave",
         *xlabel="degree",
         *ylabel="amplitude";

    for(i=0; i<NUM; i++) {
      x[i] = 0+ i*360.0/(NUM-1);  // linspace(x, 0, 360)
      y[i] = sin(x[i]*M_PI/180);
    }
    plot.data2DCurve(x, y, NUM);
    plot.title("Sine Wave");
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.plotting();
    return 0;
}
```

Program 2.5: A plotting program with annotation.

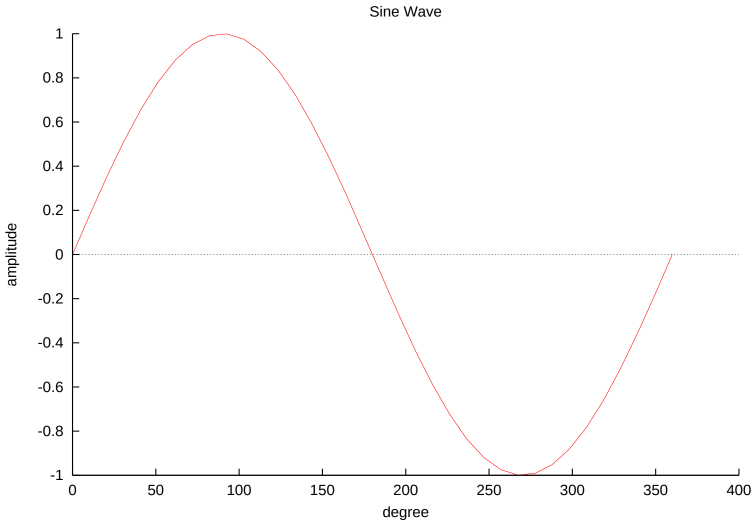


Figure 2.4: A plot with annotation.

Table 2.2: The macros for border locations.

PLOT_BORDER_BOTTOM	The bottom of the plot.
PLOT_BORDER_LEFT	The left side of the plot.
PLOT_BORDER_TOP	The top of the plot.
PLOT_BORDER_RIGHT	The right side of the plot.
PLOT_BORDER_ALL	All sides of the plot.

Program 2.6 demonstrates how arrow, text, axis limits, grid, border, and axis are handled using member functions of **CPlot** class. Figure 2.5 displays the plot produced by Program 2.6. In Program 2.6, the axis limits are set by member function **CPlot**::**axisRange**(),

```
void CPlot::axisRange(int axis, double minumum, double maximum,
                      double incr);
```

The valid macros for *axis* are listed in Table 2.1. The minumum and maximum values for an axis are given in second and third arguments, respectively. The tic marks on an axis can be set by the function

```
void ticsRange(int axis, double incr);
void ticsRange(int axis, double incr, double start);
void ticsRange(int axis, double incr, double start, double end);
```

The increment between tic marks is given in *incr*. By default, this value is calculated internally. The start and end positions for tic marks are optional arguments. By default, this value is calculated internally. For example, function calls

```
plot.axisRange(PLOT_AXIS_X, 0, 360);
plot.ticsRange(PLOT_AXIS_X, 30, 0, 360);
```

set the range of the `x`-axis from 0 to 360 degrees with tics marks at every 30 degrees. Drawing the `x` and `y` axes on a 2D plot can be enabled or disabled using member function

```
void CPlot::axis(int axis, int flag);
```

The valid macros for *axis* are the same as those for other member functions. The *flag* can be set to **PLOT_ON** to enable the drawing of the specified axis, or **PLOT_OFF** to disable the drawing of the specified axis. In Program 2.5, the drawing of `x` and `y` axes is disabled at the same time by using function call `plot.axis(PLOT_AXIS_XY, PLOT_OFF)`. Member function

```
void CPlot::boarder(int location, int flag);
```

turns a border display around the plot on or off. By default, the border is drawn on the left and bottom sides for 2D plots, and on all sides on the x-y plane for 3D plots. The valid *location* for function **CPlot**::**border**() is given in Table 2.2. Figure 2.5 is generated with borders on four sides by function call `CPlot::border(PLOT_BORDER_ALL, PLOT_ON).` The display of a grid on the x-y plane can be enabled or disabled by member function

```
void CPlot::grid(int flag);
void CPlot::grid(int flag, char *option);
```

The *flag* can be set to **PLOT_ON** to enable or **PLOT_OFF** to disable the display of the grid. For a polar plot, a polar grid will be drawn. Otherwise, the grid is rectangular. A plot can be annotated with arrows by function

Table 2.3: The macros for text locations.

PLOT_TEXT_LEFT	The left side of the text string.
PLOT_TEXT_RIGHT	The right side of the text string.
PLOT_TEXT_CENTER	The center of the text string.

```
    void CPlot::arrow(double x_head, double y_head, double z_head,
                      double x_tail, double y_tail, double z_tail);
    void CPlot::arrow(double x_head, double y_head, double z_head,
                      double x_tail, double y_tail, double z_tail,
                      char *option);
```

where (*x_head, y_head, z_head*) and (*x_tail, y_tail, z_tail*) are the coordinates of the head and tail of an arrow, respectively. The arrow points from (*x_tail, y_tail, z_tail*) to (*x_head, y_head, z_head*). These coordinates are specified using the same coordinate system as the curves of the plot. An optional argument can be used to specify other attributes of an arrow. The annotation of text on a plot is achieved by member function

```
  void CPlot::text(char *string, int just, double x, double y, double z);
```

where text *string* is placed at location (*x*,*y*) for 2D plots or (*x*,*y*,*z*) for 3D plots. The location of the text is measured in the plot coordinate system. The position of the text is adjusted by the argument *just*. The valid macros for argument *just* are given in Table 2.3. In Figure 2.5, the tail of the arrow is the location for the text `testing text` left adjusted using functions **CPlot**::**arrow**() and **CPlot**::**text**().

Additional features such as different tics marks and scales for data can be found in the reference for **CPlot** class.

2.1.3 Multiple Data Sets and Legends

A plot with multiple sets of data can be produced as shown in Figure 2.7. Figure 2.7 with legends can be generated by Program 2.7. A string of *legend* can be added to the plot by member function

```
    void CPlot::legend(char *legend, int num);
```

The number of data set to which the legend is added is indicated by the second argument *num*. Numbering of the data sets starts with zero. New legends will replace previously specified legends. This member function shall be called after plotting data have been added by member functions **CPlot**::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), or **CPlot**::**dataFile**(). The member function

```
    void CPlot::legendLocation(double x, double y, double z);
```

specifies the position of the plot legend using plot coordinates *(x, y, z)*. The position specified is the location of the top right of the box for the markers and labels of the legend as shown in Figure 2.6. By default, the location of the legend is near the upper-right corner of the plot.

A 3D plot with multiple sets of data can be produced similarly. The 3D plot in Figure 2.8 can be generated by 2.8, in which member function **CPlot**::**data3DSurface**() for adding the data to the plot is called twice. Program 2.9 demonstrates how to superimpose a curve on a surface with output shown in Figure 2.9. Because no hidden lines can be removed from non grid data, the hidden line removal is turned off by member function **CPlot**::**removeHiddenLine**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;
    char *title="Sine Wave",
         *xlabel="degree",
         *ylabel="amplitude";
    double x1=180, y1=0.02, z1=0;
    double x2=225, y2=0.1,  z2=0;

    for(i=0; i<NUM; i++) {
      x[i] = 0+ i*360.0/(NUM-1);  // linspace(x, 0, 360)
      y[i] = sin(x[i]*M_PI/180);
    }
    plot.data2DCurve(x, y, NUM);
    plot.title("Sine Wave");
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.axisRange(PLOT_AXIS_X, 0, 360);
    plot.ticsRange(PLOT_AXIS_X, 30, 0, 360);
    plot.axisRange(PLOT_AXIS_Y, -1, 1);
    plot.ticsRange(PLOT_AXIS_Y, .25, -1, 1);
    plot.axis(PLOT_AXIS_XY, PLOT_OFF);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.grid(PLOT_ON);
    plot.arrow(x1, y1, z1, x2, y2, z2);
    plot.text("testing text", PLOT_TEXT_LEFT, x2, y2, z2);
    plot.plotting();
    return 0;
}
```

Program 2.6: A plotting program with many features.

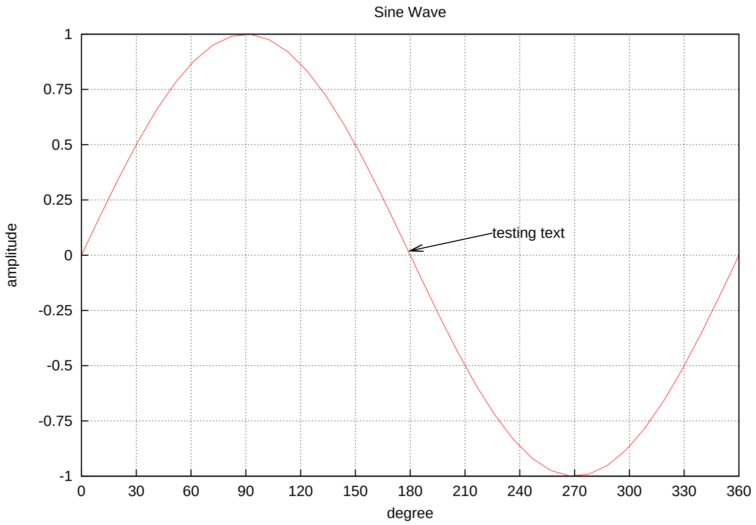


Figure 2.5: A plot with many features.

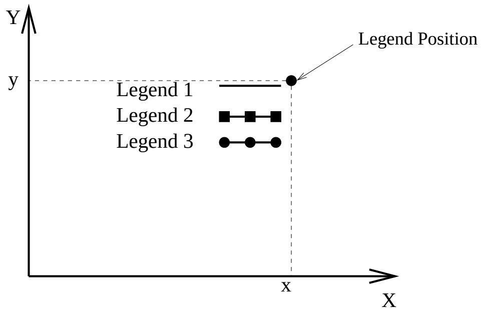


Figure 2.6: The position for legend.

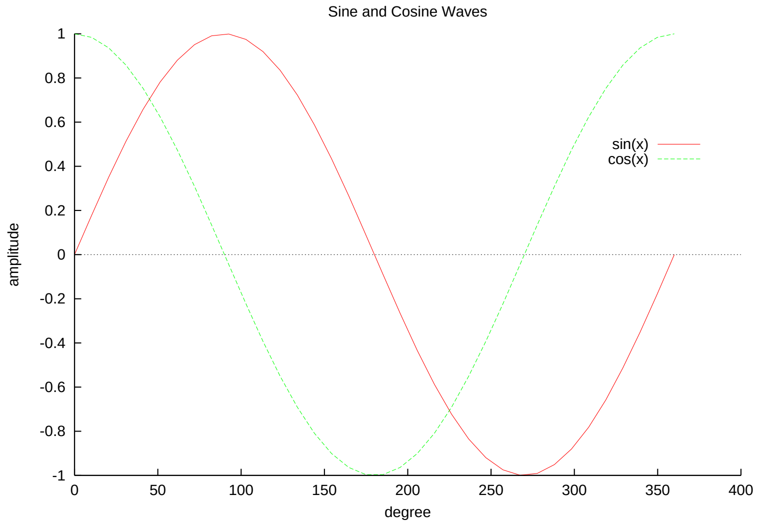


Figure 2.7: A plot with two sets of data, title, labels, and legends.

```
#include<math.h>
#include<chplot.h>

#define NUM 36
int main() {
  int i;
  double x1[NUM], y1[NUM];
  double x2[NUM], y2[NUM];
  char *title="Sine and Cosine Waves",
       *xlabel="degree",
       *ylabel="amplitude";
  class CPlot plot;

  for(i=0; i<NUM; i++) {
      x1[i] = 0+ i*360.0/(NUM-1);  // linspace(x1, 0, 360)
      x2[i] = 0+ i*360.0/(NUM-1);  // linspace(x2, 0, 360)
      y1[i] = sin(x1[i]*M_PI/180);
      y2[i] = cos(x2[i]*M_PI/180);
  }
  plot.data2DCurve(x1, y1, NUM);
  plot.data2DCurve(x2, y2, NUM);
  plot.legend("sin(x)", 0);
  plot.legend("cos(x)", 1);
  plot.legendLocation(350, 0.5, 0);
  plot.title(title);
  plot.label(PLOT_AXIS_X, xlabel);
  plot.label(PLOT_AXIS_Y, ylabel);
  plot.plotting();
  return 0;
}
```

Program 2.7: A program for plotting two functions on the same plot with title, labels, and legends.

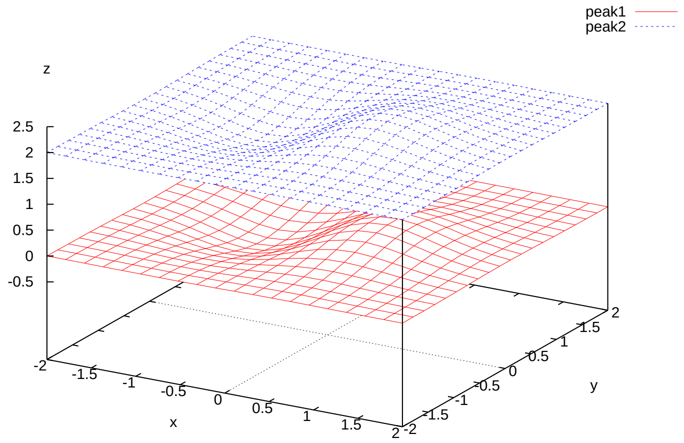


Figure 2.8: A 3D plot with two sets of data.

```
#include <chplot.h>
#include <math.h>

#define NUMX 20
#define NUMY 20
int main() {
    double x[NUMX], y[NUMY], z1[NUMX*NUMY], z2[NUMX*NUMY];
    int datasetnum =0, i, j;
    int line_type = 1, line_width = 1;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
      x[i] = -2 + i*4.0/(NUMX-1);  // linspace(x, -2, 2)
    }
    for(i=0; i<NUMY; i++) {
      y[i] = -2 + i*4.0/(NUMY-1);  // linspace(y, -2, 2)
    }
    for (i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z1[i*NUMX+j] = x[i]*exp(-x[i]*x[i]-y[j]*y[j]);
            z2[i*NUMX+j] = z1[i*NUMX+j] +2;
        }
    }
    plot.data3DSurface(x, y, z1, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum++, line_type++, line_width);
    plot.data3DSurface(x, y, z2, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum++, line_type, line_width);
    plot.legend("peak1", 0);
    plot.legend("peak2", 1);
    plot.plotting();
    return 0;
}
```

Program 2.8: A program for plotting two functions on the same 3D plot.

```
#include <chplot.h>
#include <math.h>

#define NUMX 20
#define NUMY 20
#define NUM  20
int main() {
    double x[NUMX], y[NUMY], z1[NUMX*NUMY], z2[NUMX*NUMY];
    double x0[NUM], y0[NUM], z0[NUM];
    int datasetnum=0, i, j;
    int line_type = 1, line_width = 1;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
      x[i] = -2 + i*4.0/(NUMX-1);   // linspace(x, -2, 2)
    }
    for(i=0; i<NUMY; i++) {
      y[i] = -2 + i*4.0/(NUMY-1);   // linspace(y, -2, 2)
    }
    for(i=0; i<NUM; i++) {
      x0[i] = -2 + i*4.0/(NUM-1);   // linspace(x0, -2, 2)
      y0[i] = -1;
    }
    for (i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z1[i*NUMY+j] = x[i]*exp(-x[i]*x[i]-y[j]*y[j]);
            z2[i*NUMY+j] = z1[i*NUMY+j] +2;
        }
    }
    for (i=0; i<NUM; i++)
        z0[i] = x0[i]*exp(-x0[i]*x0[i]-y0[i]*y0[i]);
    plot.data3DSurface(x, y, z1, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum++, line_type++, line_width);
    plot.data3DSurface(x, y, z2, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum++, line_type++, line_width);
    plot.data3DCurve(x0, y0, z0, NUM);
    line_type = 5;
    line_width = 2;
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum++, line_type++, line_width);
    plot.legend("peak1", 0);
    plot.legend("peak2", 1);
    plot.legend("curve", 2);
    plot.removeHiddenLine(PLOT_OFF);
    plot.plotting();
    return 0;
}
```

Program 2.9: A program superimposing a curve on a surface.

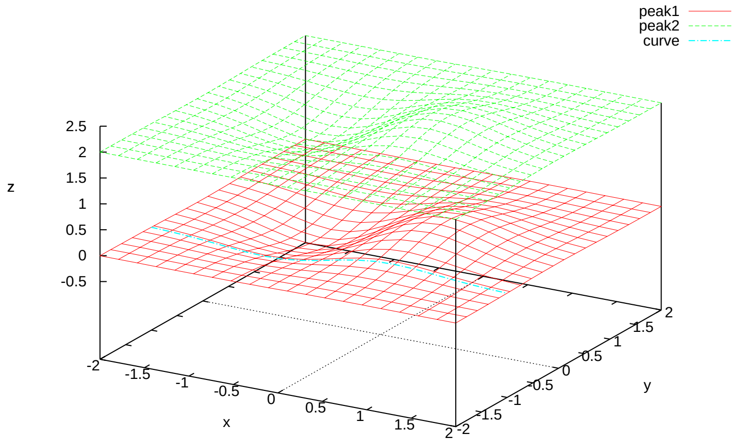


Figure 2.9: A 3D plot with a curve superimposed on a surface.

2.1.4 Subplots

Multiple plots can be displayed in the same figure and printed on the same piece of paper using function

```
int CPlot::subplot(int row, int col);
```

Function **CPlot**::**subplot**() breaks the figure into an `m`-by-`n` matrix of small subplots. The subplots are numbered as if there were a 2-dimensional matrix with the numbers of rows and columns specified in its arguments. Each index starts with 0. A pointer to **CPlot** class as a handle for subplot at location `(i,j)` can be obtained by function

```
class CPlot* CPlot::getSubplot(int row, int col);
```

where *row* and *col* are the row and column numbers of the desired subplot element, respectively. Numbering starts with zero. For example, Program 2.10 breaks a plot with four subplots in a 2-by-2 matrix. Each subplot can be annotated with title, label, etc. as if it were a separate plot. Figure 2.10 displays the plot produced by Program 2.10. In Program 2.10, to avoid division by zero, a small floating-point value `DBL_EPSILON` is used for function $\sin(x)/(x)$.

2.1.5 Export Plots

A plot can not only be displayed in a terminal screen, but can also be exported in a variety of formats for various applications. Different output types can be achieved by member function

```
void outputType(int outputtype);
void outputType(int outputtype, char *terminal);
void outputType(int outputtype, char *terminal, char *filename);
```

The argument *outputtype* can be one of the following macros **PLOT_OUTPUTTYPE_DISPLAY**, **PLOT_OUTPUTTYPE_STREAM**, and **PLOT_OUTPUTTYPE_FILE**. The output type **PLOT_OUTPUTTYPE_DISPLAY** displays the plot on the screen. The plot is displayed in its own separate window. A plot window can be closed by pressing the ‘q’ key in Unix. By default, the output type is **PLOT_OUTPUTTYPE_DISPLAY**. For the output type **PLOT_OUTPUTTYPE_STREAM**, the output

```
#include <float.h>
#include <math.h>
#include <chplot.h>

#define NUM1 36
#define NUM2 101
#define NUMX 20
#define NUMY 30
int main() {
    double x[NUM1], y[NUM1];
    double x3[NUMX], y3[NUMY], z3[NUMX*NUMY], r;
    double x4[NUM2], y4[NUM2];
    int i, j;
    class CPlot subplot, *plot;

    for(i=0; i<NUM1; i++) {
      x[i] = -M_PI + i*2*M_PI/(NUM1-1);  // linspace(x1, -PI, PI)
      y[i] = sin(x[i]);
    }
    subplot.subplot(2, 2);
    plot = subplot.getSubplot(0, 0);
    plot->data2DCurve(x, y, NUM1);

    plot = subplot.getSubplot(0, 1);
    plot->data2DCurve(x, y, NUM1);
    plot->axisRange(PLOT_AXIS_Y, -1, 1);
    plot->ticsRange(PLOT_AXIS_Y, 0.25, -1, 1);
    plot->grid(PLOT_ON);

    for(i=0; i<NUM2; i++) {
      x4[i] = -20 + i*40.0/(NUM2-1);  // linspace(x4, -20, 20)
      if(x4[i] == 0.0)
        x4[i] = DBL_EPSILON; /* x4[0] becomes epsilon */
      y4[i] = sin(x4[i])/(x4[i]);
    }
    plot = subplot.getSubplot(1, 0);
    plot->data2DCurve(x4, y4, NUM2);
    plot->label(PLOT_AXIS_Y, "sin(x)/x");

    for(i=0; i<NUMX; i++) {
      x3[i] = -10 + i*20.0/(NUMX-1);  // linspace(x3, -10, 10)
    }
    for(i=0; i<NUMY; i++) {
      y3[i] = -10 + i*20.0/(NUMY-1);  // linspace(y3, -10, 10)
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            r = sqrt(x3[i]*x3[i]+y3[j]*y3[j]);
            z3[NUMY*i+j] = sin(r)/r;
        }
    }
    plot = subplot.getSubplot(1, 1);
    plot->data3DSurface(x3, y3, z3, NUMX, NUMY);
    plot->colorBox(PLOT_OFF);

    subplot.plotting();
    return 0;
}
```

Program 2.10: A plotting program with subplots.

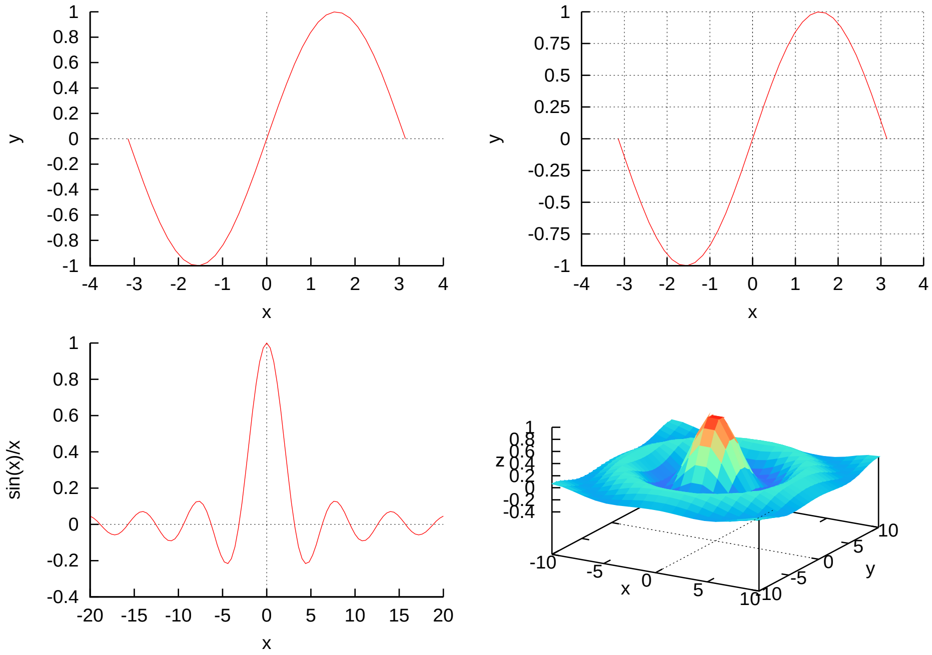


Figure 2.10: A plot with subplots.

from the plot engine is a standard output stream. This output type is useful when a Ch program is used as a CGI script in a Web server to generate a plot dynamically as a standard output stream in a png file format in Web browser displays. For **PLOT_OUTPUTTYPE_FILE**, a plot can be saved in files in a variety of different formats that can be controlled by two optional arguments *terminal* and *filename*. The supported terminal types are listed in Table 2.4. Some terminals can have additional parameters such as size and color of a plot as part of the string for the argument *terminal*. Details for each terminal are given in the reference for **CPlot** class. The last optional argument *filename* is a string containing a file name to which the plot is saved. On machines that support pipes, the output can also be piped to another program by placing the '`|`' character in front of the command name and using it as the *filename*. For example, on Unix systems, setting *terminal* to "`postscript`" and *filename* to "`|lp`" could be used to send a plot directly to a postscript printer.

Program 2.11 shows how to export a plot in the formats of encapsulated postscript, latex, pbm, and png formats.

2.1.6 Print Plots

2.1.6.1 Printing Plots in Windows

One of the following two methods can be used to print a plot in Windows.
Method 1
Step 1. Run a Ch program with plotting, click the upper left corner of the window with plot.
Step 2. Select "print" from the options menu, configure, and print accordingly.

Method 2
Step 1. Run a Ch program with plotting, click the upper left corner of the window with plot.
Step 2. Select "copy to clipboard" from the options menu.
Step 3. Open Painbrush in **Start**− >**Accessories**.

Table 2.4: The terminal types of plot output type.

Terminal	Description
aifm	Adobe Illustrator 3.0.
corel	EPS format for CorelDRAW.
dxf	AutoCAD DXF.
dxy800a	Roland DXY800A plotter.
eepic	Extended LaTeXpicture.
emtex	LaTeXpicture with emTeX specials.
epson-180dpi	Epson LQ-style 24-pin printer with 180dpi.
epson-60dpi	Epson LQ-style 24-pin printers with 60dpi.
epson-lx800	Epson LX-800, Star NL-10 and NX-100.
excl	Talaris printers.
fig	Xfig 3.1.
gpic	gpic/groff package.
hp2648	Hewlett Packard HP2647 an HP2648.
hp500c	Hewlett Packard DeskJet 500c.
hpdj	Hewlett Packard DeskJet 500.
hpgl	HPGL output.
hpljii	HP LaserJet II.
hppj	HP PaintJet and HP3630 printers.
latex	LaTeXpicture.
mf	MetaFont.
mif	Frame Maker MIF 3.00.
nec-cp6	NEC CP6 and Epson LQ-800.
okidata	9-pin OKIDATA 320/321 printers.
pcl5	Hewlett Packard LaserJet III.
pbm	Portable BitMap.
png	Portable Network Graphics.
postscript	Postscript.
pslatex	LaTeXpicture with postscript specials.
pstricks	LaTeXpicture with PSTricks macros.
starc	Star Color Printer.
tandy-60dpi	Tandy DMP-130 series printers.
texdraw	LaTeXtexdraw format.
tgif	TGIF X-Window drawing format.
tpic	LaTeXpicture with tpic specials.

```
#include<math.h>
#include<chplot.h>

#define NUM 36
int main() {
  int i;
  double x[NUM], y[NUM];
  char *title="Sine Wave",              // Define labels.
       *xlabel="degree",
       *ylabel="amplitude";
  class CPlot plot;

  for(i=0; i<NUM; i++) {
      x[i] = 0+ i*360.0/(NUM-1);  // linspace(x, 0, 360)
      y[i] = sin(x[i]*M_PI/180);
  }
  plot.data2DCurve(x, y, NUM);
  plot.title(title);
  plot.label(PLOT_AXIS_X, xlabel);
  plot.label(PLOT_AXIS_Y, ylabel);

  /* create a postscript file */
  plot.outputType(PLOT_OUTPUTTYPE_FILE, "postscript eps color", "demo.eps");
  plot.plotting();

  /* create a latex file */
  plot.outputType(PLOT_OUTPUTTYPE_FILE, "latex roman 11", "demo.tex");
  plot.plotting();

  /* create a pbm file */
  plot.outputType(PLOT_OUTPUTTYPE_FILE, "pbm", "demo.pbm");
  plot.plotting();

  /* create a png file */
  plot.outputType(PLOT_OUTPUTTYPE_FILE, "png", "demo.png");
  plot.plotting();
  printf("Files demo.eps, demo.tex, demo.pbm, demo.png were created\n");
  return 0;
}
```

Program 2.11: A program for exporting plot.

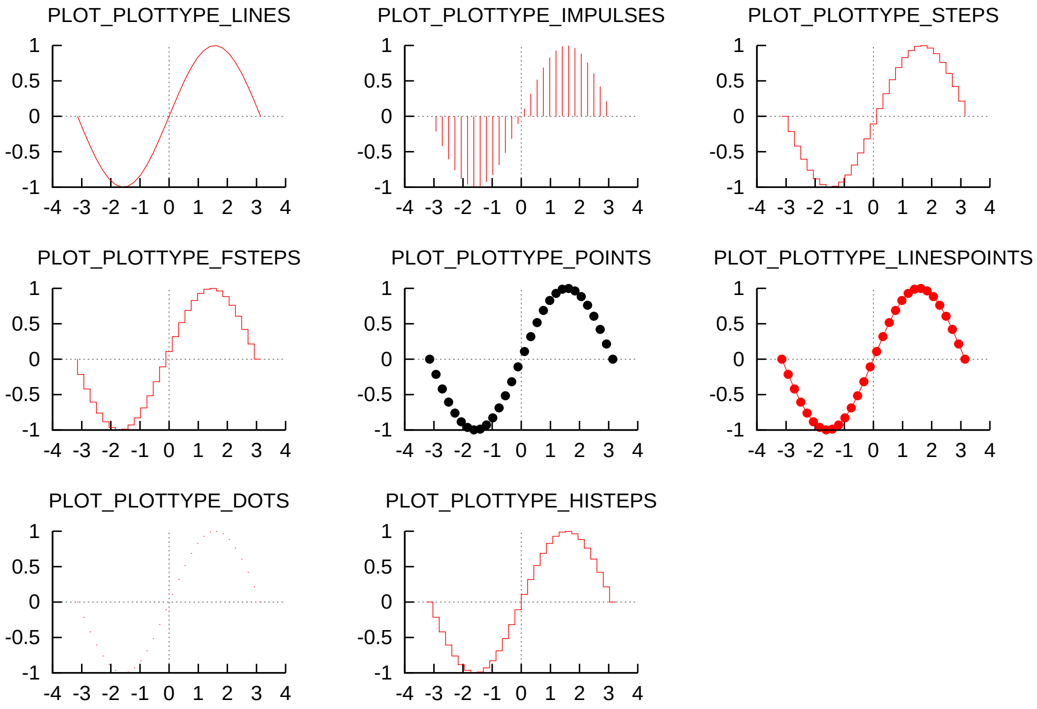


Figure 2.11: Two-dimensional plot types.

Step 4. Paste the plot by clicking “paste” from the edit menu or using the key combination `<Ctrl><V>`.
Step 5. Save the plot as a bmp file.
Step 6. Print the plot.

2.1.6.2 Printing Plots in Unix

In Unix, first, one can save a plot in a file according to the terminal type described in the previous section. Then, print it out. For example, for a postscript printer, a plot can first be saved as a color encapsulated postscript file named **filename.eps** by function `plot.outputType(PLOT_OUTPUTTYPE_FILE, "postscript eps color", "filename.eps").` The postscript file **filename.eps** can then be printed out by command **lp**. Alternatively, the plot can be printed out by setting the output type of the plot using function call `plot.outputType(PLOT_OUTPUTTYPE_FILE, "postscript eps color", "| lp").`

2.2 2D Plotting

The features presented in the previous sections can be applied to both 2D and 3D plotting. This section will describe features specific to 2D plotting only.

2.2.1 Plot Types, Line Styles, and Markers

Different plot types can be selected by function

```
  void plotType(int plot_type, int num);
```

```
void plotType(int plot_type, int num, char * option);
```

Function **CPlot**::**plotType**() sets the desired plot type for a data set to be plotted. The valid macros for argument *plot_type* are given in Table 2.5 with some corresponding plots shown in Figure 2.11. By default, a 2D plot uses plot type **PLOT_PLOTTYPE_LINES**. Data sets in the same plot can have different plot types. The argument *num* indicates the data set to which the plot type is applied. Numbering of the data sets starts with zero. New plot types replace previously specified types.

The line type, width, and color for lines, impulses, steps, etc. can be set by function

```
void lineType(int num, int line_type, int line_width);
void lineType(int num, int line_type, int line_width, char *line_color);
```

Function **CPlot**::**lineType**() sets the desired line style for a data set to be plotted. The line style and/or marker type for the plot are selected automatically. The *line_type* specifies an index for the line type used for drawing the line. The line type varies depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line when the plot is display, Changing the line type makes it dashed, dotted, or other shape when the plot is saved as a postscript file. All terminals support at least six different line types. By default, the line type is 1. The *line_width* specifies the line width. The line width is *line_width* multiplied by the default width. Typically the default width is one pixel. An optional fourth argument can specify the color of a line by a color name or RGB value, such as `"blue"` or `"#0000ff"` for color blue.

Program 2.12 illustrates how different line types are used. The plot displayed in Windows is shown in Figure 2.12. Line type is typically associated with a color. Program 2.13 illustrate how to specify colors of lines inside a program. Figure 2.13 shows the generated plot in the postscript file format.

Program 2.14 illustrates how to generate multiple plots using the same instance of a class. When Program 2.14 is executed, a plot with a red curve is first displayed. Next, a plot with blue curve is displayed. Then, the color of the curve is changed by overlaying another curve with red color. The last color dominates. The last color can be determined dynamically by a program at execution time. Finally, a new curve with color of magenta is added to the plot. Four separate plots generated by Program 2.14 are displayed in Figure 2.14.

The point type, size , and color points can be set by function

```
void pointType(int num, int point_type, int point_size);
void pointType(int num, int point_type, int point_size,
               char *point_color);
```

Function **CPlot**::**pointType**() sets the desired point style for a data set to be plotted. The *point_type* specifies an index for the point type used for drawing the point. The point type varies depending on the terminal type used (see **CPlot**::**outputType**). The value *point_type* is used to change the appearance (color and/or marker type) of a point. It is specified with an integer representing the index of the desired point type. All terminals support at least six different point types. *point_size* is an optional argument used to change the size of the point. The point size is *point_size* multiplied by the default size. If *point_type* and *point_size* are set to zero or a negative number, a default value is used. An optional fourth argument can specify the color of a point by a color name or RGB value, such as `"blue"` or `"#0000ff"` for color blue.

Program 2.15 illustrates how different point types are used. The plot displayed in Windows is shown in Figure 2.15. In Figure 2.16, two sets of data, one in line type and the other in point type, are displayed in the same plot. The source code generating this figure is listed in Program 2.16.

Table 2.5: The macros for 2D plot types.

PLOT_PLOTTYPE_BOXERRORBARS	It is a combination of the **PLOT_PLOTTYPE_BOXES** and **PLOT_PLOTTYPE_YERRORBARS** plot types.
PLOT_PLOTTYPE_BOXES	Draw a box centered about the given x coordinate.
PLOT_PLOTTYPE_BOXXYERRORBARS	A combination of **PLOT_PLOTTYPE_BOXES** and **PLOT_PLOTTYPE_XYERRORBARS** plot types.
PLOT_PLOTTYPE_CANDLESTICKS	Display box-and-whisker plots of financial or statistical data.
PLOT_PLOTTYPE_DOTS	Use dots to mark each data point.
PLOT_PLOTTYPE_FILLEDCURVES	Fill an area bounded by a curve with a solid color or pattern.
PLOT_PLOTTYPE_FINANCEBARS	Display finanacial data.
PLOT_PLOTTYPE_FSTEPS	Adjacent points are connected with two line segments, one from (x1,y1) to (x1,y2), and a second from (x1,y2) to (x2,y2).
PLOT_PLOTTYPE_HISTEPS	The point x1 is represented by a horizontal line from ((x0+x1)/2,y1) to ((x1+x2)/2,y1). Adjacent lines are connected with a vertical line from ((x1+x2)/2,y1) to ((x1+x2)/2,y2).
PLOT_PLOTTYPE_IMPULSES	Display vertical lines from the x-axis (for 2D plots) or the x-y plane (for 3D plots) to the data points.
PLOT_PLOTTYPE_LINES	Data points are connected with a line.
PLOT_PLOTTYPE_LINESPOINTS	Markers are displayed at each data point and connected with a line.
PLOT_PLOTTYPE_POINTS	Markers are displayed at each data point.
PLOT_PLOTTYPE_STEPS	Adjacent points are connected with two line segments, one from (x1,y1) to (x2,y1), and a second from (x2,y1) to (x2,y2).
PLOT_PLOTTYPE_VECTORS	Display vectors.
PLOT_PLOTTYPE_XERRORBARS	Display dots with horizontal error bars.
PLOT_PLOTTYPE_XERRORLINES	Display linepoints with horizontal error lines.
PLOT_PLOTTYPE_XYERRORBARS	Display dots with horizontal and vertical error bars.
PLOT_PLOTTYPE_XYERRORLINES	Display linepoints with horizontal and vertical error lines.
PLOT_PLOTTYPE_YERRORBARS	Display points with vertical error bars.
PLOT_PLOTTYPE_YERRORLINES	Display linepoints with vertical error lines.

```
#include <chplot.h>

int main() {
    double x, y, xx[2], yy[2];
    char text[10];
    int line_type = -1, line_width = 2, datasetnum = 0;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_X, 0, 5);
    plot.axisRange(PLOT_AXIS_Y, 0, 4);
    plot.ticsRange(PLOT_AXIS_Y, 1, 0, 4);
    plot.title("Line Types in Ch Plot");
    for (y = 3; y >= 1; y--) {
        for (x = 1; x <= 4; x++) {
            sprintf(text, "%d", line_type);
            xx[0] = x; xx[1] = x+0.5;
            yy[0] = y; yy[1] = y;
            plot.data2DCurve(xx, yy, 2);
            plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
            plot.lineType(datasetnum, line_type, line_width);
            plot.text(text, PLOT_TEXT_RIGHT, x-.125, y, 0);
            datasetnum++;
            line_type++;
         }
    }
    plot.plotting();
    return 0;
}
```

Program 2.12: A plotting program for different line types.

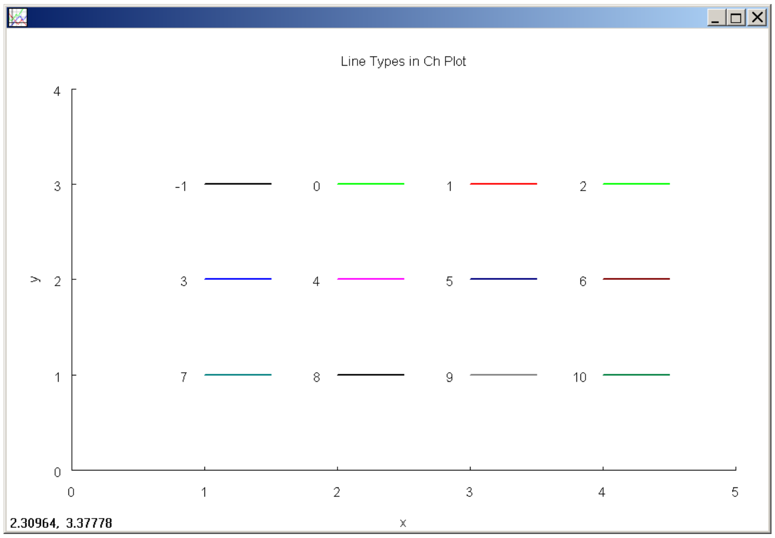


Figure 2.12: A plot with different line types displayed in Windows.

```
/* File: color3.ch */
#include <chplot.h>

/* colors of lines for displayed plot */
char *color[] = {
  "black",
  "white",
  "grey",
  "grey40",
  "grey60",
  "red",
  "yellow",
  "green",
  "blue",
  "navy",
  "cyan",
  "magenta",
  "orange",
  "gold",
  "brown",
  "purple",
};


int main() {
    double x[2], y[2];
    int i, line_type= 1, line_width = 1, datasetnum = 0, n;
    CPlot plot;

    plot.title("Line Colors in Ch Plot");
    n = sizeof(color)/sizeof(color[0]);
    y[0] = 0; y[1] = 1;
    for (i = 0; i < n; i++) {
       x[0] = i+1; x[1] = i+1;
       plot.data2DCurve(x, y, 2);
       plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
       plot.lineType(datasetnum, line_type, line_width, color[i]);
       datasetnum++;
    }
    /* color of the horizontal line added below is green */
    x[0] = 1; x[1] = 15;
    y[0] = 0.5; y[1] = .5;
    plot.data2DCurve(x, y, 2);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width, "green");
    plot.plotting();
}
```

Program 2.13: Specify colors of curves inside a program.

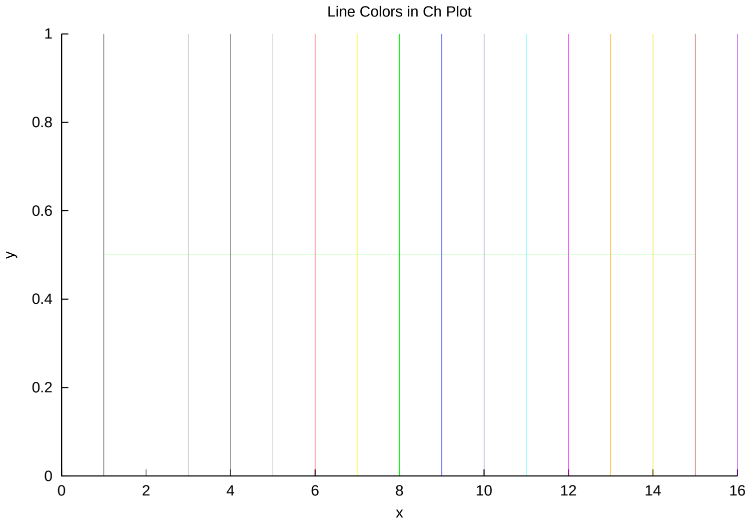


Figure 2.13: A plot with colors specified inside a program saved as a postscript file.

2.2.2 Polar Plot

A 2D plot in a polar coordinate system can be specified using member function

```
void CPlot::polarPlot(int angle_unit);
```

The argument *angle_unit* specifies the unit for measurement of angular positions. It can be one of the following macros **PLOT_ANGLE_DEG** for angles measured in degree and **PLOT_ANGLE_RAD** in radian. As shown in Program 2.17, in a polar coordinate system (θ, r), the first and second array arguments in member function call of `plot.data2D(theta, r)` are the phase angle and magnitude of points to be plotted, respectively. The polar grid displayed in Figure 2.17 is achieved by a function call of `plot.grid(PLOT_ON)`. The aspect ratio of a plot can be set by member function

```
void CPlot::sizeRatio(float ratio);
```

The meaning of *ratio* changes depending on its value. A positive *ratio* is the ratio of the length of the y-axis to the length of the x-axis. So, if *ratio* is 2, the y-axis will be twice as long as the x-axis. If *ratio* is zero, the default aspect ratio for the terminal type is used. A negative *ratio* is the ratio of the y-axis units to the x-axis units. So, if *ratio* is -2, one unit on the y-axis will be twice as long as one unit on the x-axis. In case of a polar plot, the aspect ratio should be set to 1 as shown in Program2.17.

2.3 3D Plotting

This section describes features applicable to 3D plotting only.

2.3.1 Plot Types

Like in 2D, the plot type in 3D can also be specified by member function **CPlot**::**plotType**(). The valid macros for plot type are listed in Table 2.6 with corresponding plot types displayed in Figure 2.18. By default, the plot type **PLOT_PLOTTYPE_LINES** is used in 3D plotting.

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    double x[NUM], y[NUM], y2[NUM];
    int i, line_type = 1, line_width = 1, datasetnum = 0;
    class CPlot plot;

    for(i=0; i<NUM; i++) {
      x[i] = -M_PI + i*2*M_PI/(NUM-1);  // lindata(-M_PI, M_PI, x);
      y[i] = sin(x[i]);
      y2[i] = sin(x[i])+0.5;
    }
    plot.data2DCurve(x, y, NUM);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width, "red");
    plot.legend("red line", datasetnum);
    plot.plotting();

    /* change the color of the curve from the same data set to blue  */
    plot.lineType(datasetnum, line_type, line_width, "blue");
    plot.legend("blue line", datasetnum);
    plot.plotting();

    /* overlaying blue curve with red curve .*/
    plot.data2DCurve(x, y, NUM);
    datasetnum++;
    plot.lineType(datasetnum, line_type, line_width, "red");
    plot.legend("red line", datasetnum);
    plot.plotting();

    /* add a new curve with color of magenta to the plot */
    plot.data2DCurve(x, y2, NUM);
    datasetnum++;
    plot.lineType(datasetnum, line_type, line_width, "magenta");
    plot.legend("magenta line", datasetnum);
    plot.plotting();
}
```

Program 2.14: Change the color of curve by overlaying a new curve with a different color.

Table 2.6: The macros for 3D plot types.

PLOT_PLOTTYPE_LINES	Data points are connected with a line.
PLOT_PLOTTYPE_IMPULSES	Display vertical lines from the xy plane to the data points.
PLOT_PLOTTYPE_POINTS	Markers are displayed at each data point.
PLOT_PLOTTYPE_LINESPOINTS	Markers are displayed at each data point and connected by a line.
PLOT_PLOTTYPE_SURFACES	Data points are connected and meshed in a smooth surface.
PLOT_PLOTTYPE_VECTORS	Display vectors.

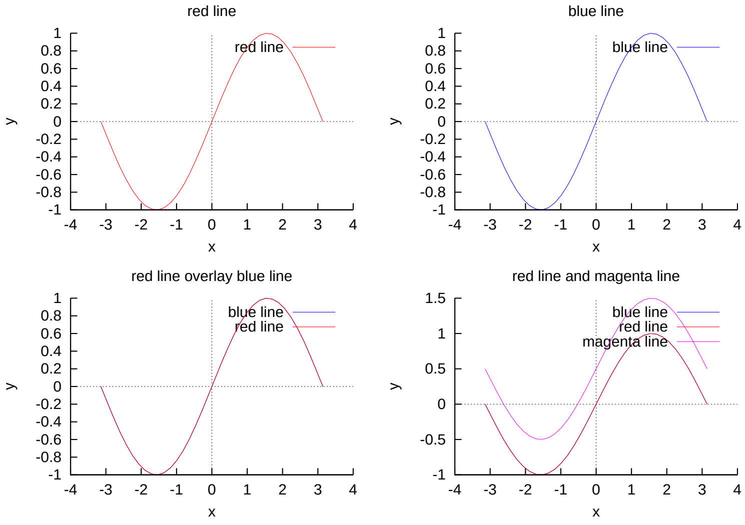


Figure 2.14: A plot with the color of curve changed by overlaying a new curve.

```
#include <chplot.h>
#include <stdio.h>

int main() {
    double x, y;
    char text[10];
    int datasetnum=0, point_type = 1, point_size = 5;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_X, 0, 7, 1);
    plot.axisRange(PLOT_AXIS_Y, 0, 5, 1);
    plot.title("Point Types in Ch Plot");
    for (y = 4; y >= 1; y--) {
        for (x = 1; x <= 6; x++) {
            sprintf(text, "%d", point_type);
            plot.point(x, y, 0);
            plot.plotType(PLOT_PLOTTYPE_POINTS, datasetnum);
            plot.pointType(datasetnum, point_type, point_size);
            plot.text(text, PLOT_TEXT_RIGHT, x-.25, y, 0);
            datasetnum++; point_type++;
        }
    }
    plot.plotting();
    return 0;
}
```

Program 2.15: A plotting program for different point types.

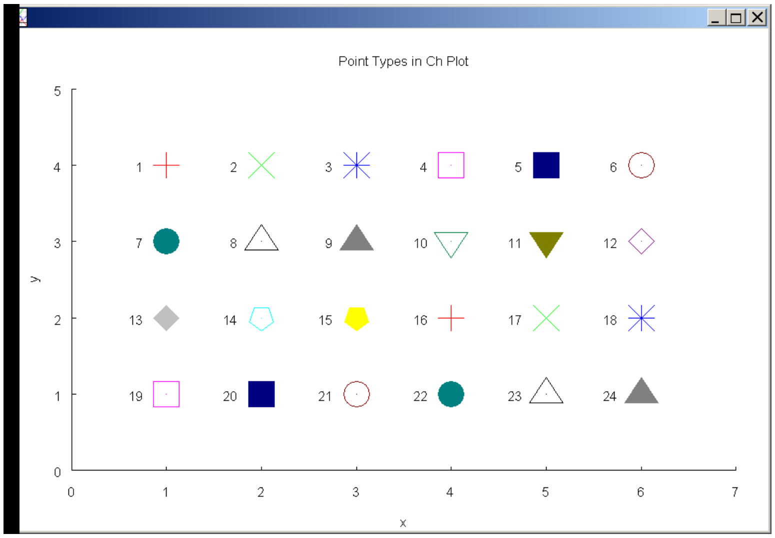


Figure 2.15: A plot with different point types dispayed in Windows.

```
#include <chplot.h>
#include <math.h>

#define NUM1 75
#define NUM2 300
int main() {
    double x1[NUM1], y1[NUM1];
    double x2[NUM2], y2[NUM2];
    class CPlot plot;
    int i, numdataset=0, point_type =1, point_size=1,
                         line_type =3,  line_width=1;

    for(i=0; i<NUM1; i++) {
      x1[i] = -2*M_PI + i*4*M_PI/(NUM1-1);  // linspace(x, -2*PI, 2*PI)
      y1[i] = x1[i]*x1[i] + 5*sin(10*x1[i]);
    }
    for(i=0; i<NUM2; i++) {
      x2[i] = -2*M_PI + i*4*M_PI/(NUM2-1);  // linspace(x, -2*PI, 2*PI)
      y2[i] = x2[i]*x2[i] + 5*sin(10*x2[i]);
    }
    plot.data2DCurve(x1, y1, NUM1);
    plot.data2DCurve(x2, y2, NUM2);
    plot.plotType(PLOT_PLOTTYPE_POINTS, numdataset);
    plot.pointType(numdataset, point_type, point_size);
    numdataset++;
    plot.plotType(PLOT_PLOTTYPE_LINES, numdataset);
    plot.lineType(numdataset, line_type, line_width);
    plot.plotting();
    return 0;
}
```

Program 2.16: A plotting program with two different plot types.

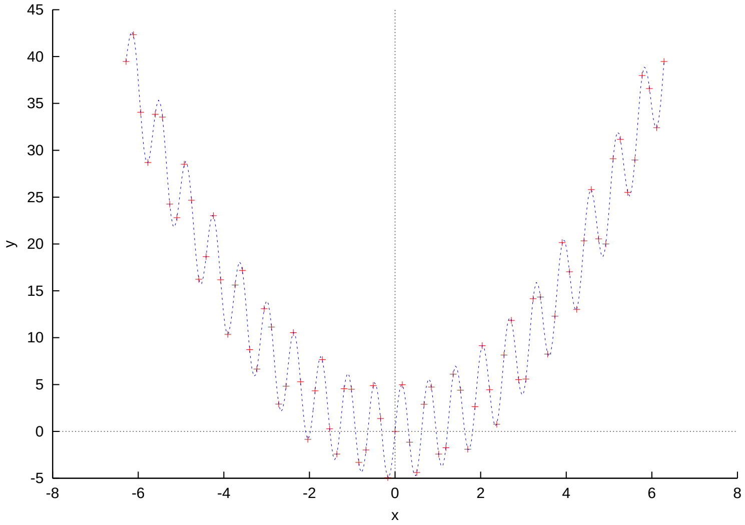


Figure 2.16: A plot with plot types of point and line.

```
#include <math.h>
#include <chplot.h>

#define NUM 360
int main() {
    int i;
    double theta[NUM], r[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
      theta[i] = 0 + i*M_PI/(NUM-1);  // linspace(theta, 0, PI)
      r[i] = sin(5*theta[i]);
    }
    plot.polarPlot(PLOT_ANGLE_RAD);
    plot.data2DCurve(theta, r, NUM);
    plot.sizeRatio(1);
    plot.grid(PLOT_ON);
    plot.plotting();
    return 0;
}
```

Program 2.17: A plotting program using a polar coordinate system.

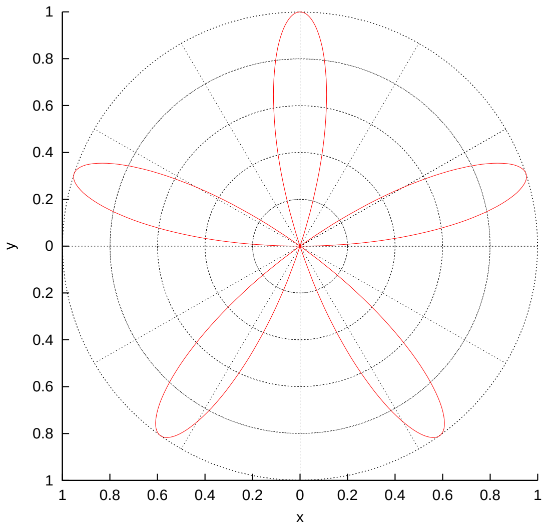


Figure 2.17: A plot in a polar coordinate system.

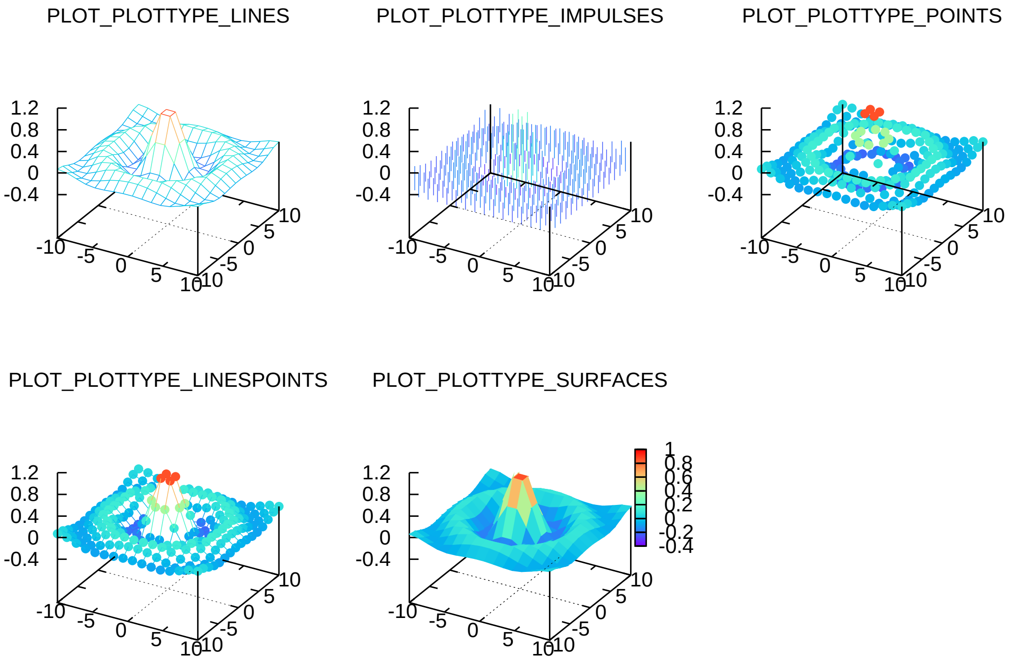


Figure 2.18: Three-dimensional plot types.

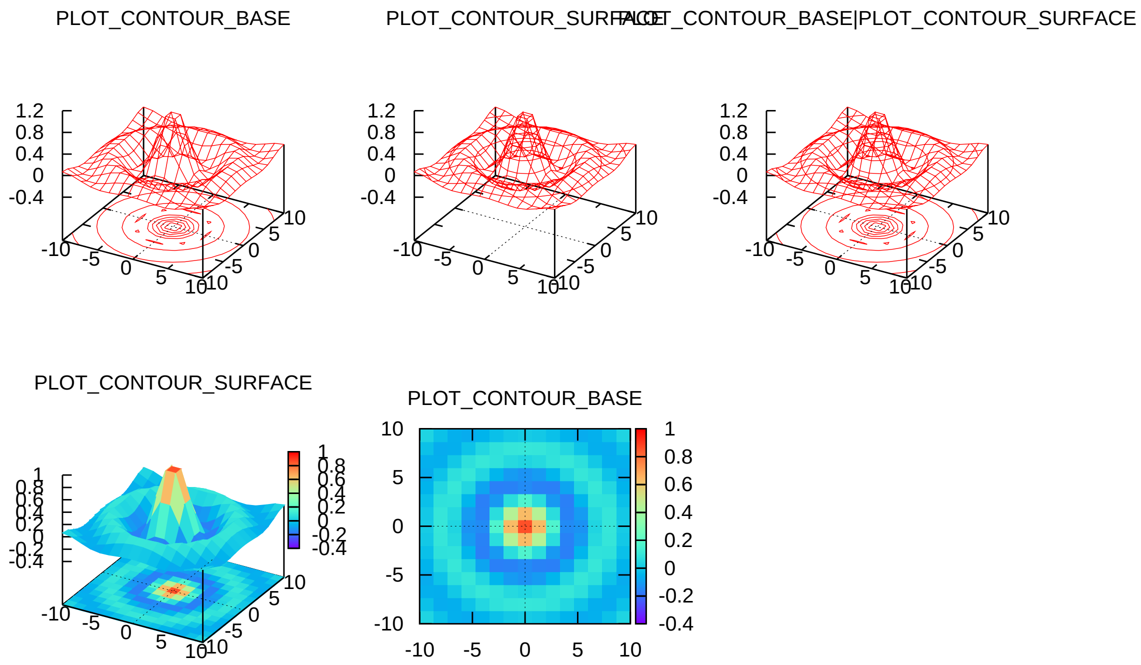


Figure 2.19: Contour modes.

2.3.2 Contour Plots

For a 3D grid data of the plot type `PLOT_PLOTTYPE_LINES`, contours can be drawn on a plot. The location of the contour lines is specified by member function

```
void CPlot::contourMode(int mode);
```

Two contour modes of **PLOT_CONTOUR_BASE** and **PLOT_CONTOUR_SURFACE** are available. The contour mode **PLOT_CONTOUR_BASE** draws contour lines on the x-y plane, whereas the contour mode **PLOT_CONTOUR_SURFACE** draws contour lines on the surface of the plot. Contour lines can be drawn on both the x-y plane and surface of the plot using the logical or (|) operator as shown in Figure 2.19.

The positions of the contour levels are determined internally unless explicitly set using member function **CPlot**::**contourLevels**(), which is prototyped as,

```
void CPlot::contourLevels(double levels[], int num);
```

This function allows contour levels for 3D grid data to be displayed at any desired z-axis position. The contour levels are stored in the array *levels* with the lowest contour as the first element of the array. The number of elements of array is provided as the second argument. In Program 2.18, the contour is located on the x-y plane and contour levels are equally spaced from -0.2 to 0.8 in the array `levels` with 6 elements. The display of labels for contours in Program 2.18 is turned on explicitly by member function

```
void CPlot::contourLabel(int flag);
```

The argument *flag* takes values of **PLOT_ON** or **PLOT_OFF**. By default, labels for contours are not displayed.

For the plot type of PLOT_PLOTTYPE_SURFACES, the contour mode of **PLOT_CONTOUR_SURFACE** will add a projected map on the xy plane. the contour mode of **PLOT_CONTOUR_SURFACE** will have a projected map on the xy plane only.

As an example, the plot generated by Program 2.18 is displayed in Figure 2.20.

2.3.3 Plotting in Different Coordinate Systems

In a two-dimensional case, a data set can be plotted in either a Cartesian or a polar coordinate system. In a three-dimensional case, a data set can be plotted in either the Cartesian, spherical, or cylindrical coordinate system. The coordinate system can be specified by member function

```
void CPlot::coordSystem(int coord_system);
void CPlot::coordSystem(int coord_system, int angle_unit);
```

The argument *coord_system* for the coordinate system can be set to one of three macros **PLOT_COORD_CARTESIAN**, **PLOT_COORD_SPHERICAL**, and **PLOT_COORD_CYLINDRICAL** which stand for Cartesian, spherical, and cylindrical coordinate systems, respectively. By default, a 3D plot uses the Cartesian coordinate system. A point in each coordinate system consists of three values. They are (x,y,z), (θ,ϕ,r), and (θ,z,r) for Cartesian, spherical, and cylindrical coordinate systems, respectively.

Points in a spherical coordinate system are mapped to the Cartesian space by the following formulas:

$$\begin{aligned} x &= r\cos(\theta)\cos(\phi) \\ y &= r\sin(\theta)\cos(\phi) \\ z &= r\sin(\phi) \end{aligned}$$

Program 2.19 generates a plot in a spherical coordinate system shown in Figure 2.21.

For a cylindrical coordinate system, points are mapped to the Cartesian space by formulas:

$$\begin{aligned} x &= r\cos(\theta) \\ y &= r\sin(\theta) \\ z &= z \end{aligned}$$

Program 2.20 generates a plot in a cylindrical coordinate system shown in Figure 2.22.

An optional argument *angle_unit* in member function **CPlot**::**coordSystem**() specifies the unit for measurement of angular positions in spherical and cylindrical coordinate systems. The valid macros for optional argument *angle_unit* are **PLOT_ANGLE_DEG** for measurement of angles in degrees and **PLOT_ANGLE_RAD** in radians.

2.4 Dynamic Web Plotting

Plotting through CGI programs is very useful for many Web-based applications. With Ch Professional Edition and CGI toolkit, plots can be very easily generated dynamically on-line. How to generate a dynamic plot will be presented in this section. We will also describe how data is encoded and decoded for transferring among the browser, Web server, and CGI programs.

In a Web-based plotting, the parameters for plotting are submitted from a Web browser, shown in Figure 2.23, with its corresponding HTML file in Program 2.21 and encoded by the browser. The parameters as name-value pairs are decoded by member function **CRequest**::**getFormNameValue**() in first CGI program

```
#include <math.h>
#include <chplot.h>

#define NUMX 30
#define NUMY 30
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double levels[6];
    double r;
    int datasetnum =0, i, j;
    int line_type = 1, line_width = 1;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
      x[i] = -10 + i*20.0/(NUMX-1);  // linspace(x, -10, 10)
    }
    for(i=0; i<NUMY; i++) {
      y[i] = -10 + i*20.0/(NUMY-1);  // linspace(y, -10, 10)
    }
    for(i=0; i<6; i++) {
      levels[i] = -0.2 + i*1.0/(6-1);  // linspace(levels, -0.2, 0.8)
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[30*i+j] = sin(r)/r;
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width);
    plot.title("function sin(r)/r");
    plot.label(PLOT_AXIS_X, "x-axis");
    plot.label(PLOT_AXIS_Y, "y-axis");
    plot.label(PLOT_AXIS_Z, "z-axis");
    plot.contourLabel(PLOT_ON);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.contourLevels(levels, 6);
    plot.plotting();
    return 0;
}
```

Program 2.18: A plotting program with contours and contour labels.

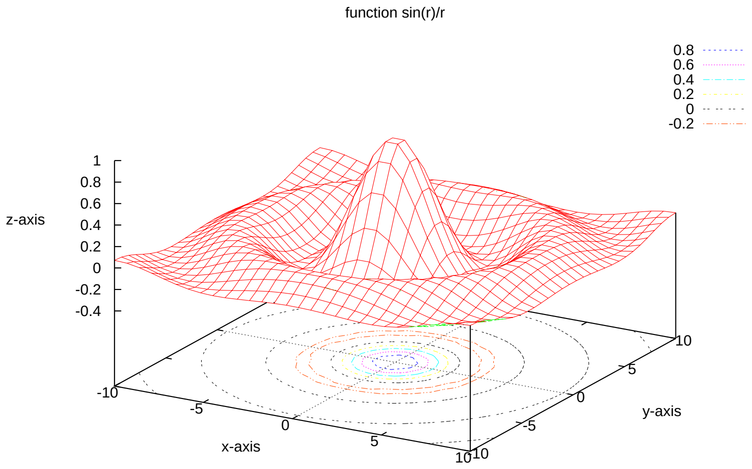


Figure 2.20: A plot with contours and contour labels.

```
#include <chplot.h>
#include <math.h>

#define NUMT 37
#define NUMP 19
int main() {
    int i;
    double theta[NUMT], phi[NUMP], r[NUMT*NUMP];
    class CPlot plot;

    for(i=0; i<NUMT; i++) {
      theta[i] = 0 + i*2*M_PI/(NUMT-1);    // linspace(theta, 0, 2*PI)
    }
    for(i=0; i<NUMP; i++) {
      phi[i] = -M_PI/2 + i*M_PI/(NUMP-1);  // linspace(phi, -PI/2, PI/2)
    }
    for(i=0; i<NUMT*NUMP; i++) {
      r[i] = 1;
    }
    plot.data3DSurface(theta, phi, r, NUMT, NUMP);
    plot.coordSystem(PLOT_COORD_SPHERICAL, PLOT_ANGLE_RAD);
    plot.axisRange(PLOT_AXIS_XY, -2.5, 2.5, 1);
    plot.plotting();
    return 0;
}
```

Program 2.19: A plotting program using a spherical coordinate system.

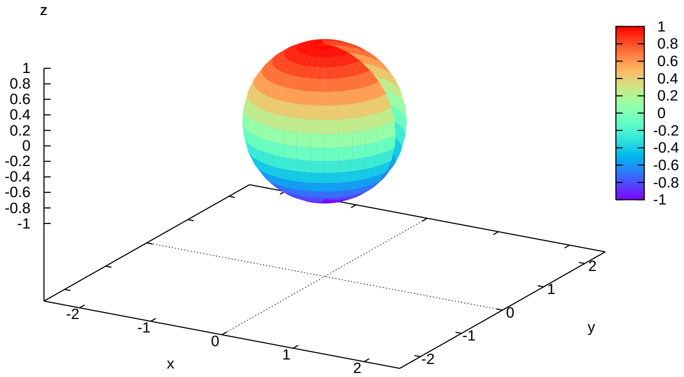


Figure 2.21: A plot in a spherical coordinate system.

```
#include <math.h>
#include <chplot.h>

#define NUMT 36
#define NUMZ 20
int main() {
    int numpoints = 36;
    double theta[NUMT], z[NUMZ], r[NUMT*NUMZ];
    int i, j;
    class CPlot plot;

    for(i=0; i<NUMT; i++) {
      theta[i] = 0 + i*360.0/(NUMT-1);    // linspace(theta, 0, 360)
    }
    for(i=0; i<NUMZ; i++) {
      z[i] = 0 + i*2*M_PI/(NUMZ-1);       // linspace(z, 0, 2*PI)
    }
    for(i=0; i<NUMT; i++) {
        for(j=0; j<NUMZ; j++) {
            r[i*NUMZ+j] = 2+cos(z[j]);
        }
    }
    plot.data3DSurface(theta, z, r, NUMT, NUMZ);
    plot.coordSystem(PLOT_COORD_CYLINDRICAL, PLOT_ANGLE_DEG);
    plot.axisRange(PLOT_AXIS_XY, -4, 4, 1);
    plot.plotting();
    return 0;
}
```

Program 2.20: A plotting program using a cylindrical coordinate system.

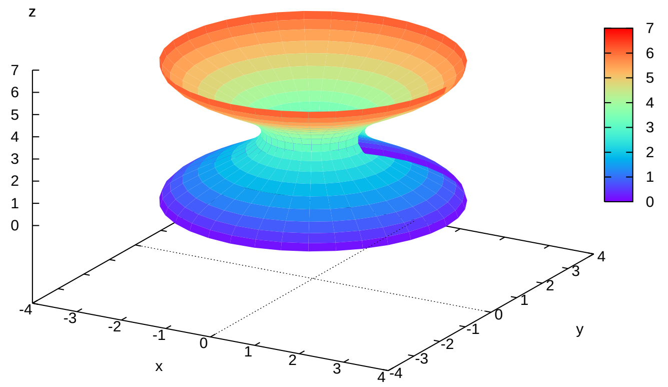
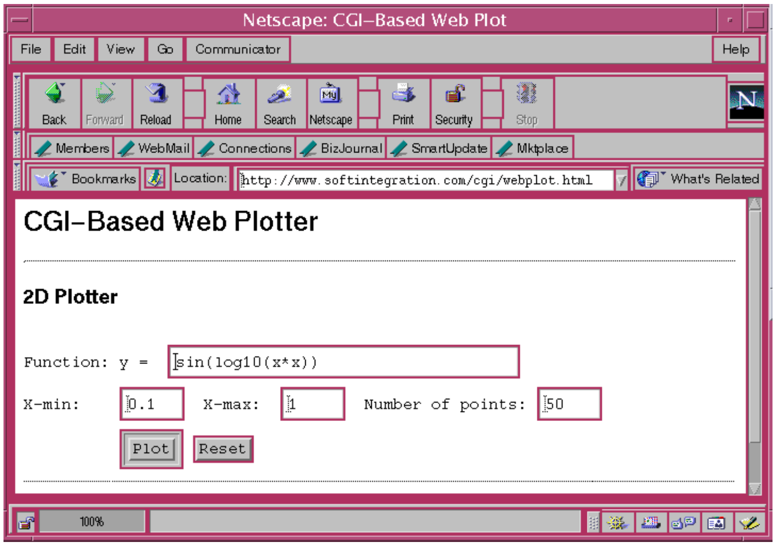


Figure 2.22: A plot in a spherical cylindrical coordinate system.

webplot1.ch shown in Program 2.22. They are then passed as query strings to the second CGI program **webplot2.ch** shown in Program 2.23. These parameters are obtained again using member function **CRequest**::**getFormNameValue**(). The plot generated as a PNG file and displayed through a Web browser is shown in Figure 2.24 .



http://www.softintegration.com/cgi/webplot.html

CGI–Based Web Plotter

2D Plotter

Function: y = sin(log10(x*x))

X-min: 0.1 X-max: 1 Number of points: 50

Figure 2.23: A Web-plotter based on the fill-out form.

```
<HTML>
<HEAD>
<TITLE>
CGI-Based Web Plot
</TITLE>
</HEAD>
<BODY bgcolor="#FFFFFF" text="#000000" vlink="#FF0000">
<H1>
CGI-Based Web Plotter
</H1>

<HR>
<H2>2D Plotter</H2>
<PRE>
<FORM method="post" action="/cgi-bin/chcgi/toolkit/demos/sample/webplot1.ch">
Function: y =  <INPUT name="expression" value="sin(log10(x*x))" size=35>
X-min:    <INPUT name="xMin" value="0.1" size=5>  X-max:  <INPUT name="xMax"
value="1" size=5>  Number of points: <INPUT name="numpoints" value="50" size=5>
          <INPUT type="submit" value="Plot"> <INPUT type="reset" value="Reset">
<HR>
</BODY>
</HTML>
```

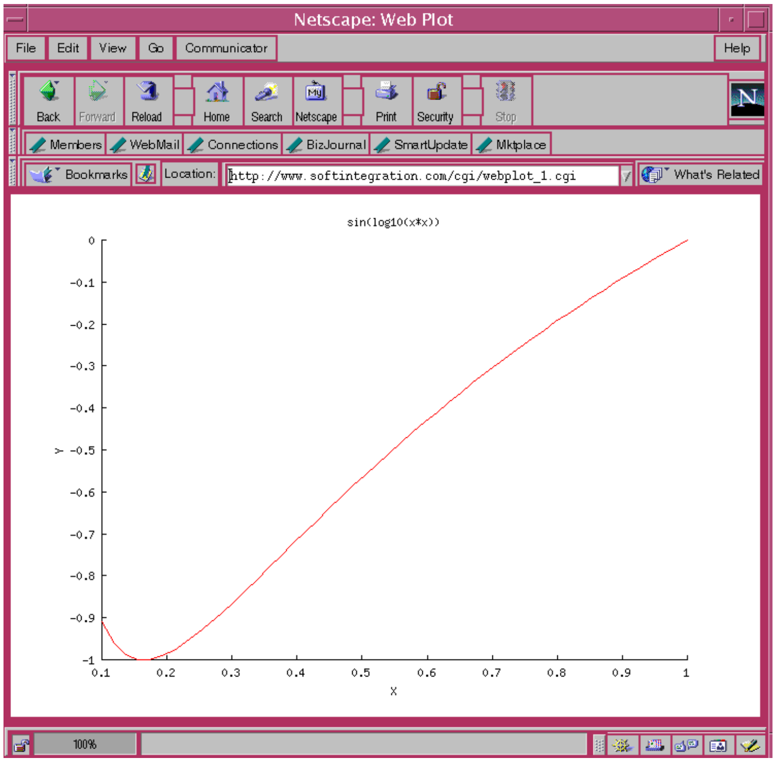
Program 2.21: HTML file for submitting plotting parameters.

```
#!/bin/ch
#include <cgi.h>

int main() {
   int i, num;
   chstrarray name, value;
   class CResponse Response;
   class CRequest Request;
   class CServer Server;

   num = Request.getFormNameValue(name, value);
   Response.setContentType("text/html");
   Response.begin();
   Response.title("Web Plot");
   printf("<center>\n");
   printf("<img src=\"/cgi-bin/chcgi/toolkit/demos/sample/webplot2.ch");
   for (i=0; i<num; i++){
      putc(i == 0 ? ’?’ : ’&’, stdout);
      fputs(Server.URLEncode(name[i]),stdout);
      putc(’=’, stdout);
      fputs(Server.URLEncode(value[i]),stdout);
   }
   printf("\">\n");
   printf("</center>\n");
   Response.end();
}
```

Program 2.22: CGI program webplot1.ch

```
#!/bin/ch
#include <cgi.h>
#include <chplot.h>

int main() {
   double MinX, MaxX, Step, x, y;
   int pointsX, pointsY, i;
   chstrarray name, value;
   class CResponse Response;
   class CRequest Request;
   class CPlot plot;

   Request.getFormNameValue(name, value);
   MinX = atof(value[1]);
   MaxX = atof(value[2]);
   pointsX = atoi(value[3]);
   double x1[pointsX], y1[pointsX];

   Step = (MaxX - MinX)/(pointsX-1);
   for(i=0;i<pointsX;i++) {
     x = MinX + (i*Step);
     y = streval(value[0]);
     x1[i] = x;
     y1[i] = y;
   }

   Response.setContentType("image/png");
   Response.begin();
   plotxy(x1, y1, value[0], "X", "Y", &plot);
   /* output plot in color png file format */
   plot.outputType(PLOT_OUTPUTTYPE_STREAM, "png");
   plot.plotting();
   Response.end();
}
```

Program 2.23: CGI program webplot2.ch



Location: http://www.softintegration.com/cgi/webplot_1.cgi

Figure 2.24: Plot generated through the Web plotting.

Chapter 3

Distribution of Applications with SIGL

Applications developed with SoftIntegration Graphical Library (SIGL) can be distributed according to the procedures described in this chapter. Please note that applications developed using SoftIntegration Graphical Library Student Edition bundled with Ch Student Edition cannot be distributed.

3.1 Build an Application for Distribution

For distribution of your application built using SIGL to run in a machine without SIGL installed, each instance of the CPlot class should be instantiated using the constructor

```
     CPlot::CPlot(const char *licensestr);
```

with an argument of license string from SoftIntegration as shown below.

```
   // create a license string.
   // fill '...' and some info  below with the actual contents
   //  from SoftIntegration
   char licensestr[] =
"<LICENSE softinteg graphical_library .... \
  options=\"This is ... only\" _ck=937baff264 sig="60PG4... QR4YPY\
  ABCS...ACDCD\">";

   CPlot plot(licensestr);
```

If an application built using SIGL runs in a machine with SIGL installed, the license string is ignored. Therefore, an instance of the CPlot class can just be instantiated simply using

```
   CPlot plot;
```

3.2 Distribution in Windows

Applications developed with SoftIntegration Graphical Library can be distributed in Windows as follows.

1. Copy the dynamically linked library libchplot.dll from the directory SILIB/lib to the Windows system directory C:\windows\system32 or C:\winnt\system32, where SILIB is the home directory of the SoftIntegration Graphical Library.

If your application is compiled with compiler option /MD and linked with the static library SILIB/lib/libchplot_a.lib, the application does not need the dynamically linked library libchplot.dll at runtime. You can ignore this step.

2. Create a runtime home directory for the SoftIntegration Graphical Library, say C:\sigl. Setup the environment variable SIGL_HOME to this directory. An environment variable can be setup under `System` menu in the `Control Panel` for Windows NT, 2000, and XP. You can set up environment variable for Windows 9x from autoexec.bat with ”set SIGL_HOME=C:\sigl”. You need to reboot the computer to effectively setup an environment for Win9x.

3. Copy runtime plotting programs in SILIB\bin in your developer machine to SIGL_HOME\bin in the target machine.

4. Copy your application with SIGL in a command directory in the target machine

5. When you build an application in Microsoft Visual Studio, and the application uses the C run-time libraries (CRT), distribute the appropriate CRT DLL from the following list with your application:

   - Msvcr90.dll for Microsoft Visual C++ 2008
   - Msvcr80.dll for Microsoft Visual C++ 2005
   - Msvcr71.dll for Microsoft Visual C++ .NET 2003 with the Microsoft .NET Framework 1.1

   For Msvcr71.dll, you should install the CRT DLL into your application program files directory. You should not install these files into the Windows system directories. For Msvcr80.dll and Msvcr90.dll, you should install the CRT as Windows side-by-side assemblies. You may consult with documentation from Microsoft about how to distribute a binary application build using VC++ in .NET.

3.3 Distribution in Unix and Mac OS X

Applications developed with SoftIntegration Graphical Library can be distributed in Unix and Mac OS X as follows.

1. Create a runtime home directory for the SoftIntegration Graphical Library, say /usr/local/sigl. Setup the environment variable SIGL_HOME to this directory.

2. Copy runtime plotting programs in SILIB/bin in your developer machine to SIGL_HOME/bin in the target machine.

3. Copy your application with SIGL in a command directory in the target machine

Chapter 4

Reference for Class CPlot

The header file **chplot.h** available in both Ch and C++ contains the definition of the plotting **CPlot** class, defined macros used with **CPlot**, and definitions of high-level plotting functions that are based on **CPlot** class .

The plotting class **CPlot** allows for high-level creation and manipulation of plots within the Ch language environment. **CPlot** can be used directly within many types of Ch programs, including applications, function files, and CGI programs. Plots can be generated from data arrays or files, and can be displayed on a screen, saved in a large number of different file formats, or generated as a stdout stream in png or gif file format on the Web.

Compatability between Ch and C++

High-level plotting functions **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyf**(), **plotxyz**(), and **plotxyzf**() as well as member functions **CPlot**::**data2D**() and **CPlot**::**data3D**() based on array of reference are available only in Ch. These functions are designed to be easy to use and allow plots to be created quickly. These functions can be used in conjunction with other **CPlot** member functions to create more sophisticated plots.

CPlot Class

The **CPlot** class can be used to produce two dimensional (2D) and three dimensional (3D) plots through a Ch program. Member functions of the **CPlot** class generate actual plots using a plotting engine. Gnuplot is used internally as the plotting engine for display in this release of the Ch language environment.

Public Data

None.

Public Member Functions

Function	Description
CPlot()	Class constructor. Creates and initializes a new instance of the class.
˜**CPlot**()	Class destructor. Frees memory associated with a instance of the class.
arrow()	Add an arrow to a plot.
autoScale()	Enable or disable autoscaling of plot axes.
axis()	Enable or disable drawing of x-y axis on 2D plots.

axisRange()	Set the range for a plot axis.
axes()	Specify the axes for a data set.
barSize()	Set the size of error bars.
border()	Enable or disable drawing of a border around the plot.
borderOffsets()	Set plot offsets of the plot border.
boundingBoxOrigin()	obsolte, use CPlot::origin().
boxBorder()	Enable or disable drawing of a border for a plot of box type.
boxFill()	Fill a box or filled curve with a solidcolor or pattern.
boxWidth()	Set the width for a box.
changeViewAngle()	Change the view angles for a 3D plot.
circle()	Add a circle to a 2D plot.
colorBox()	Enable or disable the drawing of a color box for 3D surface plots.
contourLevels()	Set contour levels for 3D plot to be displayed at specific locations.
contourLabel()	Enable or disable contour labels for 3D surface plots.
contourLevels()	Set contour levels for 3D plot to be displayed at specific locations.
contourMode()	Set the contour display mode for 3D surface plots.
coordSystem()	Set the coordinate system for a 3D plot.
data()	Add 2D, 3D, or multi-dimensional data to an instance of the **CPlot** class.
data2D()	Add one or more 2D data sets to an instance of the **CPlot** class (for Ch only).
data2DCurve()	Add a set of data for 2D curve to an instance of the **CPlot** class.
data3D()	Add one or more 3D data sets to an instance of the **CPlot** class (for Ch only).
data3DCurve()	Add a set of data for 3D curve to an instance of the **CPlot** class.
data3DSurface()	Add a set of data for 3D surface to an instance of the **CPlot** class.
dataFile()	Add a data file to an instance of the **CPlot** class.
dataSetNum()	Obtain the current data set number in an instance of the **CPlot** class.
deleteData()	Remove data from a previously used instance of the **CPlot** class.
deletePlots()	Remove any data from a previously used instance of the **CPlot** class and reinitialize option to default values.
dimension()	Set plot dimension to 2D or 3D.
displayTime()	Display the current time and date on the plot.
enhanceText()	Use enhanced text for special symbols.
func2D()	Add a set of 2D data using a function to an instance of the **CPlot** class.
func3D()	Add a set of 3D data using a function to an instance of the **CPlot** class.
funcp2D()	Add a set of 2D data using a function with a paramenter to an instance of the **CPlot** class.
funcp3D()	Add a set of 3D data using a function with a parameter to an instance of the **CPlot** class.
getLabel()	Get the label of an axis.
getOutputType()	Get the output type of a plot.
getSubplot()	Get a pointer to an element of a subplot.
getTitle()	Get the title of a plot.
grid()	Enable or disable display of a grid.
isUsed()	Test if an instance of the **CPlot** class has been used.
label()	Set axis labels.
legend()	Add a legend for a data set.
legendLocation()	Specify the plot legend location.
legendOption()	Set options for legends of a plot.
line()	Add a line to a plot.
lineType()	Set the line type, width, and color for lines, impulses, steps, etc.
margins()	Set plot margins.

origin()	Set the location of the origin for the bounding box of the plot.
outputType()	Set the plot output type.
plotType()	Set the plot type.
plotting()	Produce a plot from an instance of the **CPlot** class.
point()	Add a point to a plot.
pointType()	Set the point type, size, and color.
polarPlot()	Set a 2D plot to use the polar coordinate system.
polygon()	Add a polygon to a plot.
rectangle()	Add a rectangle to a 2D plot.
removeHiddenLine()	Enable or disable hidden line removal for 3D plots.
scaleType()	Set the axis scale type for a plot.
showMesh()	Enable or disable display of mesh of a 3D plot.
size()	Scale the plot itself relative to the size of the output file or canvas.
size3D()	Change the size of a 3D plot.
sizeOutput()	Change the size of the output file.
sizeRatio()	Change the aspect ratio of a plot.
smooth()	Smooth plotting curves by interpolation and approximation of data.
subplot()	Create a group of subplots.
text()	Add a text string to a plot.
tics()	Enable or disable display of axis tics.
ticsDay()	Set axis tic-mark labels to days of the week.
ticsDirection()	Set the direction in which axis tic-marks are drawn.
ticsFormat()	Set the number format for tic labels.
ticsLabel()	Set location and text label for arbitrary axis labels.
ticsLevel()	Set the z-axis offset for drawing of tics in 3D plots.
ticsLocation()	Specify the location of axis tic marks to be on the border or the axis.
ticsMirror()	Enable or disable display of axis tics on the opposite axis.
ticsMonth()	Set axis tic-mark labels to months.
ticsPosition()	Add tic-marks at the specified positions to an axis.
ticsRange()	Specify the range for a series of tics on an axis.
title()	Set the plot title.

Macros

The following macros are defined for the **CPlot** class.

Macro	Description
PLOT_ANGLE_DEG	Select units in degree for angular values.
PLOT_ANGLE_RAD	Select units in radian for angular values.
PLOT_AXIS_X	Select the x axis only.
PLOT_AXIS_X2	Select the x2 axis on the top only.
PLOT_AXIS_XY	Select the x and y axes.
PLOT_AXIS_XYZ	Select the x, y, and z axes.
PLOT_AXIS_Y	Select the y axis only.
PLOT_AXIS_Y2	Select the y2 axis on the right only.
PLOT_AXIS_Z	Select the z axis only.

PLOT_BORDER_BOTTOM	The bottom of the plot.
PLOT_BORDER_LEFT	The left side of the plot.
PLOT_BORDER_TOP	The top of the plot.
PLOT_BORDER_RIGHT	The right side of the plot.
PLOT_BORDER_ALL	All sides of the plot.
PLOT_BOXFILL_EMPTY	Do not fill a box.
PLOT_BOXFILL_SOLID	Fill a box with a solid color.
PLOT_BOXFILL_PATTERN	Fill a box with a pattern.
PLOT_CONTOUR_BASE	Draw contour lines for a surface plot on the x-y plane.
PLOT_CONTOUR_SURFACE	Draw contour lines for a surface plot on the surface.
PLOT_COORD_CARTESIAN	Use a Cartesian coordinate system for a 3D plot.
PLOT_COORD_CYLINDRICAL	Use a cylindrical coordinate system in a 3D plot.
PLOT_COORD_SPHERICAL	Use a spherical coordinate system in a 3D plot.
PLOT_OFF	Flag to disable an option.
PLOT_ON	Flag to enable an option.
PLOT_OUTPUTTYPE_DISPLAY	Display the plot on a screen.
PLOT_OUTPUTTYPE_FILE	Output the plot to a file.
PLOT_OUTPUTTYPE_STREAM	Output the plot as a stdout stream.
PLOT_PLOTTYPE_BOXERRORBARS	It is a combination of the **PLOT_PLOTTYPE_BOXES** and **PLOT_PLOTTYPE_YERRORBARS** plot types.
PLOT_PLOTTYPE_BOXES	Draw a box centered about the given x coordinate.
PLOT_PLOTTYPE_BOXXYERRORBARS	A combination of **PLOT_PLOTTYPE_BOXES** and **PLOT_PLOTTYPE_XYERRORBARS** plot types.
PLOT_PLOTTYPE_CANDLESTICKS	Display box-and-whisker plots of financial or statistical data.
PLOT_PLOTTYPE_DOTS	Use dots to mark each data point.
PLOT_PLOTTYPE_FILLEDCURVES	Fill an area bounded in a side by a curve with a solid color or pattern.
PLOT_PLOTTYPE_FINANCEBARS	Display finanacial data.
PLOT_PLOTTYPE_FSTEPS	Adjacent points are connected with two line segments, one from (x1,y1) to (x1,y2), and a second from (x1,y2) to (x2,y2).
PLOT_PLOTTYPE_HISTEPS	The point x1 is represented by a horizontal line from ((x0+x1)/2,y1) to ((x1+x2)/2,y1). Adjacent lines are connected with a vertical line from ((x1+x2)/2,y1) to ((x1+x2)/2,y2).
PLOT_PLOTTYPE_IMPULSES	Display vertical lines from the x-axis (for 2D plots) or the x-y plane (for 3D plots) to the data points.
PLOT_PLOTTYPE_LINES	Data points are connected with a line.
PLOT_PLOTTYPE_LINESPOINTS	Markers are displayed at each data point and connected with a line.
PLOT_PLOTTYPE_POINTS	Markers are displayed at each data point.
PLOT_PLOTTYPE_STEPS	Adjacent points are connected with two line segments, one from (x1,y1) to (x2,y1), and a second from (x2,y1) to (x2,y2).
PLOT_PLOTTYPE_SURFACES	Data points are connected and smoothed as a surface. For 3D plot only.
PLOT_PLOTTYPE_VECTORS	Display vectors.
PLOT_PLOTTYPE_XERRORBARS	Display dots with horizontal error bars.
PLOT_PLOTTYPE_XERRORLINES	Display linepoints with horizontal error lines.

PLOT_PLOTTYPE_XYERRORBARS	Display dots with horizontal and vertical error bars.
PLOT_PLOTTYPE_XYERRORLINES	Display linepoints with horizontal and vertical error lines.
PLOT_PLOTTYPE_YERRORBARS	Display points with vertical error bars.
PLOT_PLOTTYPE_YERRORLINES	Display linepoints with vertical error lines.
PLOT_SCALETYPE_LINEAR	Use a linear scale for a specified axis.
PLOT_SCALETYPE_LOG	Use a logarithmic scale for a specified axis.
PLOT_TEXT_CENTER	Center text at a specified point.
PLOT_TEXT_LEFT	Left justify text at a specified point.
PLOT_TEXT_RIGHT	Right justify text at a specified point.
PLOT_TICS_IN	Draw axis tic-marks inward.
PLOT_TICS_OUT	Draw axis tic-marks outward.

Functions

The following functions are implemented using the **CPlot** class and avaialble in both SIGL and Ch.

Function	Description
fplotxy()	Plot a 2D function of *x* in a specified range or initialize an instance of the **CPlot** class.
fplotxyz()	Plot a 3D function of *x* and *y* in a specified range or initialize an instance of the **CPlot** class.
plotxy()	Plot a 2D data set or initialize an instance of the **CPlot** class.
plotxyf()	Plot 2D data from a file or initialize an instance of the **CPlot** class.
plotxyz()	Plot a 3D data set or initialize an instance of the **CPlot** class.
plotxyzf()	Plot 3D data from a file or initialize an instance of the **CPlot** class.

References

T. Williams, C. Kelley, D. Denholm, D. Crawford, et al., *Gnuplot — An Interactive plotting Program*, Version 3.7, December 3, 1998, ftp://ftp.gnuplot.vt.edu/.

Copyright Notice of Gnuplot

```
/*[
 * Copyright 1986 - 1993, 1998   Thomas Williams, Colin Kelley
 *
 * Permission to use, copy, and distribute this software and its
 * documentation for any purpose with or without fee is hereby granted,
 * provided that the above copyright notice appear in all copies and
 * that both that copyright notice and this permission notice appear
 * in supporting documentation.
 *
 * Permission to modify the software is granted, but not the right to
 * distribute the complete modified source code.  Modifications are to
 * be distributed as patches to the released version.  Permission to
 * distribute binaries produced by compiling modified sources is granted,
 * provided you
 *   1. distribute the corresponding source modifications from the
 * released version in the form of a patch file along with the binaries,
 *   2. add special version identification to distinguish your version
 * in addition to the base release version number,
 *   3. provide your name and address as the primary contact for the
 * support of your modified version, and
 *   4. retain our contact information in regard to use of the base
 * software.
 * Permission to distribute the released version of the source code
 * along with corresponding source modifications in the form of a patch
 * file is granted with same provisions 2 through 4 for binary
 * distributions.
 *
 * This software is provided "as is" without express or implied warranty
 * to the extent permitted by applicable law.
]*/
```

CPlot::**arrow**

Synopsis in Ch
#include <chplot.h>
void arrow(**double** *x_head*, **double** *y_head*, ... /* **double** *z_head*, **double** *x_tail*, **double** *y_tail*, **double** *z_tail*, [**char** **option*] */);

Synopsis in C++
#include <chplot.h>
void arrow(**double** *x_head*, **double** *y_head*, **double** *x_tail*, **double** *y_tail*);
void arrow(**double** *x_head*, **double** *y_head*, **double** *x_tail*, **double** *y_tail*, **char** **option*);
void arrow(**double** *x_head*, **double** *y_head*, **double** *z_head*, **double** *x_tail*, **double** *y_tail*, **double** *z_tail*);
void arrow(**double** *x_head*, **double** *y_head*, **double** *z_head*, **double** *x_tail*, **double** *y_tail*, **double** *z_tail*, **char** **option*);

Syntax in Ch and C++
arrow(*x_head*, *y_head*, *x_tail*, *y_tail*)
arrow(*x_head*, *y_head*, *x_tail*, *y_tail*, *option*)
arrow(*x_head*, *y_head*, *z_head*, *x_tail*, *y_tail*, *z_tail*)
arrow(*x_head*, *y_head*, *z_head*, *x_tail*, *y_tail*, *z_tail*, *option*)

Purpose
Add an arrow to a plot.

Return Value
None.

Parameters

x_head The x coordinate of the head of the arrow.

y_head The y coordinate of the head of the arrow.

z_head For 2D plots this is ignored. For 3D plots, the z coordinate of the head of the arrow.

x_tail For x coordinate of the tail of the arrow.

y_tail The y coordinate of the tail of the arrow.

z_tail For 2D plots this is ignored. For 3D plots, the z coordinate of the tail of the arrow.

option The option for the arrow.

Description
This function adds an arrow to a plot. The arrow points from (*x_tail*, *y_tail*, *z_tail*) to (*x_head*, *y_head*, *z_head*). These coordinates are specified using the same coordinate system as the curves of the plot.

An arrow is not counted as a curve. Therefore, it does not affect the number of legends added by **CPlot**::**legend**(legend, num).

The optional argument `option` of string type with the following values can be used to fine tune the arrow based on the argument for set arrow command of the gnuplot.

```
{ {nohead | head | backhead | heads}
  {size <length>,<angle>{,<backangle>}}
  {filled | empty | nofilled}
  {front | back}
  { {linestyle | ls <line_style>}
    | {linetype | lt <line_type>}
      {linewidth | lw <line_width} } } }
```

Specifying ‘nohead‘ produces an arrow drawn without a head—a line segment. This gives you yet another way to draw a line segment on the plot. By default, an arrow has a head at its end. Specifying ‘backhead‘ draws an arrow head at the start point of the arrow while ‘heads‘ draws arrow heads on both ends of the line. Not all terminal types support double-ended arrows.

Head size can be controlled by ‘size <length>,<angle>‘ or ‘size <length>,<angle>,<backangle>‘, where ‘<length>‘ defines length of each branch of the arrow head and ‘¡angle¿‘ the angle (in degrees) they make with the arrow. ‘<Length>‘ is in x-axis units; this can be changed by ‘first‘, ‘second‘, ‘graph‘, ‘screen‘, or ‘character‘ before the ¡length¿; see ‘coordinates‘ for details. ‘¡Backangle¿‘ only takes effect when ‘filled‘ or ‘empty‘ is also used. Then, ‘<backangle>‘ is the angle (in degrees) the back branches make with the arrow (in the same direction as ‘<angle>‘). The ‘fig‘ terminal has a restricted backangle function. It supports three different angles. There are two thresholds: Below 70 degrees, the arrow head gets an indented back angle. Above 110 degrees, the arrow head has an acute back angle. Between these thresholds, the back line is straight.

Specifying ‘filled‘ produces filled arrow heads (if heads are used). Filling is supported on filled-polygon capable terminals, otherwise the arrow heads are closed but not filled. The same result (closed but not filled arrow head) is reached by specifying ‘empty‘.

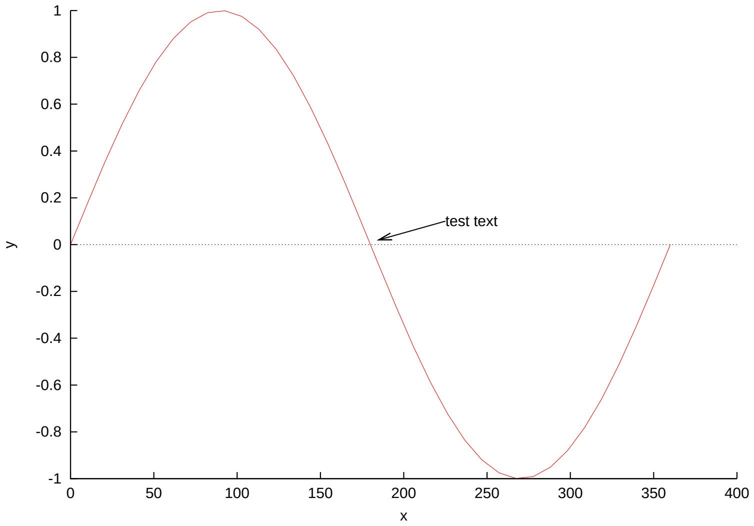
If ‘front‘ is given, the arrow is written on top of the graphed data. If ‘back‘ is given (the default), the arrow is written underneath the graphed data. Using ‘front‘ will prevent an arrow from being obscured by dense data.

The ‘linetype‘ is followed by an integer index representing the line type for drawing. The line type varies depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line or make it dashed or dotted. All terminals support at least six different line types. The ‘linewidth‘ is followed by a scaling factor for the line width. The line width is ‘linewidth‘ multiplied by the default width. Typically the default width is one pixel.

Example 1
Compare with output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360);
       y[i] = sin(x[i]*M_PI/180);
    }

    plot.arrow(185, 0.02, 225, 0.1);
    plot.text("test text", PLOT_TEXT_LEFT, 225, 0.1);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output

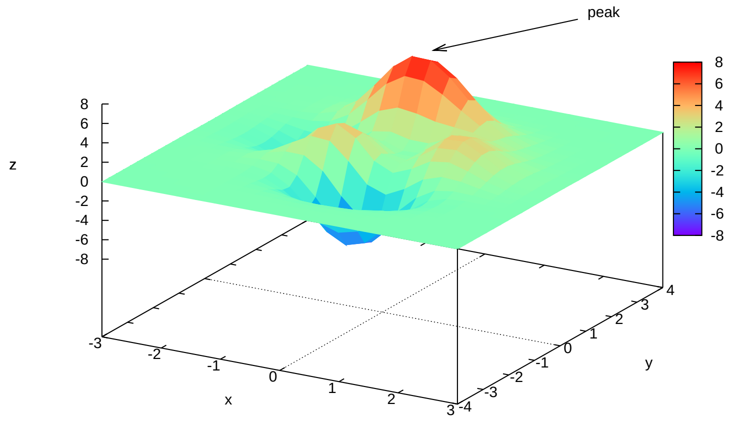


Example 2

Compare with the output for examples in **CPlot::data3D**() and **CPlot::data3DSurface**().

```
#include <math.h>
#include <chplot.h>

#define NUMX 20
#define NUMY 30
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    int i,j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -3 + i*6.0/(NUMX-1); // linspace(x, -3, 3);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -4 + i*8.0/(NUMY-1); // linspace(y, -4, 4);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[NUMY*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }

    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.arrow(0, 2, 8, 2, 3, 12);
    plot.text("peak", PLOT_TEXT_LEFT, 2.1, 3.15, 12.6);
    plot.plotting();
    return 0;
}
```

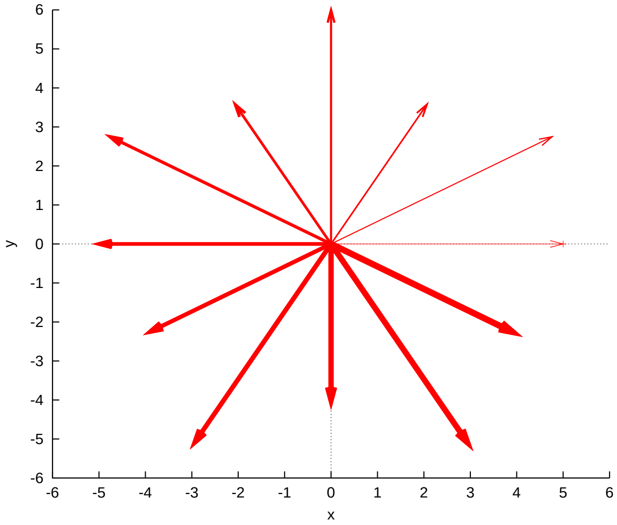
Output



Example 3

```
#include <math.h>
#include <chplot.h>

int main() {
    double x[12], y[12];
    char option[64];
    int i;
    class CPlot plot;
    int point_type = 0;

    for (i=0; i<12; i++) {
        x[i] = (5+sin(150*i*M_PI/180))*cos(30*i*M_PI/180);
        y[i] = (5+sin(150*i*M_PI/180))*sin(30*i*M_PI/180);
        sprintf(option, "linetype 1 linewidth %d", i+1);
        plot.arrow(x[i], y[i], 0, 0, 0, 0, option);
    }
    plot.axisRange(PLOT_AXIS_XY, -6.0, 6.0, 1);   /* one point cannot do autorange */
    plot.point(x[0], y[0], 0); /* CPlot::arrow() itself is not a data set */
    plot.plotType(PLOT_PLOTTYPE_POINTS, 0);
    plot.sizeRatio(-1);
    plot.plotting();
    return 0;
}
```

Output



Example 4
See an example on page 179 for **CPlot:plotType**() on how `option` is used in comparison with the plot type **PLOT_PLOTTYPE_VECTORS**.

See Also
CPlot::**circle**(), **CPlot**::**data2D**(), **CPlot**::**outputType**(), **CPlot**::**plotType**(), **CPlot**::**point**(), **CPlot**::**polygon**(), **CPlot**::**rectangle**().

CPlot::autoScale

Synopsis
#include <**chplot.h**>
void autoScale(**int** *axis*, **int** *flag*);

Purpose
Set autoscaling of axes on or off.

Return Value
None.

Parameters
axis The axis to be set. Valid values are:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

flag Enable or disable auto scaling.

PLOT_ON The option is enabled.

PLOT_OFF The option is disabled.

Description

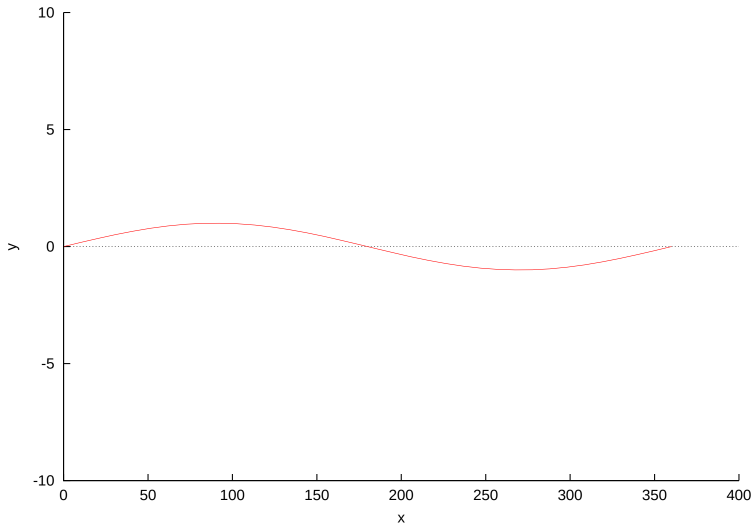
Autoscaling of the axes can be **PLOT_ON** or **PLOT_OFF**. Default is **PLOT_ON**. If autoscaling for an axis is disabled, an axis range of [-10:10] is used.

Example

Compare with the output for examples in **CPlot::data2D**() and **CPlot::data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for (i=0; i<NUM; i++) {
      x[i] = 0+i*360.0/(NUM-1);    // linspace(x, 0, 360);
      y[i] = sin(x[i]*M_PI/180);
    }
    plot.autoScale(PLOT_AXIS_Y, PLOT_OFF);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



CPlot::axis

Synopsis
#include <chplot.h>
void axis(**int** *axis*, **int** *flag*);

Purpose
Enable or disable drawing of x-y axis on 2D plots.

Return Value
None.

Parameters
axis The *axis* parameter can take one of the following values:

- **PLOT_AXIS_X** Select the x axis only.
- **PLOT_AXIS_X2** Select the x2 axis only.
- **PLOT_AXIS_Y** Select the y axis only.
- **PLOT_AXIS_Y2** Select the y2 axis only.
- **PLOT_AXIS_XY** Select the x and y axes.

flag This parameter can be set to:

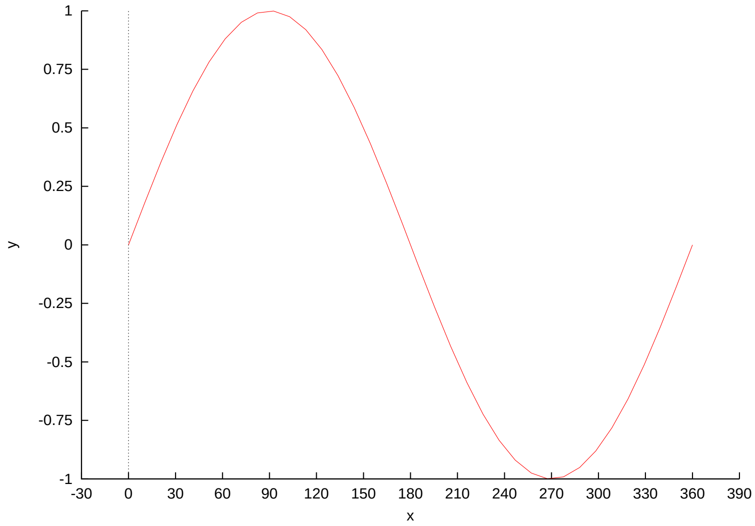
- **PLOT_ON** Enable drawing of the specified axis.
- **PLOT_OFF** Disable drawing of the specified axis.

Description
Enable or disable the drawing of the x-y axes on 2D plots. By default, the x and y axes are drawn.

Example
Compare with the output for the example in **CPlot**::**axisRange**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    double x[NUM], y[NUM];
    int i;
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.axisRange(PLOT_AXIS_X, -30, 390);
    plot.ticsRange(PLOT_AXIS_X, 30, -30, 390);
    plot.axisRange(PLOT_AXIS_Y, -1, 1);
    plot.ticsRange(PLOT_AXIS_Y, .25, -1, 1);
    plot.axis(PLOT_AXIS_X, PLOT_OFF);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



CPlot::axisRange

Synopsis in Ch
#include <chplot.h>
void axisRange(int *axis*, **double** *minimum*, **double** *maximum*, ... /* [**double** *incr*] */);

Synopsis in C++
#include <chplot.h>
void axisRange(int *axis*, **double** *minimum*, **double** *maximum*);
void axisRange(int *axis*, **double** *minimum*, **double** *maximum*, **double** *incr*);

Syntax in Ch and C++
axisRange(*axis*, *minimum*, *maximum*)
axisRange(*axis*, *minimum*, *maximum*, *incr*)

Purpose
Specify the range for an axis.

Return Value
None.

Parameters
axis The *axis* parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

minimum The axis minimum.

maximum The axis maximum.

incr The increment between tic marks. By default or when *incr* is 0, the increment between tic marks is calculated internally.

Description

The range for an axis can be explicitly specified with this function. Autoscaling for the specified axis is disabled and any previously specified labeled tic-marks are overridden. If the axis is logarithmic specified by the member function **scaleType**(), the increment will be used as a multiplicative factor.

Note that

```
    plot.axisRange(axis, min, max, incr);
```

is obsolete. Use

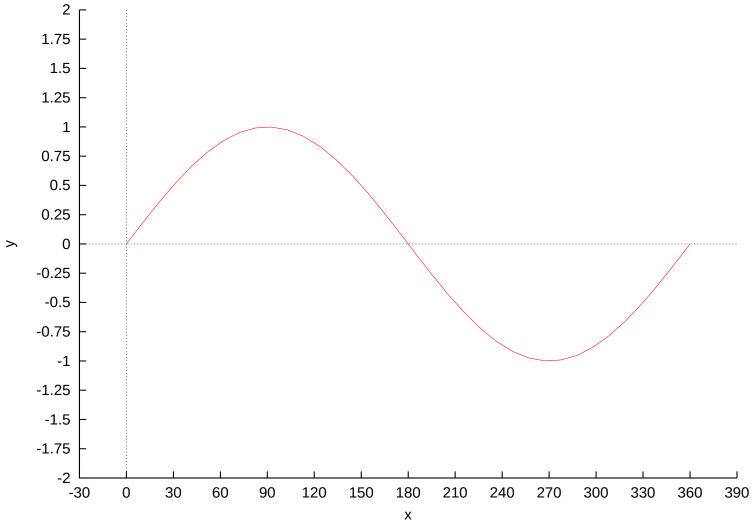
```
    plot.axisRange(axis, min, max);
    plot.ticsRange(axis, incr);
or
    plot.ticsRange(axis, incr, start);
    plot.ticsRange(axis, incr, start, end);
```

Example 1

Compare with the output for the examples in **CPlot**::**axis**() and **CPlot**::**grid**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.axisRange(PLOT_AXIS_X, -30, 390);
    plot.ticsRange(PLOT_AXIS_X, 30);
    plot.axisRange(PLOT_AXIS_Y, -2, 2);
    plot.ticsRange(PLOT_AXIS_Y, 0.25);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

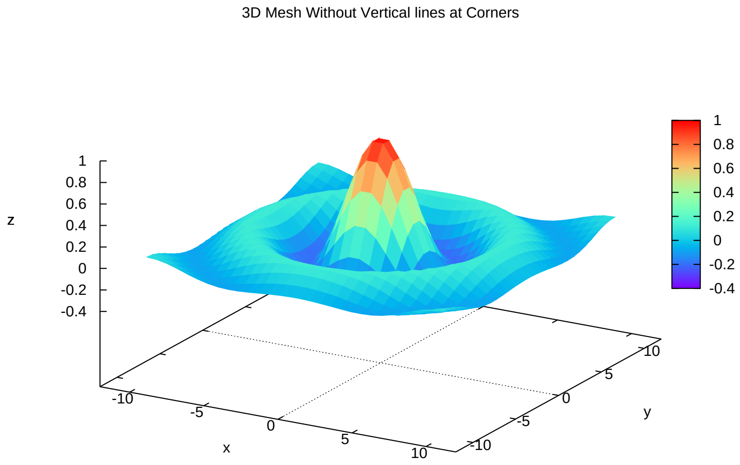
Output



Example 2
3D mesh plot without vertical lines at the corners. Compare with the output for examples in **CPlot::data3D()** and **CPlot::data3DSurface()**.

```
#include <chplot.h>
#include <math.h>

#define NUMX 30
#define NUMY 30

int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double r;
    int i, j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -10 + i*20.0/(NUMX-1); // linspace(x, -10, 10);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -10 + i*20.0/(NUMY-1); // linspace(y, -10, 10);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[NUMY*i+j] = sin(r)/r;
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.axisRange(PLOT_AXIS_XY, -12, 12, 5);
    plot.title("3D Mesh Without Vertical lines at Corners");
    plot.plotting();
    return 0;
}
```

Output

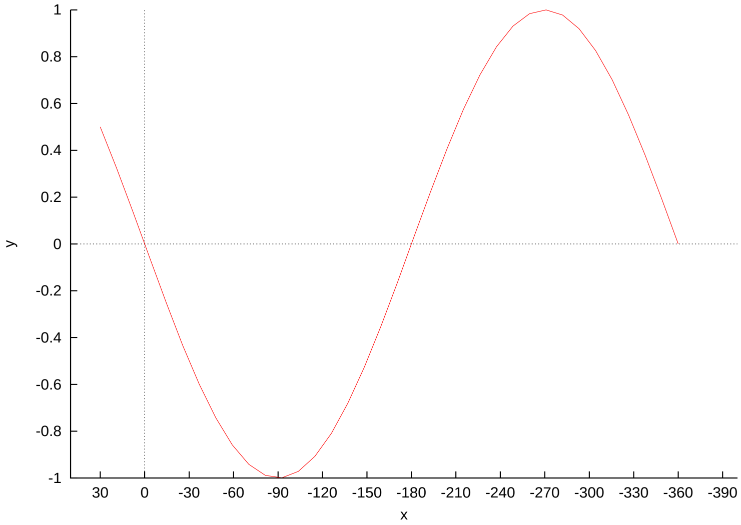


Example 3

X axis range is reversed.

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= -360 + i*390.0/(NUM-1); // linspace(x, -360, 30)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.axisRange(PLOT_AXIS_X, 50, -400);
    plot.ticsRange(PLOT_AXIS_X, -30);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::ticsRange(), **CPlot::autoScale**(), **CPlot::ticsLabel**().

CPlot::axes

Synopsis in Ch
#include <chplot.h>
void axes(int *num*, **string_t** *axes*);

Synopsis in C++
#include <chplot.h>
void axes(int *num*, **char** * *axes*);

Syntax
axes(*num*, *axes*)

Purpose
Specify the axes for a data set.

Return Value
None.

Parameters

num The data set the axes are specified.

axes The axes for the specified data set.

Description

A **CPlot** class lets you use each of the four borders – x (bottom), x2 (top), y (left) and y2 (right) – as an independent axis. The **axes**() function lets you choose which pair of axes a given set of data specified in `num` is plotted against.

There are four possible sets of axes available. The argument `axes` is used to select the axes for which a particular line should be scaled. The string `"x1y1"` refers to the axes on the bottom and left; `"x2y2"` to those on the top and right; `"x1y2"` to those on the bottom and right; and `"x2y1"` to those on the top and left.

Other options such as labels and ranges can be specified other member functions by selecting a proper axis with one of following macros.

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

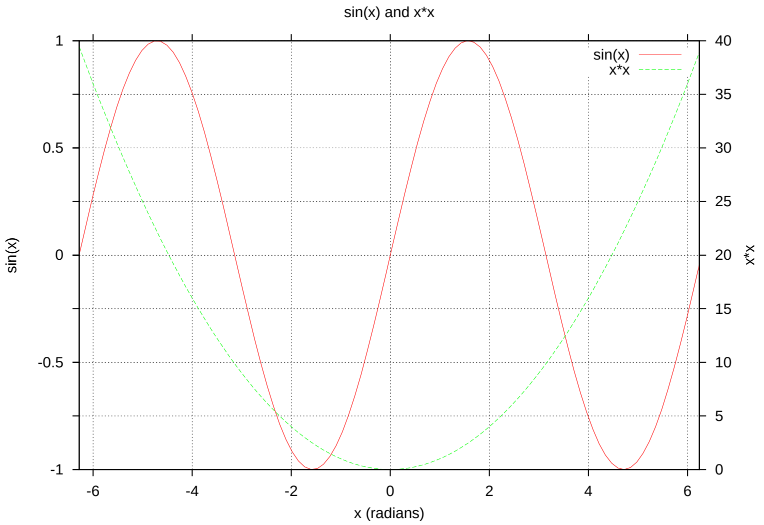
PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

Example 1

```
#include <chplot.h>
#include <math.h>

double func1(double x) {
   return sin(x);
}
double func2(double x) {
   return x*x;
}

int main() {
    class CPlot plot;
    double x0= -6.28, xf = 6.24;
    int num = 100;

    plot.title("sin(x) and x*x");
    plot.label(PLOT_AXIS_X, "x (radians)");
    plot.label(PLOT_AXIS_Y, "sin(x)");
    plot.label(PLOT_AXIS_Y2, "x*x");
    plot.axisRange(PLOT_AXIS_X, x0, xf);
    plot.ticsMirror(PLOT_AXIS_X, PLOT_ON);
    plot.axisRange(PLOT_AXIS_X2, x0, xf);
    plot.ticsRange(PLOT_AXIS_Y, 0.5);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_Y2, PLOT_ON);
    plot.ticsDirection(PLOT_TICS_OUT);
    plot.tics(PLOT_AXIS_Y2, PLOT_ON);
    plot.ticsRange(PLOT_AXIS_Y2, 5);
    plot.grid(PLOT_ON, "x y y2");
    plot.func2D(x0, xf, num, func1);
    plot.legend("sin(x)", 0);
    plot.func2D(x0, xf, num, func2);
    plot.legend("x*x", 1);
    plot.axes(1, "x2y2");
    plot.plotting();
}
```

Output



Example 2

See an example on page 183 for plot type **PLOT_PLOTTYPE_FINANCEBARS** in **CPlot**:**plotType**(). In that example, the y2 axis is used to display different data.

See Also

CPlot::**legend**(), **CPlot**::**axisRange**().

CPlot::barSize

Synopsis

#include <**chplot.h**>
void barSize(**double** *size*);

Syntax

barSize(*size*)

Purpose

Set the size of error bars.

Return Value

None.

Parameters

size The size of error bars. The value for `size` is in the range [0, 1.0].

Description

This function specifies the size of of error bars for a plot type of **PLOT_PLOTTYPE_XERRORBARS**, **PLOT_PLOTTYPE_XYERRORBARS**, and

PLOT_PLOTTYPE_YERRORBARS. specified by member function **CPlot**::**plotType**(). The value for `size` is in the range [0, 1.0]. The value 0 is for no error bar. The value 1.0 is for the largest error bar.

Example

See an example on page 183 for the plot type **PLOT_PLOTTYPE_XYERRORBARS** in **CPlot**::**plotType**().

See Also

CPlot::**boxWidth**(), **CPlot**::**plotType**().

CPlot::border()

Synopsis

#include <**chplot.h**>

void border(**int** *location*, **int** *flag*);

Purpose

Enable or disable display of a bounding box around the plot.

Return Value

None.

Parameter

location The location of the effected border segment.

- **PLOT_BORDER_BOTTOM** The bottom of the plot.
- **PLOT_BORDER_LEFT** The left side of the plot.
- **PLOT_BORDER_TOP** The top of the plot.
- **PLOT_BORDER_RIGHT** The right side of the plot.
- **PLOT_BORDER_ALL** All sides of the plot.

flag Enable or disable display of a box around the boundary or the plot.

- **PLOT_ON** Enable drawing of the box.
- **PLOT_OFF** Disable drawing of the box.

Description

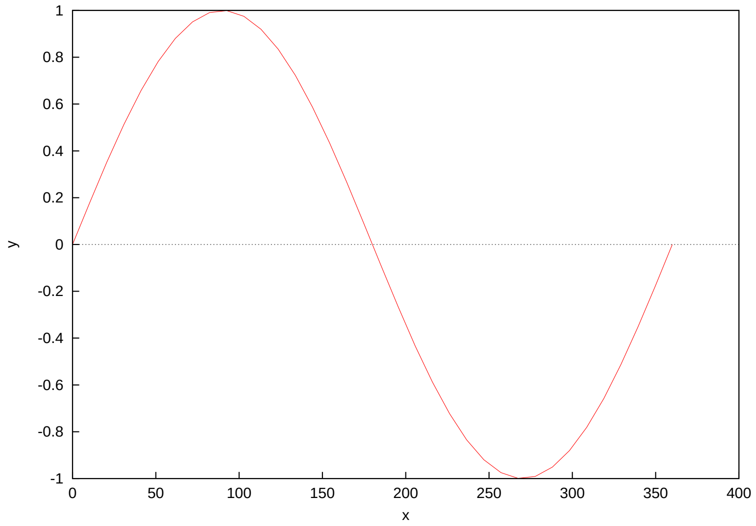
This function turns the display of a border around the plot on or off. By default, the border is drawn on the left and bottom sides for 2D plots, and on all sides for 3D plots.

Example 1

Compare with the example output in **CPlot**::**ticsMirror**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;
    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.data2DCurve(x, y, NUM);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.plotting();
    return 0;
}
```

Output

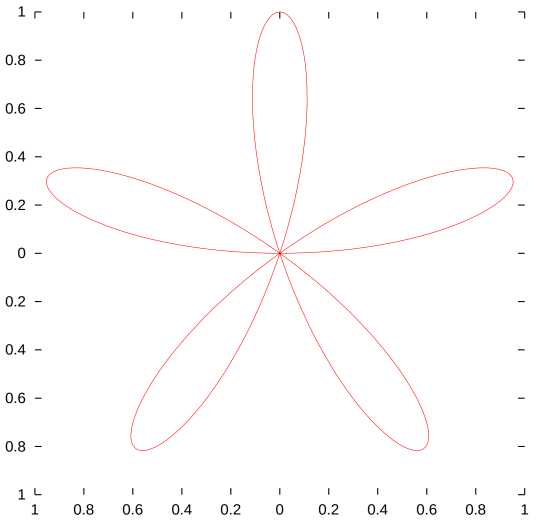


Example 2

Compare with the example output in **CPlot**::**polarPlot**().

```
#include <math.h>
#include <chplot.h>

#define NUM 360
int main() {
    int i;
    double theta[NUM], r[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       theta[i]= 0 + i*M_PI/(NUM-1); // linspace(theta, 0, M_PI);
       r[i] = sin(5*theta[i]); // Y-axis data.
    }
    plot.polarPlot(PLOT_ANGLE_RAD);
    plot.data2DCurve(theta, r, NUM);
    plot.label(PLOT_AXIS_XY, NULL);
    plot.sizeRatio(1);
    plot.border(PLOT_BORDER_ALL, PLOT_OFF);
    plot.tics(PLOT_AXIS_XY, PLOT_OFF);
    plot.axis(PLOT_AXIS_XY, PLOT_OFF);
    plot.plotting();
    return 0;
}
```

Output



CPlot::borderOffsets

Synopsis
#include <**chplot.h**>
void borderOffsets(**double** *left*, **double** *right*, **double** *top*, **double** *bottom*);

Purpose
Specify the size of offsets around the data points of a 2D plot in the same units as the plot axis.

Return Value
None.

Parameters

left The offset on the left side of the plot.

right The offset on the right side of the plot.

top The offset on the top of the plot.

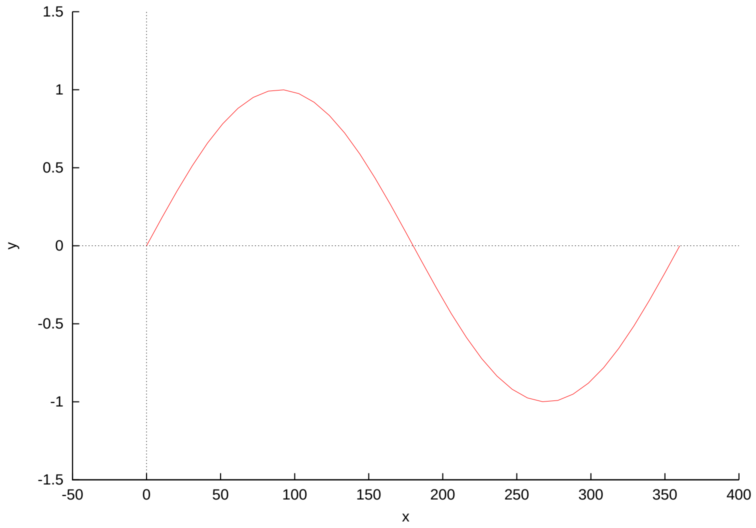
bottom The offset on the bottom of the plot.

Description
For 2D plots, this function specifies the size of offsets around the data points of an autoscaled plot. This function can be used to create empty space around the data. The *left* and *right* arguments are specified in the units of the x-axis. The *top* and *bottom* arguments are specified in the units of the y-axis.

Example
Compare with the output for examples in **CPlot::data2D**() and **CPlot::data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.data2DCurve(x, y, NUM);
    plot.borderOffsets(30, 30, .5, .25);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::autoScale(), **CPlot::axisRange**(), **CPlot::ticsLabel**(), **CPlot::margins**().

CPlot::boxBorder

Synopsis in Ch
#include <chplot.h>
void boxBorder(int *num*, ... /* [**int** *linetype*] */);

Synopsis in C++
#include <chplot.h>
void boxBorder(int *num*);
void boxBorder(int *num*, **int** *linetype*);

Syntax in Ch and C++
boxBorder(*num*)
boxBorder(*num*, *linetype*)

Purpose
Enable or disable drawing of a border for a plot of box type.

Return Value
None.

Parameters

num The data set of plot type of box or filled curve to be bounded with a border. If `num` is -1, the border handling will be applied to all boxes and filled curves for the plot.

linetype The *linetype* parameter specifies the line type used to draw bounding lines.

Description
This function specifies how borders for boxes and filled curves will handled for a plot type of **PLOT_PLOTTYPE_BOXES**, **PLOT_PLOTTYPE_BOXERRORBARS**, **PLOT_PLOTTYPE_BOXXYERRORBARS**, **PLOT_PLOTTYPE_CANDLESTICKS**, and **PLOT_PLOTTYPE_FILLEDCURVES** specified by member function **CPlot**::**plotType**(). By default, the box or filled curve is bounded by a solid line of the current line type. If the line type is not specified by the function call of **boxBorder**(*num*), no bounding lines are drawn.

Example
See an example on page 74 for **CPlot**:**boxFill**().

See Also
CPlot::**boxFill**(), **CPlot**::**boxWidth**(), **CPlot**::**plotType**().

CPlot::boxFill

Synopsis in Ch
#include <chplot.h>
void boxFill(int *num*, **int** *style*, ... /* [**double** *density*], **int** *patternnum*] */);

Synopsis in C++
#include <chplot.h>
void boxFill(int *num*, **int** *style*);
void boxFill(int *num*, **int** *style*, **double** *density*);
void boxFill(int *num*, **int** *style*, **int** *patternnum*);

Syntax in Ch and C++
boxFill(*num*, *PLOT_BOXFILL_EMPTY*)
boxFill(*num*, *PLOT_BOXFILL_SOLID*)
boxFill(*num*, *PLOT_BOXFILL_SOLID*, `density`)
boxFill(*num*, *PLOT_BOXFILL_PATTERN*)
boxFill(*num*, *PLOT_BOXFILL_PATTERN*, *patternnum*)

Purpose
Fill a box or filled curve with a solid color or pattern.

Return Value

None.

Parameters

num The data set of plot type of box or filled curve to be filled. If `num` is -1, the fill style will be applied to all boxes and filled curves for the plot.

style The *style* parameter can take one of the following values:

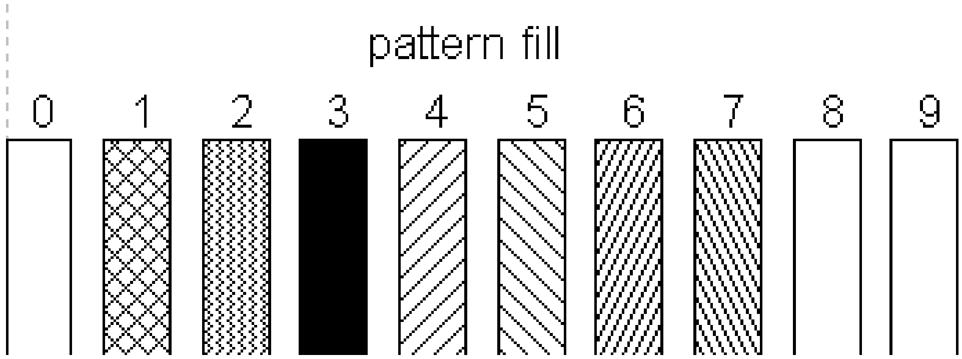
PLOT_BOXFILL_EMPTY Do not fill the box. The default is to have an empty box.

PLOT_BOXFILL_SOLID Fill the boxes or filled curves with a solid color.

PLOT_BOXFILL_PATTERN Fil the boxes or filled curves with a pattern.

density The density of solid color in the range of [0, 1.0]. If the value for `density` is 0.0, the box is empty. If it is 1.0, the inner area is of the same color as the current line type. If no `density` parameter is given, it defaults to 1.0.

patternnum The paramenter`patternnum` option causes filling to be done with a fill pattern supplied by the terminal driver. The kind and number of available fill patterns depend on the terminal driver. If multiple datasets using filled boxes are plotted, the pattern cycles through all available pattern types, starting from `patternnum`, much as the line type cycles for multiple line plots. The available patterns in Windows are shown below.



Description

This function specifies how boxes and filled curves are filled with a solid color or pattern. The fill style can be applied to plot types of **PLOT_PLOTTYPE_BOXES**, **PLOT_PLOTTYPE_BOXERRORBARS**, **PLOT_PLOTTYPE_BOXXYERRORBARS**, **PLOT_PLOTTYPE_CANDLESTICKS**, and **PLOT_PLOTTYPE_FILLEDCURVES** specified by member function **CPlot**::**plotType**().

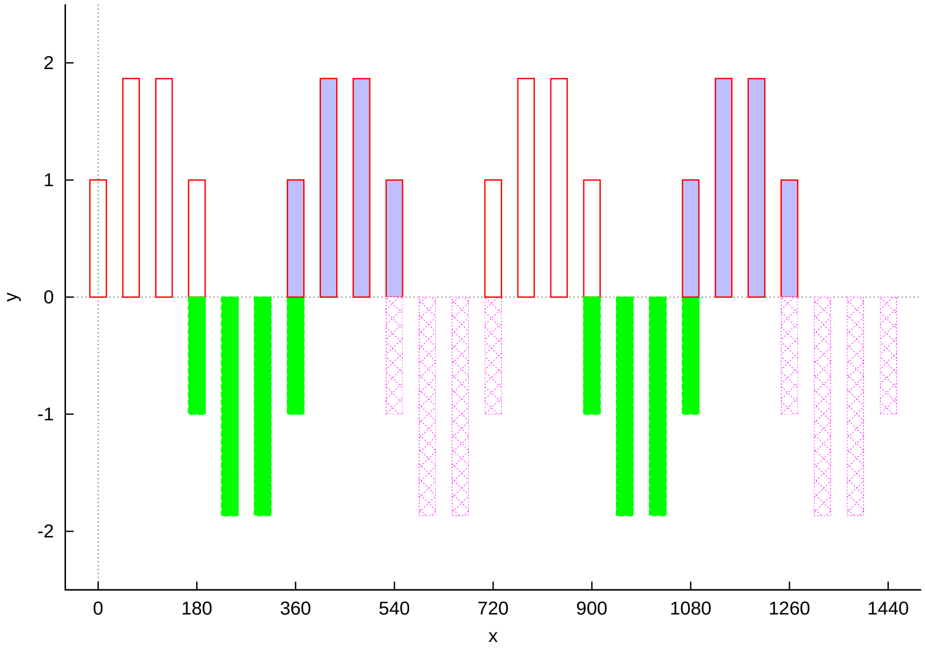
Example

In this example, each box has width of 30 specified by **CPlot**::**boxWidth**(). The first box is by default empty. The second box is filled with a solid color. The third one is filled with a solid color of density 0.25. The border of this box uses the line type for the first box specified by **CPlot**::**boxBorder**(). The fourth box is filled with a pattern. The border of this box is empty. The four boxes are repeated twice by a outer loop with index j.

```
#include <math.h>
#include <chplot.h>

#define N 4
```

```
#define M 2
#define NUM 4

int sign(double x) {
    if(x>0)
      return 1;
    else if (x <0)
      return -1;
    else
      return 0;
}

int main() {
    int i, j, k, linetype, linewidth, patternnum=1;
    double x[NUM], y[NUM];
    class CPlot plot;
    double begin;

    begin = 0.01;
    for(j=0; j< M; j++) {
      for(i=0; i<N; i++) {
        for(k=0; k<NUM; k++) {
          // lindata(begin, begin+180-0.01, x);
          x[k] = begin + k *(180-0.01)/(NUM-1);
          y[k] = sin(x[k]*M_PI/180);
          y[k] += sign(y[k])*1;
        }
        plot.data2DCurve(x, y, NUM);
        linetype = i+1;
        linewidth = 2;
        plot.plotType(PLOT_PLOTTYPE_BOXES, i+j*N, linetype, linewidth);
        plot.boxWidth(30);
        if(i==1)
           plot.boxFill(i+j*N, PLOT_BOXFILL_SOLID);
        else if(i==2)  {
           plot.boxFill(i+j*N, PLOT_BOXFILL_SOLID, 0.25);
           plot.boxBorder(i+j*N, 1);
        }
        else if(i==3)  {
           plot.boxBorder(i+j*N);
           plot.boxFill(i+j*N, PLOT_BOXFILL_PATTERN, patternnum);
        }
        begin += 180;
      }
    }
    plot.axisRange(PLOT_AXIS_X, -60, N*M*180+60);
    plot.ticsRange(PLOT_AXIS_X, 180, 0, N*M*180);
    plot.axisRange(PLOT_AXIS_Y, -2.5, 2.5);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::boxBorder(), **CPlot::boxWidth()**, **CPlot::plotType()**.

CPlot::boxWidth

Synopsis
#include <**chplot.h**>
void boxWidth(double *width*);

Syntax
boxWidth(*width*)

Purpose
Set the width for a box.

Return Value
None.

Parameters
width The width of boxes to be drawon.

Description
This function specifies the width of boxes for a plot type of **PLOT_PLOTTYPE_BOXES**, **PLOT_PLOTTYPE_BOXERRORBARS**, **PLOT_PLOTTYPE_BOXXYERRORBARS**, and **PLOT_PLOTTYPE_CANDLESTICKS** specified by member function **CPlot::plotType()**.

By default, adjacent boxes are extended in width until they touch each other. A different default width may be specified using the this member function.

The parameter `width` for the boxwidth is interpreted as being a number of units along the current x axis. If the x axis is a log-scale then the value of boxwidth is absolute only at x=1; this physical width is maintained everywhere along the axis (i.e. the boxes do not become narrower the value of x increases). If the range spanned by a log scale x axis is far from x=1, some experimentation may be required to find a useful value of boxwidth.

The default is superseded by explicit width information taken from an extra data column for a plot type of **PLOT_PLOTTYPE_BOXES**, **PLOT_PLOTTYPE_BOXERRORBARS**, or **PLOT_PLOTTYPE_BOXXYERRORBARS**, In a four-column data set, the fourth column will be interpreted as the box width unless the `width` is set to -2.0, in which case the width will be calculated automatically.

To set the box width to automatic for four-column data, use

```
plot.boxwidth(-2);
```

The same effect can be achieved with the option of `using` keyword for member function **CPlot**::**dataFile**() as follows.

```
plot.dataFile(datafile, "using 1:2:3:4:(-2)");
```

To set the box width to an absolute value of 30, use

```
plot.boxWidth(2);
```

Example
See an example on page 74 for **CPlot**:**boxFill**().

See Also
CPlot::**boxBorder**(), **CPlot**::**boxFill**(), **CPlot**::**plotType**().

CPlot::changeViewAngle

Synopsis
#include <**chplot.h**>
void changeViewAngle(**double** *x_rot*, **double** *z_rot*);

Purpose
Change the viewpoint for a 3D plot.

Return Value
None.

Parameters
x_rot The angle of rotation for the plot (in degrees) along an axis horizontal axis of the screen.

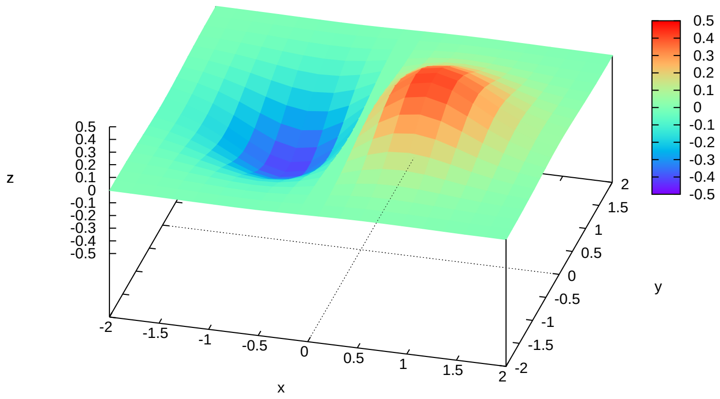
z_rot The angle of rotation for the plot (in degrees) along an axis perpendicular to the screen.

Description
This function provides rotation of a 3D plot. *x_rot* and *z_rot* are bounded by the range [0:180] and the range [0:360] respectively. By default, 3D plots are displayed with *x_rot* $= 60^\circ$ and *z_rot* $= 30^\circ$.

Example
Compare with the output for examples in **CPlot**::**data3D**() and **CPlot**::**data3DSurface**().

```
#include <math.h>
#include <chplot.h>

#define NUMX 20
#define NUMY 20
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    int i, j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -2 + i*4.0/(NUMX-1); // linspace(x, -2, 2);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -2 + i*4.0/(NUMY-1); // linspace(y, -2, 2);
    }
    for (i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[i*NUMY+j] = x[i]*exp(-x[i]*x[i]-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.changeViewAngle(45, 15);
    plot.plotting();
    return 0;
}
```

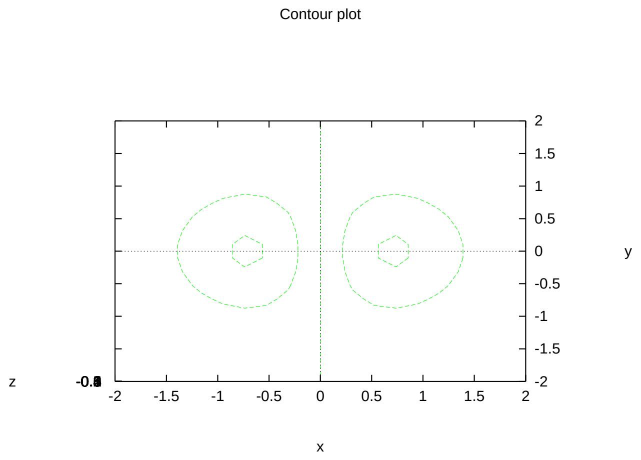
Output



Example

```
#include <chplot.h>
#include <math.h>

#define NUMX 20
#define NUMY 20
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    int datasetnum =0, i, j;
    int line_type = 1, line_width = 1;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -2 + i*4.0/(NUMX-1); // linspace(x, -2, 2);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -2 + i*4.0/(NUMY-1); // linspace(y, -2, 2);
    }
    for (i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[i*NUMY+j] = x[i]*exp(-x[i]*x[i]-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.changeViewAngle(0,0);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.showMesh(PLOT_OFF);
    plot.title("Contour plot");
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::data3D(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**().

CPlot::circle

Synopsis

#include <chplot.h>
int circle(double *x_center*, **double** *y_center*, **double** *r*);

Purpose

Add a circle to a 2D plot.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

x_center The x coordinate of the center of the circle.

y_center The y coordinate of the center of the circle.

r The radius of the circle.

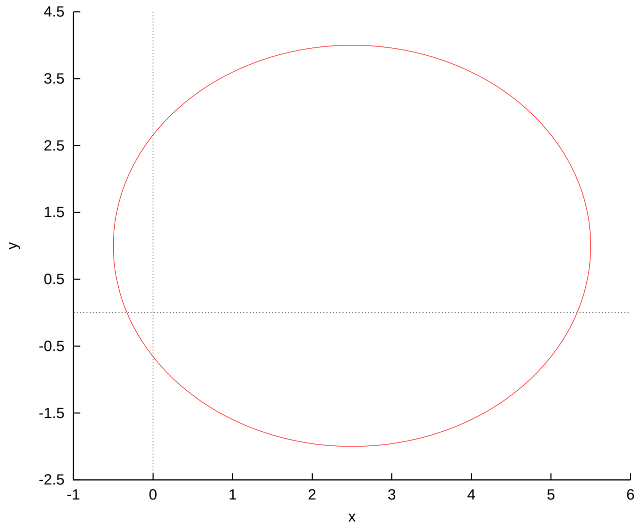
Description

This function adds a circle to a 2D plot. It is a convenience function for creation of geometric primitives. A circle added with this function is counted as a data set for later calls to **CPlot**::**legend**() and **CPlot**::**plotType**(). For rectangular plots, *x* and *y* are the coordinates of the center of the circle and *r* is the radius of the circle, all specified in units of the x and y axes. For polar plots, the location of the center of the circle is specified in polar coordinates where *x* is θ and *y* is r.

Example 1

```
#include <chplot.h>

int main() {
    double x_center = 2.5, y_center = 1.0, r = 3;
    class CPlot plot;

    plot.circle(x_center, y_center, r);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_Y, -2.5, 4.5, 1.0);
    plot.plotting();
    return 0;
}
```

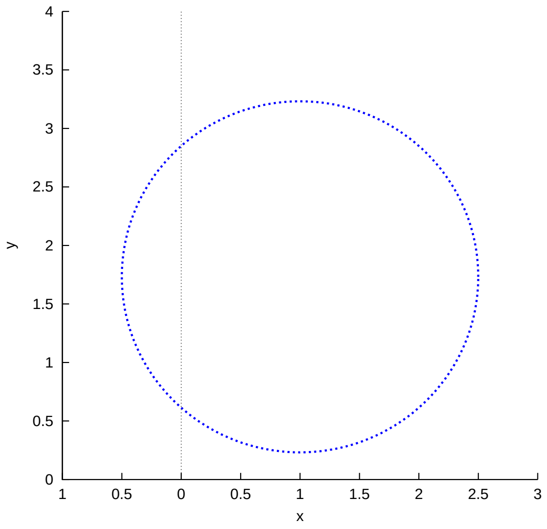
Output



Example 2

```
#include <chplot.h>
#include <math.h>

int main() {
    double x_center = M_PI/3, y_center = 2.0, r = 1.5;
    class CPlot plot;

    plot.polarPlot(PLOT_ANGLE_RAD); /* Polar coordinate system */
    plot.circle(x_center, y_center, r);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.lineType(0, 3, 4);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_X, -1, 3, 0.5);
    plot.axisRange(PLOT_AXIS_Y, 0, 4, 0.5);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::data2D(), **CPlot::data2DCurve**(), **CPlot::line**(), **CPlot::outputType**(), **CPlot::plotType**(), **CPlot::point**(), **CPlot::polygon**(), **CPlot::rectangle**().

CPlot::colorBox

Synopsis
#include <chplot.h>
void colorBox(int *flag*);

Purpose
Enable or disable the drawing of a color box for a 3D surface plot.

Return Value
None.

Parameter

flag This parameter can be set to:

PLOT_ON Enable the color box.

PLOT_OFF Disable the color box.

Description

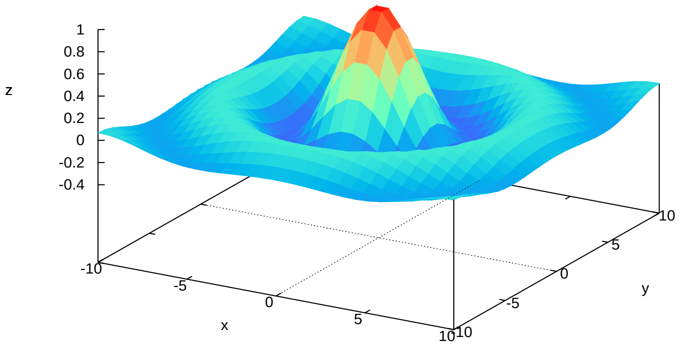
Enable or disable the drawing of a color box, which contains the color scheme, i.e. the gradient of the smooth color with the maximum and minimum values of the color palette. This is applicable only for 3D surface plots. By default, the color box is drawn.

Example 1

Compare with the output for the example in **CPlot**::**data3D**().

```
#include <chplot.h>
#include <math.h>

#define NUMX 30
#define NUMY 30

int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double r;
    int i, j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -10 + i*20.0/(NUMX-1); // linspace(x, -10, 10);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -10 + i*20.0/(NUMY-1); // linspace(y, -10, 10);
    }

    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[NUMY*i+j] = sin(r)/r;
        }
    }

    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
}
```

Output



See Also
CPlot::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**contourMode**(), **CPlot**::**plotType**().

CPlot::contourLabel

Synopsis
#include <**chplot.h**>
void contourLabel(int *flag*);

Purpose
Set display of contour line elevation labels for 3D plots on or off.

Return Value
None.

Parameter
flag Enable or disable drawing of contour line labels.

PLOT_ON Enable contour labels.

PLOT_OFF Disable contour labels.

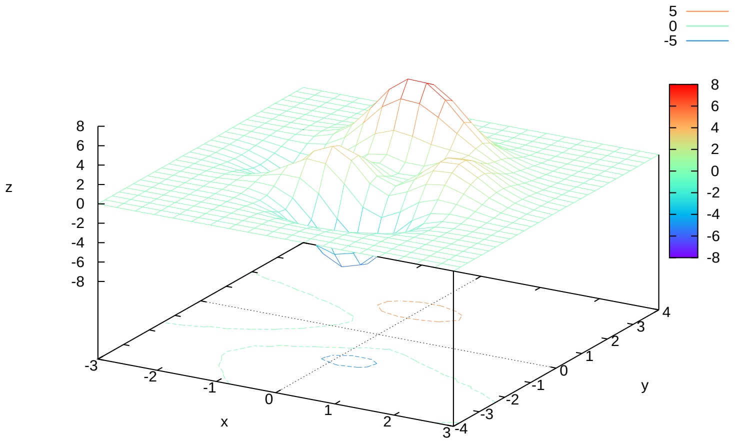
Description
Enable or disable contour line labels for 3D surface plots. labels appear with the plot legend. By default, labels for contours are not displayed.

Example 1
Compare with the output for examples in **CPlot**::**data3D**(), **CPlot**::**data3DSurface**(). **CPlot**::**contourLevels**(), and **CPlot**::**showMesh**().

```
#include <math.h>
#include <chplot.h>

#define NUMX 20
```

```
#define NUMY 30

int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    int datasetnum =0, i, j;
    int line_type = 1, line_width = 1;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -3 + i*6.0/(NUMX-1); // linspace(x, -3, 3);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -4 + i*8.0/(NUMY-1); // linspace(y, -4, 4);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[NUMY*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width);
    plot.contourLabel(PLOT_ON);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.plotting();
    return 0;
}
```

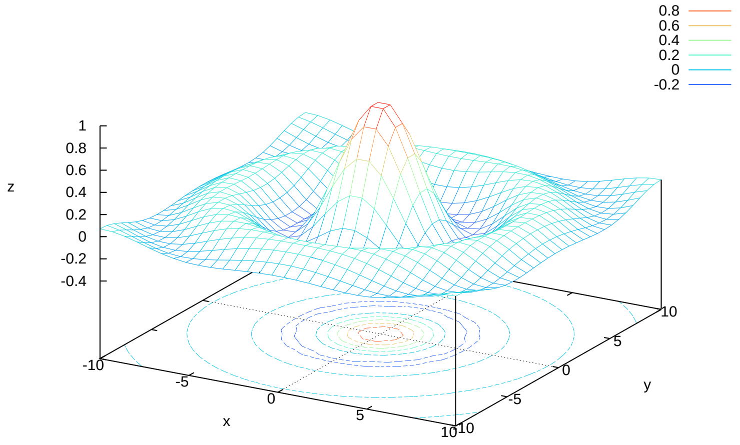
Output



Example 2

```
#include <math.h>
#include <chplot.h>

#define NUMX 30
#define NUMY 30
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double r;
    int datasetnum =0, i, j;
    int line_type = 1, line_width = 1;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -10 + i*20.0/(NUMX-1); // linspace(x, -10, 10);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -10 + i*20.0/(NUMY-1); // linspace(y, -10, 10);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[NUMY*i+j] = sin(r)/r;
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width);
    plot.contourLabel(PLOT_ON);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::data3D(), **CPlot::contourLevels**(), **CPlot::contourMode**(), **CPlot::legend**(), **CPlot::showMesh**().

CPlot::contourLevels

Synopsis in Ch

#include <chplot.h>

void contourLevels(**double** *levels*[:], ... /* [**int** *n*] */);

Synopsis in C++
#include <**chplot.h**>
void contourLevels(**double** *levels*[], **int** *n*);

Syntax in Ch
contourLevels(*level*)
contourLevels(*level*, *n*)

Syntax in C++
contourLevels(*level*, *n*)

Purpose
Set contour levels for 3D plot to be displayed at specific locations.

Return Value
None.

Parameter
levels An array of z-axis levels for contours to be drawn.

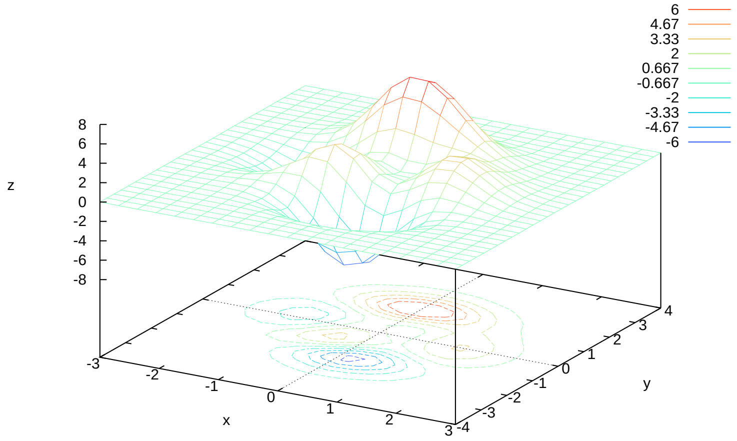
n The number of elements of array *levels*.

Description
This function allows contour levels for 3D grid data to be displayed at any desired z-axis position. The contour levels are stored in an array where the lowest contour is in the first array element.

Example 1
Compare with the output from the example in **CPlot**::**contourLabel**().

```
#include <math.h>
#include <chplot.h>

#define NUMX 20
#define NUMY 30
#define NUMLV 10
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double levels[NUMLV];
    int datasetnum =0, i, j;
    int line_type = 1, line_width = 1;
    class CPlot plot;

    for(i=0; i<NUMLV; i++) {
       levels[i]= -6 + i*12.0/(NUMLV-1); // linspace(levels, -6, 6);
    }
    for(i=0; i<NUMX; i++) {
       x[i]= -3 + i*6.0/(NUMX-1); // linspace(x, -3, 3);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -4 + i*8.0/(NUMY-1); // linspace(y, -4, 4);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[NUMY*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width);
    plot.contourLabel(PLOT_ON);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.contourLevels(levels, NUMLV);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
    return 0;
}
```

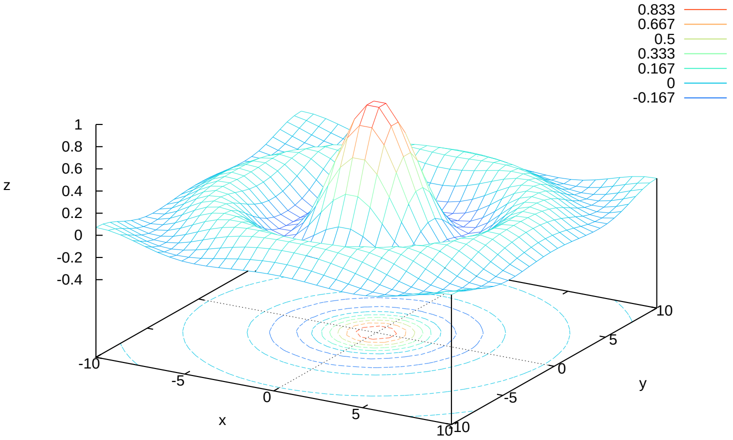
Output



Example 2

```
#include <math.h>
#include <chplot.h>

#define NUMX 30
#define NUMY 30
#define NUMLV 6

int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double levels[NUMLV];
    double r;
    int datasetnum =0, i, j;
    int line_type = 1, line_width = 1;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -10 + i*20.0/(NUMX-1); // linspace(x, -10, 10);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -10 + i*20.0/(NUMY-1); // linspace(y, -10, 10);
    }
    for(i=0; i<NUMLV; i++) {
       levels[i]= -0.2 + i*1.0/(NUMLV-1); // linspace(levels, -0.2, 0.8);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[NUMY*i+j] = sin(r)/r;
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width);
    plot.contourLabel(PLOT_ON);
    plot.contourMode(PLOT_CONTOUR_BASE);
    plot.contourLevels(levels, NUMLV);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::data3D(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **CPlot::contourLabel**(), **CPlot::contourMode**(), **CPlot::showMesh**().

CPlot::contourMode

Synopsis

#include <chplot.h>
void contourMode(int *mode*);

Purpose

Selects options for the drawing of contour lines in 3D plots.

Return Value

None.

Parameter

mode The following contour modes are available and can be combined using the logical or (|) operator:

PLOT_CONTOUR_BASE Contour lines drawn on the xy plane.

PLOT_CONTOUR_SURFACE Contour lines drawn on the surface of the plot.

Description

This option is available for display of 3D grid data.

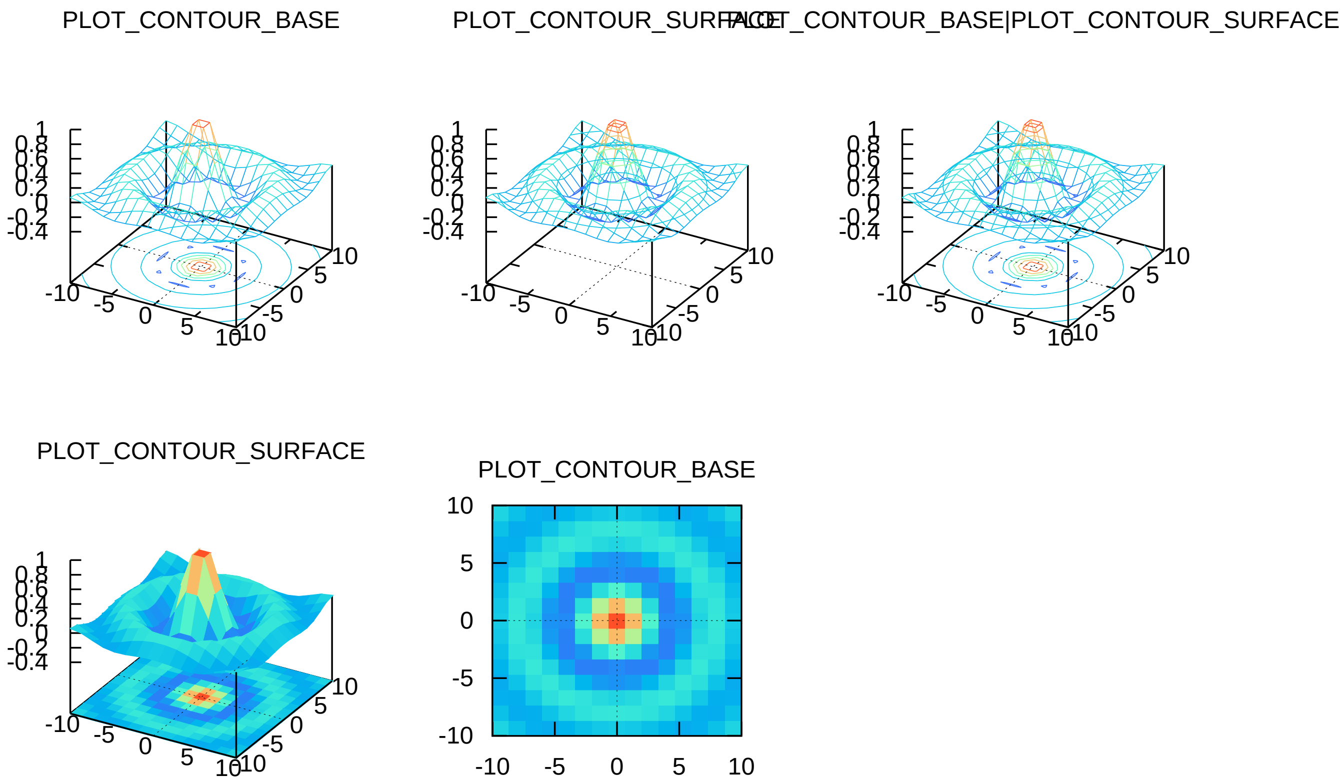
For the plot type of PLOT_PLOTTYPE_LINES, the positions of the contour levels are determined internally unless explicitly set using **CPlot**::**contourLevels**(). The **PLOT_CONTOUR_SURFACE** option does not work with hidden line removal. The hidden lines are removed by default. It can be disabled by member function **CPlot**::**removeHiddenLine**().

For the plot type of PLOT_PLOTTYPE_SURFACES, the contour mode of **PLOT_CONTOUR_SURFACE** will add a projected map on the xy plane. the contour mode of **PLOT_CONTOUR_SURFACE** will have a projected map on the xy plane only.

Example

```
#include <chplot.h>
#include <math.h>

#define NUMX 16
#define NUMY 16
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double r;
    int i, j, line_type = 1, line_width = 1;
    class CPlot subplot, *spl;

    for(i=0; i<NUMX; i++) {
       x[i]= -10 + i*20.0/(NUMX-1); // linspace(x, -10, 10);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -10 + i*20.0/(NUMY-1); // linspace(y, -10, 10);
    }

    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[NUMY*i+j] = sin(r)/r;
        }
    }
    subplot.subplot(2,3);
    spl = subplot.getSubplot(0,0);
    spl->data3DSurface(x, y, z, NUMX, NUMY);
    spl->plotType(PLOT_PLOTTYPE_LINES, 0);
    spl->lineType(0, line_type, line_width);
    spl->contourMode(PLOT_CONTOUR_BASE);
    spl->removeHiddenLine(PLOT_OFF);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_CONTOUR_BASE");
    spl = subplot.getSubplot(0,1);
    spl->data3DSurface(x, y, z, NUMX, NUMY);
    spl->plotType(PLOT_PLOTTYPE_LINES, 0);
    spl->lineType(0, line_type, line_width);
    spl->contourMode(PLOT_CONTOUR_SURFACE);
    spl->removeHiddenLine(PLOT_OFF);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_CONTOUR_SURFACE");
    spl = subplot.getSubplot(0,2);
    spl->data3DSurface(x, y, z, NUMX, NUMY);
    spl->plotType(PLOT_PLOTTYPE_LINES, 0);
    spl->lineType(0, line_type, line_width);
    spl->contourMode(PLOT_CONTOUR_BASE|PLOT_CONTOUR_SURFACE);
    spl->removeHiddenLine(PLOT_OFF);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_CONTOUR_BASE|PLOT_CONTOUR_SURFACE");
    spl = subplot.getSubplot(1,0);
    spl->data3DSurface(x, y, z, NUMX, NUMY);
    spl->contourMode(PLOT_CONTOUR_SURFACE);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_CONTOUR_SURFACE");
    spl = subplot.getSubplot(1,1);
    spl->data3DSurface(x, y, z, NUMX, NUMY);
    spl->contourMode(PLOT_CONTOUR_BASE);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_CONTOUR_BASE");

    subplot.plotting();
    return 0;
}
```

Output



See Also
CPlot::data3D(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **CPlot::contourLevels**(), **CPlot::removeHiddenLine**(), **CPlot::showMesh**().

CPlot::coordSystem

Synopsis in Ch
#include <chplot.h>
void coordSystem(int *coord_system*, ... /* [**int** *angle_unit*] */);

Synopsis in C++
#include <chplot.h>
void coordSystem(int *coord_system*);
void coordSystem(int *coord_system*, **int** *angle_unit*);

Syntax in Ch and C++
coordSystem(PLOT_COORD_CARTESIAN)
coordSystem(PLOT_COORD_SPHERICAL)
coordSystem(PLOT_COORD_SPHERICAL, PLOT_ANGLE_DEG)
coordSystem(PLOT_COORD_SPHERICAL, PLOT_ANGLE_RAD)
coordSystem(PLOT_COORD_CYLINDRICAL)
coordSystem(PLOT_COORD_CYLINDRICAL, PLOT_ANGLE_DEG)

coordSystem(**PLOT_COORD_CYLINDRICAL**, **PLOT_ANGLE_RAD**)

Purpose

Purpose

Select coordinate system for 3D plots.

Return Value

None.

Parameters

coord_system The coordinate system can be set to any of the following:

- **PLOT_COORD_CARTESIAN** Cartesian coordinate system. Each data point consists of three values (x,y,z).
- **PLOT_COORD_SPHERICAL** Spherical coordinate system. Each data point consists of three values (θ,ϕ,r).
- **PLOT_COORD_CYLINDRICAL** Cylindrical coordinates. Each data point consists of three values (θ,z,r).

angle_unit an optional parameter to specify the unit for mesurement of an angular position in **PLOT_COORD_SPHERICAL** and **PLOT_COORD_CYLINDRICAL** coordinate systems. The following options are available:

- **PLOT_ANGLE_DEG** Angles measured in degree.
- **PLOT_ANGLE_RAD** Angles measured in radian.

Description

This function selects the coordinate system for 3D plots. For a spherical coordinate system, points are mapped to Cartesian space by:

$$\begin{aligned} x &= r\cos(\theta)\cos(\phi) \\ y &= r\sin(\theta)\cos(\phi) \\ z &= r\sin(\phi) \end{aligned}$$

For a cylindrical coordinate system, points are mapped to Cartesian space by:

$$\begin{aligned} x &= r\cos(\theta) \\ y &= r\sin(\theta) \\ z &= z \end{aligned}$$

The default coordinate system is **PLOT_COORD_CARTESIAN**. For **PLOT_COORD_SPHERICAL** and **PLOT_COORD_CYLINDRICAL**, the default unit for *angle_unit* is **PLOT_ANGLE_RAD**.

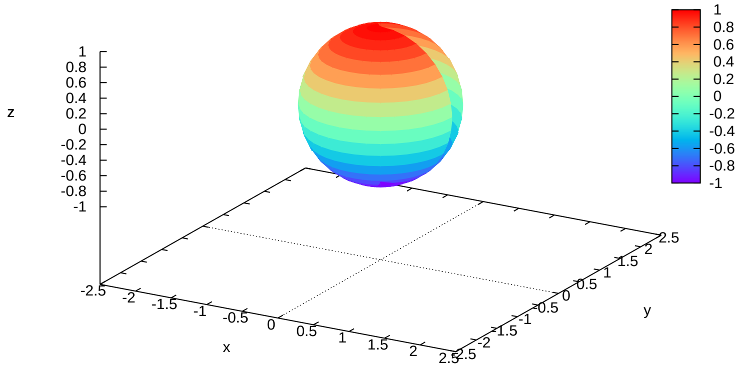
Example 1

See **CPlot**::**data3D**().

Example 2

```
#include <chplot.h>
#include <math.h>

#define NUMX 37
#define NUMY 19
int main() {
    int i;
    double theta[NUMX], phi[NUMY], r[NUMX*NUMY];
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       theta[i]= 0 + i*2*M_PI/(NUMX-1); // linspace(theta, 0, 2*M_PI)
    }

    for(i=0; i<NUMY; i++) {
       phi[i]= -M_PI/2 + i*M_PI/(NUMY-1); // linspace(phi, -M_PI/2, M_PI/2)
    }
    for(i=0; i<NUMX*NUMY; i++) {
       r[i] = 1; // r = (array double [703])1;
    }
    plot.data3DSurface(theta, phi, r, NUMX, NUMY);
    plot.coordSystem(PLOT_COORD_SPHERICAL, PLOT_ANGLE_RAD);
    plot.axisRange(PLOT_AXIS_XY, -2.5, 2.5, 0.5);
    plot.plotting();
    return 0;
}
```

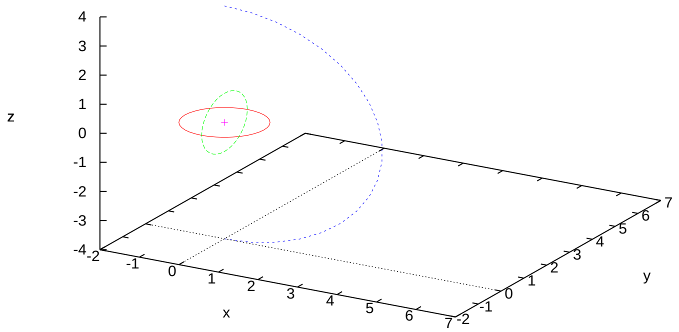
Output



Example 3

```
#include <chplot.h>
#include <math.h>

#define NUM1 37
#define NUM2 37
#define NUM3 20
int main() {
    int i;
    double theta1[NUM1], phi1[NUM1], r1[NUM1];
    double theta2[NUM2], phi2[NUM2], r2[NUM2];
    double theta3[NUM3], phi3[NUM3], r3[NUM3];
    class CPlot plot;

    for(i=0; i<NUM1; i++) {
       theta1[i]= 0 + i*2*M_PI/(NUM1-1); // linspace(theta1, 0, 2*M_PI);
       phi1[i] = 0; // phi1 = (array double [37])0;
       r1[i] = 1;   // r1 = (array double [37])1;
       theta2[i] = M_PI/2; // theta2 = (array double [37])M_PI/2;
    }
    for(i=0; i<NUM2; i++) {
       phi2[i]= 0 + i*2*M_PI/(NUM2-1); // linspace(phi2, 0, 2*M_PI);
       r2[i] = 1; // r2 = (array double [37])1;
    }
    for(i=0; i<NUM3; i++) {
       theta3[i] = 0; // theta3 =  (array double [20])0;
       phi3[i]= -M_PI/2 + i*M_PI/(NUM2-1); // linspace(phi3, -M_PI/2, M_PI/2);
       r3[i] = 4; // r3 = (array double [20])4;
    }
    plot.data3DCurve(theta1, phi1, r1, NUM1);
    plot.data3DCurve(theta2, phi2, r2, NUM2);
    plot.data3DCurve(theta3, phi3, r3, NUM3);
    plot.point(0, 0, 0);
    plot.coordSystem(PLOT_COORD_SPHERICAL, PLOT_ANGLE_RAD);
    plot.axisRange(PLOT_AXIS_XY, -2, 7, 1);
    plot.ticsLevel(0);
    plot.removeHiddenLine(PLOT_OFF);
    plot.plotting();
    return 0;
}
```

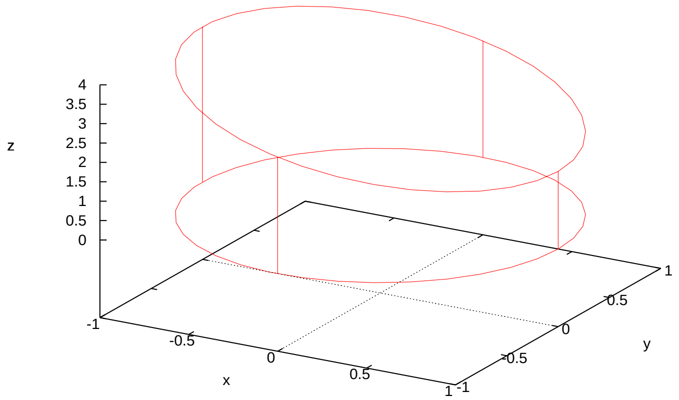
Output



Example 4

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double r1[NUM], theta1[NUM], z1[NUM];
    double r2[NUM], theta2[NUM], z2[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       theta1[i]= 0 + i*360.0/(NUM-1); // linspace(theta1, 0, 360);
       // z1=(array double [numpoints])3+
       //    sin((theta1-(array double [numpoints])90)*M_PI/180);
       z1[i] = 3 + sin((theta1[i] - 90)*M_PI/180);
       r1[i] = 1;
       theta2[i]= 0 + i*360.0/(NUM-1); // linspace(theta2, 0, 360);
       z2[i] = 0; // z2 = (array double [numpoints])0;
       r2[i] = 1; // r2=(array double [numpoints])1;
    }
    plot.data3DCurve(theta1, z1, r1, NUM);
    plot.data3DCurve(theta2, z2, r2, NUM);
    plot.coordSystem(PLOT_COORD_CYLINDRICAL, PLOT_ANGLE_DEG);
    plot.line(0, 0, 1, 0, 2, 1);
    plot.line(90, 0, 1, 90, 3, 1);
    plot.line(180, 0, 1, 180, 4, 1);
    plot.line(270, 0, 1, 270, 3, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.lineType(0, 1, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 1);
    plot.lineType(1, 1, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 2);
    plot.lineType(2, 1, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 3);
    plot.lineType(3, 1, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 4);
    plot.lineType(4, 1, 1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 5);
    plot.lineType(5, 1, 1);
    plot.plotting();
    return 0;
}
```

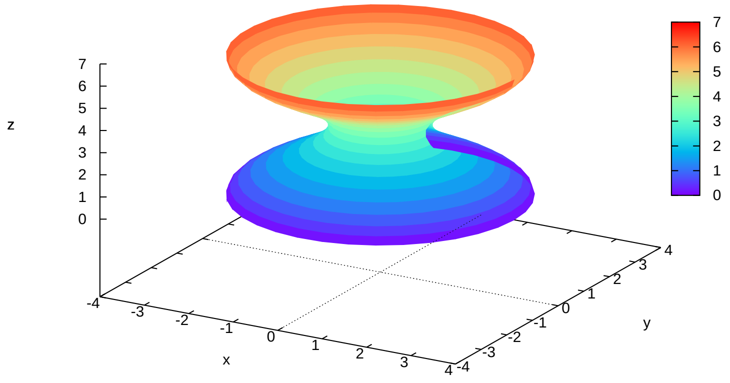
Output



Example 5

```
#include <math.h>
#include <chplot.h>

#define NUMT 36
#define NUMZ 20
int main() {
    double theta[NUMT], z[NUMZ], r[NUMT*NUMZ];
    int i, j;
    class CPlot plot;

    for(i=0; i<NUMT; i++) {
      theta[i] = 0 + i*360.0/(NUMT-1);    // linspace(theta, 0, 360)
    }
    for(i=0; i<NUMZ; i++) {
      z[i] = 0 + i*2*M_PI/(NUMZ-1);       // linspace(z, 0, 2*PI)
    }
    for(i=0; i<NUMT; i++) {
        for(j=0; j<NUMZ; j++) {
            r[i*NUMZ+j] = 2+cos(z[j]);
        }
    }
    plot.data3DSurface(theta, z, r, NUMT, NUMZ);
    plot.coordSystem(PLOT_COORD_CYLINDRICAL, PLOT_ANGLE_DEG);
    plot.axisRange(PLOT_AXIS_XY, -4, 4, 1);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::data3D(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**().

CPlot::data

Synopsis
#include <chplot.h>
int data(**void** **x*, **int** *row*, **int** `col`);

Purpose
Add 2D, 3D, or multi-dimensional data set to an instance of the **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

x A two-dimensional array of double type used for the plot.

row The number of rows for the array `x`.

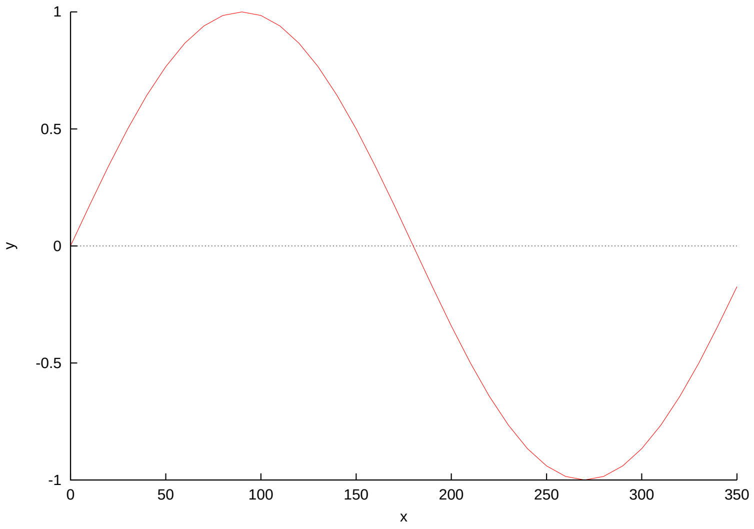
col The number of columns for the array `x`.

Description
The data for a plot can be placed in either a file or in the memory. The opaque pointer `x` is the address for a two-dimensional array of double type. The size of the array is specified by its number of rows and columns. The data with multiple columns for a plot type such as candlesticks, finance bars, boxes, etc. can be added by this member function.

Example 1
In this example, a data set for 2D plot is added.

```
#include <math.h>
#include <chplot.h>

#define ROW 36
#define COL 2
int main() {
    int i;
    double a[ROW][COL];
    class CPlot plot;

    for(i=0; i< ROW; i++) {
      a[i][0] = i*10;
      a[i][1] = sin(a[i][0]*M_PI/180);
    }
    plot.data(a, ROW, COL);
    plot.plotting();
    return 0;
}
```

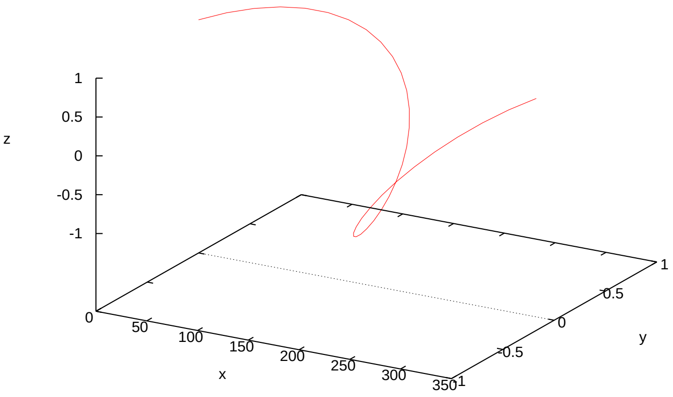
Output



Example 2 In this example, a data set for 3D plot is added.

```
#include <math.h>
#include <chplot.h>

#define ROW 36
#define COL 3
int main() {
    int i;
    double a[ROW][COL];
    class CPlot plot;

    for(i=0; i< ROW; i++) {
      a[i][0] = i*10;
      a[i][1] = sin(a[i][0]*M_PI/180);
      a[i][2] = cos(a[i][0]*M_PI/180);
    }
    plot.data(a, ROW, COL);
    plot.dimension(3);
    plot.plotting();
    return 0;
}
```

Output



Example 3 In this example, a data set for candlesticks is added by `plot.data(a, ROW, COL)` and `plot.data(b, ROW, COL)`.

```
/* File: data_2.cpp to process data in candlesticks.dat */
/*       compare with plotType_cs.ch */
#include <chplot.h>

#define ROW 10
#define COL 5
int main() {
    class CPlot plot;
    double a[ROW][COL], b[ROW][COL];
    char filename[]="candlesticks.dat";
    FILE *stream;
    int i;

    stream = fopen(filename, "r");
    if(stream == NULL) {
      fprintf(stderr, "Error: cannot open file ’%s’ for reading\n", filename);
      return -1;
    }
    /* "using 1:3:2:6:5" */
    for(i = 0; i<ROW; i++) {
      fscanf(stream, "%lf%lf%lf%*lf%lf%lf",
            &a[i][0], &a[i][2], &a[i][1], &a[i][4], &a[i][3]);
      printf("%lf %lf %lf %lf %lf\n",
            a[i][0], a[i][1], a[i][2], a[i][3], &a[i][4]);
    }
    rewind(stream);
    /* using 1:4:4:4:4" */
    for(i = 0; i<ROW; i++) {
         fscanf(stream, "%lf%*lf%*lf%lf%*lf%*lf", &b[i][0], &b[i][1]);
         b[i][2] = b[i][1];
         b[i][3] = b[i][1];
         b[i][4] = b[i][1];
         printf("%lf %lf\n", b[i][0], b[i][1]);
    }
    fclose(stream);

    plot.label(PLOT_AXIS_X, "");
    plot.label(PLOT_AXIS_Y, "");
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot.title("box-and-whisker plot adding median value as bar");
    plot.axisRange(PLOT_AXIS_X, 0, 11);
    plot.axisRange(PLOT_AXIS_Y, 0, 10);
    //plot.dataFile("candlesticks.dat", "using 1:3:2:6:5");
    //plot.legend("’candlesticks.dat’ using 1:3:2:6:5", 0);
    plot.data(a, ROW, COL);
    plot.legend("array a", 0);
    plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 0, "linetype 1 linewidth 2 whiskerbars");
    plot.boxFill(0, PLOT_BOXFILL_EMPTY);
    //plot.dataFile("candlesticks.dat", "using 1:4:4:4:4");
    //plot.legend("’candlesticks.dat’ using 1:4:4:4:4", 1);
    plot.data(b, ROW, COL);
    plot.legend("array b", 1);
    plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 1, "linetype -1 linewidth 2");
    plot.boxWidth(0.2);
    plot.plotting();
    return 0;
}
```

Output

The output is the same as that from program plotType_cs.cpp on page 181 for **CPlot**::**plotType**().

See Also

CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**dataFile**(), **CPlot**::**plotType**().

CPlot::data2DCurve

Synopsis

#include <**chplot.h**>
int data2DCurve(**double** *x*[], **double** *y*[], int n);

Purpose

Add a set of data for 2D curve to an instance of **CPlot** class.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

x a one-dimensional array of double type used for the x axis of the plot.

y a one-dimentaional array of double type for the y axis.

n number of elements of arrays *x* and *y*.

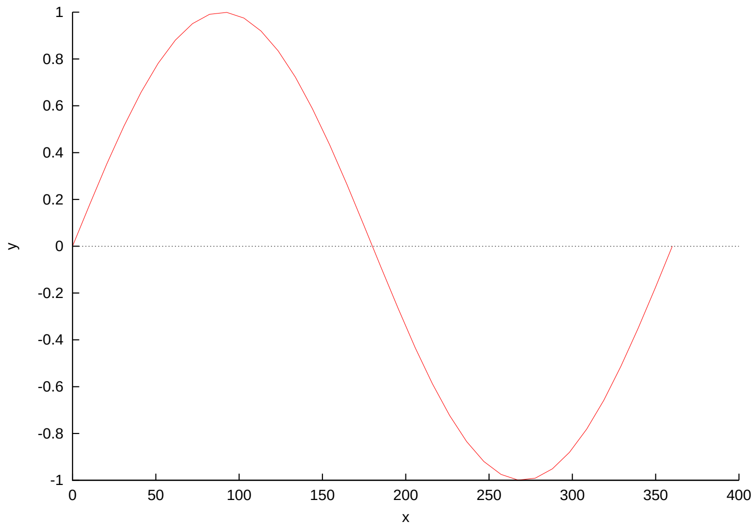
Description

This function adds the specified x-y data to a previously declared instance of the **CPlot** class. The parameter *x* is a one-dimensional array of size n. The parameter *y* is a one-dimensional array of size n. Data points with a y value of NaN are internally removed before plotting occurs. ”Holes” in a data set can be constructed by manually setting elements of *y* to this value. The plot of the data is generated using the **CPlot**::**plotting** member function.

Example 1
Compare with the output for examples in **CPlot::arrow()**, **CPlot::autoScale()**, **CPlot::borderOffsets()**, **CPlot::data2D()**, **CPlot::displayTime()**, **CPlot::label()**, **CPlot::ticsLabel()**, **CPlot::margins()**, **CPlot::ticsDirection()**, **CPlot::ticsFormat()**, **CPlot::ticsLocation()**, and **CPlot::title()**.

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i< NUM; i++) {
      x[i] = i*10;
      y[i] = sin(x[i]*M_PI/180);
    }
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::data2D(), **CPlot::data3D()**, **CPlot::data3DCurve()**, **CPlot::data3DSurface()**, **CPlot::dataFile()**, **CPlot::plotting()**, **plotxy()**.

CPlot::data3DCurve

Synopsis
#include <**chplot.h**>
int data3DCurve(double *x*[], **double** *y*[], **double** *z*[], **int** n);

Purpose

Add a set of data for 3D curve to an instance of **CPlot** class.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

x a one-dimensional array of size n used for the x axis of the plot.

y a one-dimensional array of size n used for the y axis of the plot.

z a one-dimensional array of size n used for the z axis of the plot.

n the number of elements for arrays *x*, *y*, and *z*.

Description

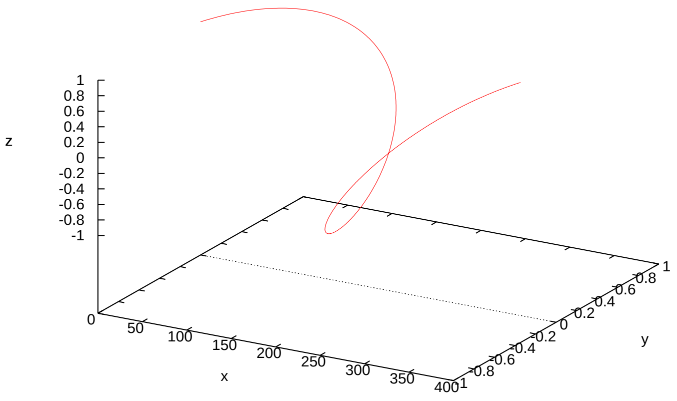
Add a set of data for 3D curve to an instance of **CPlot** class. Arrays `x`, *y*, and *z* have the same number of elements of size *n*. In a Cartesian coordinate system, these arrays represent data in X-Y-Z coordinates. In a cylindrical coordinate system, *x* represents θ, *y* for z, and *z* for r. In a spherical coordinate system, *x* represents θ, *y* for ϕ, and *z* for r. Data points with a z value of NaN are internally removed before plotting occurs. ”Holes” in a data set can be constructed by manually setting elements of *z* to this value.

Example 1

Compare with output for examples in **CPlot**::**data3D**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM], z[NUM];
    class CPlot plot;

    for(i=0; i< NUM; i++) {
      x[i] = i*10;
      y[i] = sin(x[i]*M_PI/180);
      z[i] = cos(x[i]*M_PI/180);
    }
    plot.data3DCurve(x, y, z, NUM);
    plot.plotting();
    return 0;
}
```

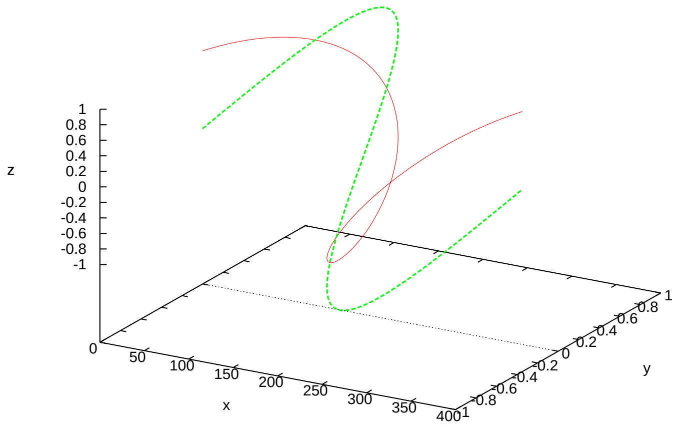
Output



Example 2
Compare with output for examples in **CPlot::data3D**().

```
#include <math.h>
#include <chplot.h>

#define NUM 360
int main() {
    double x[NUM], y[NUM], z1[NUM], z2[NUM];
    int i;
    class CPlot plot;

    for(i=0; i<360; i++) {
        x[i]  = i;
        y[i]  = sin(x[i]*M_PI/180);
        z1[i] = cos(x[i]*M_PI/180);
        z2[i] = y[i];
    }
    plot.data3DCurve(x, y, z1, NUM);
    plot.data3DCurve(x, y, z2, NUM);
    plot.plotType(PLOT_PLOTTYPE_LINES, 1);
    plot.lineType(1, 0, 3); /* set the second data set to
                               use the default line type
                               and a line width three
                               times the default */
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::data2D(), **CPlot::data2DCurve**(), **CPlot::data3D**(), **CPlot::data3DSurface**(), **CPlot::dataFile**(), **CPlot::plotting**(), **plotxyz**().

CPlot::data3DSurface

Synopsis
#include <chplot.h>
int data3DSurface(double *x*[], **double** *y*[], **double** *z*[], **int** n, **int** m);

Purpose
Add a set of data for 3D surface to an instance of **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

x a one-dimensional array of size n used for the x axis of the plot.

y a one-dimensional array of size m used for the y axis of the plot.

z a one-dimensional array of size nXm

n the number of elements for array *x*.

m the number of elements for array *y*.

Description
Add a set of data for 3D surface plot to an instance of **CPlot** class. If one-dimensional array `x` has the number of elements of size n, and *y* has size m, *z* shall be a one-dimensional array of size $n_z = n \cdot m$. In a Cartesian coordinate system, arrays *x*, *y*, and *z* represent values in X-Y-Z coordinates, respectively. In a cylindrical coordinate system, arrays *x*, *y*, and *z* represent θ), z, and r coordinates, respectively. In a spherical coordinate system, arrays *x*, *y*, and *z* represent θ), ϕ, and r coordinates, respectively. Hidden line removal is

enabled automatically (see **CPlot**::**removeHiddenLine**()). If it is desired to plot both grid data and non-grid data on the same plot, hidden line removal should be disabled manually after all data are added. Data points with a z value of NaN are internally removed before plotting occurs. ”Holes” in a data set can be constructed by manually setting elements of *z* to this value.

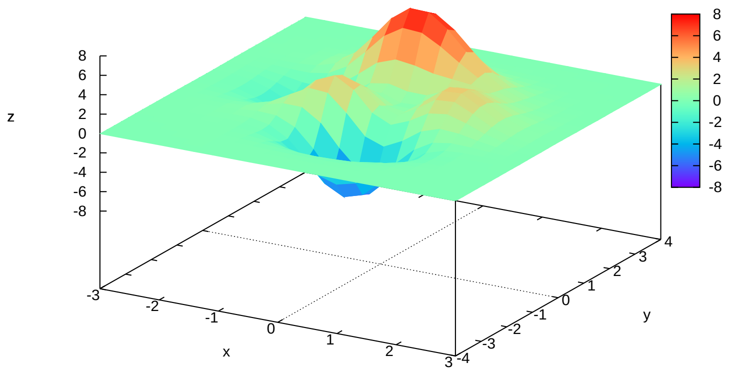
It is important to note that for a 3D grid, the ordering of the z data is important. For calculation of the z values, the *x* value is held constant while *y* is cycled through its range of values. The *x* value is then incremented and *y* is cycled again. This is repeated until all the data are calculated. So, for a 10x20 grid the data shall be ordered as follows:

x1	y1	z1
x1	y2	z2
x1	y3	z3
.		
.		
.		
x1	y18	z18
x1	y19	z19
x1	y20	z20
x2	y1	z21
x2	y2	z22
x2	y3	z23
.		
.		
.		
x10	y18	z198
x10	y19	z199
x10	y20	z200

Example 1
Compare with output for examples in **CPlot**:**data3D**(), **CPlot**::**arrow**(), **CPlot**::**contourLabel**(), **CPlot**::**grid**(), **CPlot**::**removeHiddenLine**(), **CPlot**:**size3D**(), **CPlot**::**changeViewAngle**(), and **CPlot**::**ticsLevel**().

```
#include <math.h>
#include <chplot.h>

#define N 20
#define M 30
int main() {
    double x[N], y[M], z[N*M];
    int i,j;
    class CPlot plot;

    for(i=0; i<N; i++) {
      x[i] = -3 + i*6/19.0;  // linspace(x, -3, 3)
    }
    for(i=0; i<M; i++) {
      y[i] = -4 + i*8/29.0;  // linspace(y, -4, 4)
    }
    for(i=0; i<N; i++) {
        for(j=0; j<M; j++) {
            z[M*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, N, M);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::data2D(), **CPlot::data2DCurve**(), **CPlot::data3D**(), **CPlot::data3DCurve**(), **CPlot::dataFile**(), **CPlot::plotting**(), **plotxyz**().

CPlot::dataFile

Synopsis in Ch

#include <**chplot.h**>
int dataFile(**string_t** *file*, ... /* [**char** *option*] */);

Synopsis in C++

#include <**chplot.h**>
int dataFile(**char** **file*);
int dataFile(**char** **file*, **char** *option*);

Syntax in Ch and C++

dataFile(*file*)
dataFile(*file*, *option*)

Purpose

Add a data file to an existing instance of the **CPlot** class.

Return Value

This function returns 0 on success and -1 on failure.

Parameter

file name of the file to be plotted.

option The option for the data file.

Description

Add a data file to an existing plot variable. Each data file corresponds to a single data set. The data file should be formatted with each data point on a separate line. 2D data are specified by two values per point. An empty line in a 2D data file causes a break of the curve in the plot. Multiple curves can be plotted in this manner, however the plot style will be the same for all curves. 3D data are specified by three values per data point.

For a 3D grid or surface data, each row is separated in the data file by a blank line. By default, hidden lines are not removed for 3D plotting using a data file. Use function **CPlot**::**removeHiddenLine**() to remove hidden lines.

The symbol # will comment out a subsequent text terminated at the end of a line in a data file. For example, a 3 x 3 grid would be represented as follows:

```
x1   y1   z1
x1   y2   z2
x1   y3   z3

x2   y1   z4
x2   y2   z5
x2   y3   z6

x3   y1   z7
x3   y2   z8
x3   y3   z9
```

Two empty lines in the data file will cause a break in the plot. Multiple curves or surfaces can be plotted in this manner, however, the plot style will be the same for all curves or surfaces. Member function **CPlot**::**dimension**(3) must be called before 3D data file can be added.

The option for the data file is as follows.

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

If a format is specified, each datafile record is read using the C library's **scanf**() function, with the specified format string. Otherwise the record is read and broken into columns at spaces or tabs.

The resulting array of data is then sorted into columns according to the entries. Each `<entry>` may be a simple column number, which selects the datum, an expression enclosed in parentheses, or empty. The expression can use `$1` to access the first item read, `$2` for the second item, and so on. A column number of 0 generates a number increasing (from zero) with each point, and is reset upon encountering two blank records. A column number of -1 gives the dataline number, which starts at 0, increments at single blank records, and is reset at double blank records. A column number of -2 gives the index number, which is incremented only when two blank records are found. An empty `<entry>` will default to its order in the list of entries. For example, `using ::4\verb` is interpreted as `using 1:2:4\verb`.

If the `using\verb` list has but a single entry, that `<entry>` will be used for y and the data point number is used for x; for example, `using 1\verb` is identical to `using 0:1\verb`. If the `using\verb` list has two entries, these will be used for x and y. Additional entries are usually plot stype of errors in x and/or y. See **CPlot**::**plotType**() for details about plotting styles that make use of error information.

The C Function **scanf()** accepts several numerical specifications **CPlot** requires all inputs to be double-precision floating-point variables, so `"%lf"` is essentially the only permissible specifier. A format string given by the user must contain at least one such input specifier, and no more than seven of them. scanf() expects to see white space—a blank, tab `("\t")`, newline `("\n")`, or formfeed `("\f")`—between numbers; anything else in the input stream must be explicitly skipped. Note that the use of `"\t", "\n"`, or `"\f"` requires use of double-quotes rather than single-quotes.

Examples:

This creates a plot of the sum of the 2nd and 3rd data against the first: The format string specifies comma- rather than space-separated columns.

```
using 1:($2+$3) '%lf,%lf,%lf'
```

In this example the data are read from a using a more complicated format:

```
using "%*lf%lf%*20[^\n]%lf"
```

The meaning of this format is:

```
%*lf        ignore a number
%lf         read a double-precision number (x by default)
%*20[^\n]   ignore 20 non-newline characters
%lf         read a double-precision number (y by default)
```

One trick is to use the C ternary `'?:'\verb` operator to filter data:

```
using 1:($3>10 ? $2 : 1/0)
```

which plots the datum in column two against that in column one provided the datum in column three exceeds ten. `1/0` is undefined; **CPlot** quietly ignores undefined points, so unsuitable points are suppressed.

If timeseries data are being used, the time can span multiple columns. The starting column should be specified. Note that the spaces within the time must be included when calculating starting columns for other data. E.g., if the first element on a line is a time with an embedded space, the y value should be specified as column three.

It should be noted that for three cases a) without option, b) with option of `using 1:2`, c) with option `using ($1):($2)` can be subtly different: 1) if the datafile has some lines with one column and some with two, the first will invent x values when they are missing, the second will quietly ignore the lines with one column, and the third will store an undefined value for lines with one point (so that in a plot with lines, no line joins points across the bad point); 2) if a line contains text at the first column, the first will abort the plot on an error, but the second and third should quietly skip the garbage.

In fact, it is often possible to plot a file with lots of lines of garbage at the top simply by specifying

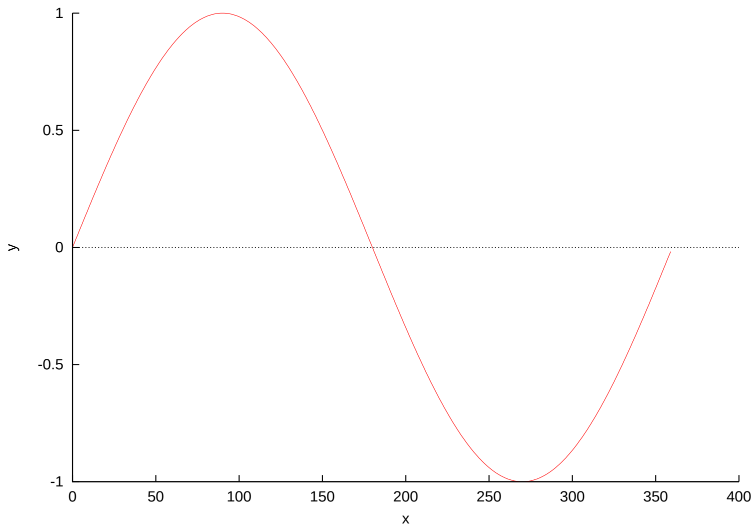
```
using 1:2
```

However, if you want to leave text in your data files, it is safer to put the comment character '#' in the first column of the text lines.

Example

```
#include <stdio.h>
#include <chplot.h>
#include <math.h>

int main() {
    char *filename;
    int i;
    class CPlot plot;
    FILE *out;

    filename = tmpnam(NULL);               //Create temporary file.
    out=fopen (filename ,"w");              //Write data to file.
    for(i=0;i<=359;i++) fprintf(out,"%i %f \n",i,sin(i*M_PI/180));
    fclose(out);
    plot.dataFile(filename);
    plot.plotting();
    remove(filename);
    return 0;
}
```

Output



Examples

See an example on page 179 for **CPlot**:**plotType**(). For comparison with data from **CPlot**::**dataFile**() and **CPlot**::**data**(), see programs on pages 100 and 181 for plot with candlesticks.

See Also

CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**, **CPlot**::**data3DCurve**, **CPlot**::**data3DSurface**, **CPlot**::**outputType**(), **CPlot**::**plotting**(), **plotxyf**(), **plotxyzf**().

CPlot::dataSetNum

Synopsis
#include <chplot.h>
int dataSetNum();

Purpose
Obtain the current data set number in an instance of **CPlot** class.

Return Value
The current data set number in an instance of **CPlot** class. The first data set number is 0. If there is no data in the instance of the CPlot class, the return value is -1.

Parameters
None.

Description
This function returns the current data set number in an instance of **CPlot** class.

Example

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i, num;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i< NUM; i++) {
      x[i] = i*10;
      y[i] = sin(x[i]*M_PI/180);
    }
    num = plot.dataSetNum();
    printf("The number of data set is %d\n", num);
    plot.data2DCurve(x, y, NUM);
    num = plot.dataSetNum();
    printf("The number of data set is %d\n", num);
    plot.data2DCurve(x, y, NUM);
    num = plot.dataSetNum();
    printf("The number of data set is %d\n", num);
    plot.plotting();
}
```

Output in console

```
The number of data set is -1
The number of data set is 0
The number of data set is 1
```

CPlot::deleteData

Synopsis
#include <**chplot.h**>
void deleteData();

Purpose
Delete all plot data of an instance of the **CPlot** class.

Return Value
None.

Parameters
None.

Description
This function frees all memory associated with previously allocated plot data, plot type, legends, plot axes, points, lines, polygons, rectangles, and circles. Unlike **CPlot**::**deletePlots**(), this function does not reset plotting options to their default values. This function allows for the reuse of a single instance of the **CPlot** class to create multiple plots.
See Also
CPlot::**arrow**(), **CPlot**::**circle**(), **CPlot**::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**dataFile**(), **CPlot**::**deletePlots**(), **CPlot**::**ticsLabel**(), **CPlot**::**line**(), **CPlot**::**point**(), **CPlot**::**polygon**, **CPlot**::**rectangle**(), **CPlot**::**text**().

CPlot::deletePlots

Synopsis
#include <**chplot.h**>
void deletePlots();

Purpose
Delete all plot data and reinitialize an instance of the class to default values.

Return Value
None.

Parameters
None.

Description
This function frees all memory associated with previously allocated plot data, plot type, legends, plot axes, text strings, arrows, points, lines, polygons, rectangles, circles, and labeled tic-marks. This function also resets all plotting options to their default values. This function allows for the reuse of a single instance of the **CPlot** class to create multiple plots. This function is used internally by **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyf**(), **plotxyzf**().

See Also
CPlot::**arrow**(), **CPlot**::**circle**(), **CPlot**::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(),

CPlot::data3DCurve(), **CPlot::data3DSurface**(), **CPlot::dataFile**(), **CPlot::deleteData**(), **CPlot::ticsLabel**(), **CPlot::line**(), **CPlot::point**(), **CPlot::polygon**, **CPlot::rectangle**(), **CPlot::text**(), **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyf**(), **plotxyzf**().

CPlot::dimension

Synopsis
#include <chplot.h>
void dimension(int *dim*);

Purpose
Set plot dimension to 2D or 3D.

Return Value
None.

Parameter
dim 2 for 2D and 3 for 3D. Default is 2.

Description
Set the dimension of the plot. The plot dimension should be set before data are added to the plot if member functions **CPlot**::dataThreeD(), **CPlot**::dataThreeDCurve(), or **CPlot**::dataThreeDSurface() are not called before. This member function must be used when 3D plotting data are added by **CPlot**::dataFile() and **CPlot::polygon**().

Example
See **CPlot::polygon**().

See Also
CPlot::data2D(), **CPlot::data2DCurve**(), **CPlot::data3D**(), **CPlot::data3DCurve**(),
CPlot::data3DSurface(), **CPlot::dataFile**(), **CPlot::polygon**().

CPlot::displayTime

Synopsis
#include <chplot.h>
void displayTime(double *x_offset*, **double** *y_offset*);

Purpose
Display the current time and date.

Return Value
None.

Parameters
x_offset Offset of the time-stamp in the x direction from its default location.

y_offset Offset of the time-stamp in the y direction from its default location.

Description

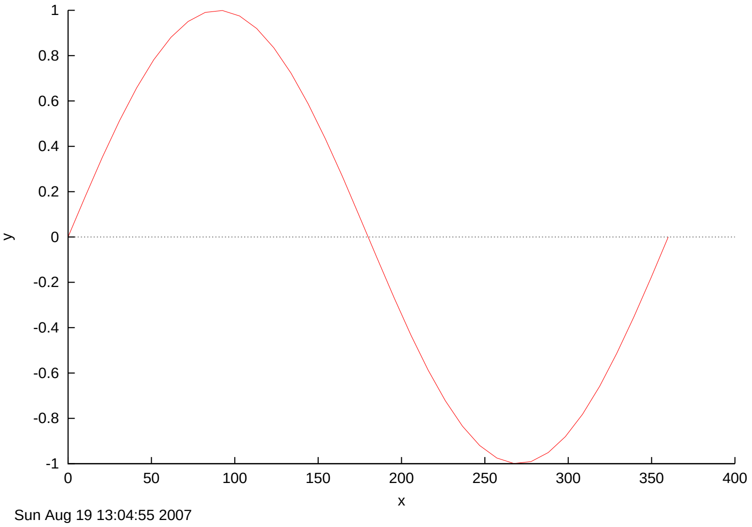
This function places the current time and date near the left margin. The exact location is device dependent. The offset values, *x_offset* and *y_offset*, are in screen coordinates and are measured in characters. For example, if both *x_offset* and *y_offset* are 2, the time will be moved approximately two characters to the right and two characters above the default location.

Example

Compare with the output for examples in **CPlot::data2D**() and **CPlot::data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.displayTime(10,0);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



CPlot::enhanceText

Synopsis

#include <chplot.h>
void enhanceText();

Purpose

Use special symbols in text.

Return Value

None.

Parameters

None.

Description

This function turns on the enhanced text mode for terminal and output files in PostScript, PNG, JPEG, GIF formats that support additional text formatting information embedded in the text string. With this function call, special symbols can be used in text such as those passed arguments of member functions **CPlot**::**label**(), **CPlot**::**legend**(), **CPlot**::**text**(), **CPlot**::**title**().

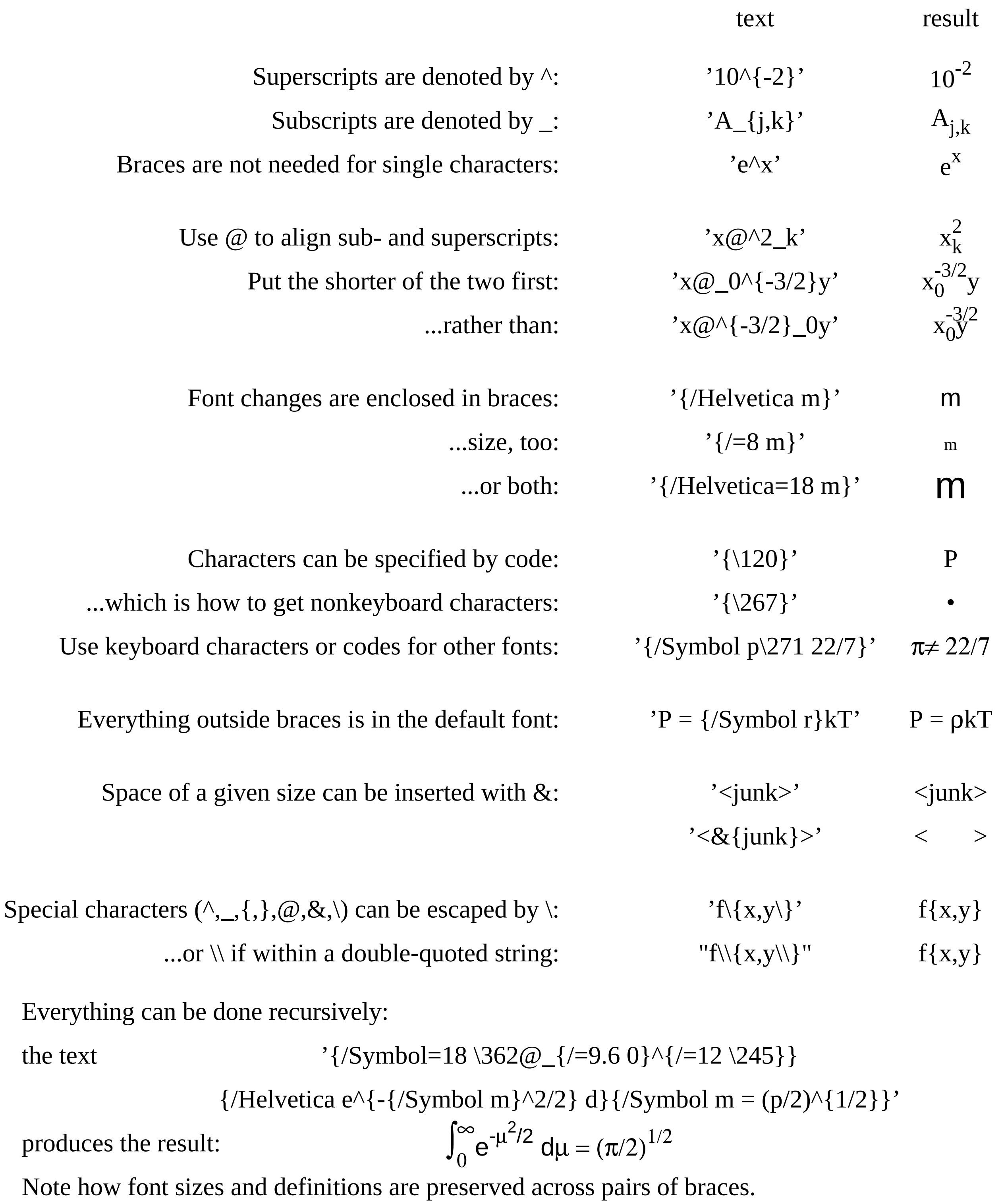
The syntax for the enhanced text.text is shown in Figure 4.1. The character codes for the enhanced text is shown in Figure 4.2. Braces can be used to place multiple-character text where a single character is expected, for example, `3ˆ{2x}`. To change the font and/or size, use the full form

```
{/[fontname][=fontsize | *fontscale] text}
```

For example, `{/Symbol=20 p}` is a 20-point π and `{/*0.5 B} is a` B at an half of whatever fontsize is currently in effect. The `'/'` character *must* be the first character after the `'{'` character to use the enhanced text.

You can access specia l symbols numerically by specifying `\character-code` in octal number, for example, `{/Symbol \160}` is the symbol for π.

You can escape control characters using `\`.



	text	result
Superscripts are denoted by ^:	'10^{-2}'	10^{-2}
Subscripts are denoted by _:	'A_{j,k}'	$A_{j,k}$
Braces are not needed for single characters:	'e^x'	e^x
Use @ to align sub- and superscripts:	'x@^2_k'	x_k^2
Put the shorter of the two first:	'x@_0^{-3/2}y'	$x_0^{-3/2}y$
...rather than:	'x@^{-3/2}_0y'	$x_0y^{-3/2}$
Font changes are enclosed in braces:	'{/Helvetica m}'	m
...size, too:	'{/=8 m}'	m
...or both:	'{/Helvetica=18 m}'	m
Characters can be specified by code:	'{\120}'	P
...which is how to get nonkeyboard characters:	'{\267}'	•
Use keyboard characters or codes for other fonts:	'{/Symbol p\271 22/7}'	π≠ 22/7
Everything outside braces is in the default font:	'P = {/Symbol r}kT'	P = ρkT
Space of a given size can be inserted with &:	'<junk>'	<junk>
	'<&{junk}>'	< >
Special characters (^,_,{,},@,&,\) can be escaped by \:	'f\{x,y\}'	f{x,y}
...or \\ if within a double-quoted string:	"f\\{x,y\\}"	f{x,y}

Everything can be done recursively:

the text '{/Symbol=18 \362@_{/=9.6 0}^{/=12 \245}}
{/Helvetica e^{-{/Symbol m}^2/2} d}{/Symbol m = (p/2)^{1/2}}'

produces the result: $\int_0^\infty e^{-\mu^2/2}\, d\mu = (\pi/2)^{1/2}$

Note how font sizes and definitions are preserved across pairs of braces.

Figure 4.1: Syntax for the enhanced text.

T = text (here Times-Roman) S = Symbol Z = ZapfDingbats E = ISO Latin-1 encoding
(the "E" character set is accessed via a member function CPlot::nativeCmd("set encoding"))

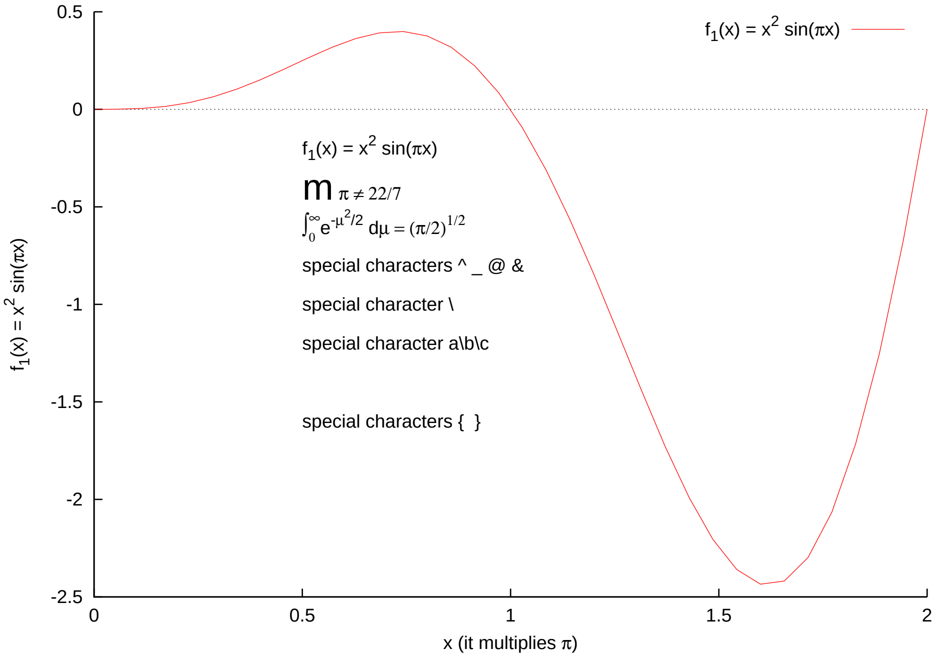
	T	S	Z	E		T	S	Z	E		T	S	Z	E		T	S	Z	E		T	S	Z	E
040					111	I	Ι	✩	I	162	r	ρ	❒	r	256	ﬁ	→	③	®	327	×	⋅	↕	×
041	!	!	✁	!	112	J	ϑ	✪	J	163	s	σ	▲	s	257	ﬂ	↓	④	¯	330	Ø	¬	➘	Ø
042	"	∀	✂	"	113	K	Κ	✫	K	164	t	τ	▼	t	260	°	°	⑤	°	331	Ù	∧	➙	Ù
043	#	#	✃	#	114	L	Λ	✬	L	165	u	υ	◆	u	261	–	±	⑥	±	332	Ú	∨	➚	Ú
044	$	∃	✄	$	115	M	Μ	✭	M	166	v	ϖ	❖	v	262	†	″	⑦	²	333	Û	⇔	➛	Û
045	%	%	☎	%	116	N	Ν	✮	N	167	w	ω	◗	w	263	‡	≥	⑧	³	334	Ü	⇐	➜	Ü
046	&	&	✆	&	117	O	Ο	✯	O	170	x	ξ	❘	x	264	·	×	⑨	´	335	Ý	⇑	➝	Ý
047	’	∋	✇	’	120	P	Π	✰	P	171	y	ψ	❙	y	265	µ	∝	⑩	µ	336	Þ	⇒	➞	Þ
050	(	(	✈	(	121	Q	Θ	✱	Q	172	z	ζ	❚	z	266	¶	∂	❶	¶	337	ß	⇓	➟	ß
051	)	)	✉	)	122	R	Ρ	✲	R	173	{	{	❛	{	267	•	•	❷	·	340	à	◊	➠	à
052	*	∗	☛	*	123	S	Σ	✳	S	174	|	|	❜	|	270	‚	÷	❸	¸	341	Æ	〈	➡	á
053	+	+	☞	+	124	T	Τ	✴	T	175	}	}	❝	}	271	„	≠	❹	¹	342	â	®	➢	â
054	,	,	✌	,	125	U	Υ	✵	U	176	~	∼	❞	~	272	”	≡	❺	º	343	ª	©	➣	ã
055	-	−	✍	−	126	V	ς	✶	V	220				ı	273	»	≈	❻	»	344	ä	™	➤	ä
056	.	.	✎	.	127	W	Ω	✷	W	221				`	274	…	…	❼	¼	345	å	∑	➥	å
057	/	/	✏	/	130	X	Ξ	✸	X	222				´	275	‰	⏐	❽	½	346	æ	⎛	➦	æ
060	0	0	✐	0	131	Y	Ψ	✹	Y	223				ˆ	276	¾	⎯	❾	¾	347	ç	⎜	➧	ç
061	1	1	✑	1	132	Z	Ζ	✺	Z	224				˜	277	¿	↵	❿	¿	350	Ł	⎝	➨	è
062	2	2	✒	2	133	[	[	✻	[	225				¯	300	À	ℵ	➀	À	351	Ø	⎡	➩	é
063	3	3	✓	3	134	\	∴	✼	\	226				˘	301	`	ℑ	➁	Á	352	Œ	⎢	➪	ê
064	4	4	✔	4	135	]	]	✽	]	227				˙	302	´	ℜ	➂	Â	353	º	⎣	➫	ë
065	5	5	✕	5	136	^	⊥	✾	^	230				¨	303	ˆ	℘	➃	Ã	354	ì	⎧	➬	ì
066	6	6	✖	6	137	_	_	✿	_	232				˚	304	˜	⊗	➄	Ä	355	í	⎨	➭	í
067	7	7	✗	7	140	‘		❀	‘	233				¸	305	¯	⊕	➅	Å	356	î	⎩	➮	î
070	8	8	✘	8	141	a	α	❁	a	235				˝	306	˘	∅	➆	Æ	357	ï	⎪	➯	ï
071	9	9	✙	9	142	b	β	❂	b	236				˛	307	˙	∩	➇	Ç	360	ð			ð
072	:	:	✚	:	143	c	χ	❃	c	237				ˇ	310	¨	∪	➈	È	361	æ	〉	➱	ñ
073	;	;	✛	;	144	d	δ	❄	d	240		€			311	É	⊃	➉	É	362	ò	∫	➲	ò
074	<	<	✜	<	145	e	ε	❅	e	241	¡	ϒ	❡	¡	312	˚	⊇	➊	Ê	363	ó	⌠	➳	ó
075	=	=	✝	=	146	f	φ	❆	f	242	¢	′	❢	¢	313	¸	⊄	➋	Ë	364	ô	⎮	➴	ô
076	>	>	✞	>	147	g	γ	❇	g	243	£	≤	❣	£	314	Ì	⊂	➌	Ì	365	ı	⌡	➵	õ
077	?	?	✟	?	150	h	η	❈	h	244	⁄	⁄	❤	¤	315	˝	⊆	➍	Í	366	ö	⎞	➶	ö
100	@	≅	✠	@	151	i	ι	❉	i	245	¥	∞	❥	¥	316	˛	∈	➎	Î	367	÷	⎟	➷	÷
101	A	Α	✡	A	152	j	ϕ	❊	j	246	ƒ	ƒ	❦	¦	317	ˇ	∉	➏	Ï	370	ł	⎠	➸	ø
102	B	Β	✢	B	153	k	κ	❋	k	247	§	♣	❧	§	320	—	∠	➐	Ð	371	ø	⎤	➹	ù
103	C	Χ	✣	C	154	l	λ	●	l	250	¤	♦	♣	¨	321	Ñ	∇	➑	Ñ	372	œ	⎥	➺	ú
104	D	Δ	✤	D	155	m	μ	❍	m	251	'	♥	♦	©	322	Ò	®	➒	Ò	373	ß	⎦	➻	û
105	E	Ε	✥	E	156	n	ν	■	n	252	“	♠	♥	ª	323	Ó	©	➓	Ó	374	ü	⎫	➼	ü
106	F	Φ	✦	F	157	o	ο	❏	o	253	«	↔	♠	«	324	Ô	™	➔	Ô	375	ý	⎬	➽	ý
107	G	Γ	✧	G	160	p	π	❐	p	254	‹	←	①	¬	325	Õ	∏	→	Õ	376	þ	⎭	➾	þ
110	H	Η	★	H	161	q	θ	❑	q	255	›	↑	②	-	326	Ö	√	↔	Ö	377	ÿ			ÿ

Figure 4.2: character codes for the enhanced text.

Example 1

```
/* File: enhanceText.cpp */
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;
    char funcname[] = "f_1(x) = x^2 sin({/Symbol p}x)";

    for(i=0; i< NUM; i++) {
      x[i] = i*(2.0)/(NUM-1);
      y[i] = x[i]*x[i]*sin(x[i]*M_PI);
    }
    plot.data2DCurve(x, y, NUM);
    plot.enhanceText();
    plot.label(PLOT_AXIS_X, "x (it multiplies {/Symbol p})");
    plot.label(PLOT_AXIS_Y, funcname);
    plot.legend(funcname, 0);
    plot.text(funcname, PLOT_TEXT_LEFT, 0.5, -0.2, 0);
    plot.text("{/Helvetica=28 m} {/Symbol p \271 22/7}", PLOT_TEXT_LEFT, 0.5, -0.4, 0);
    plot.text("{/Symbol=18 \362@_{/=9.6 0}^{/=12 \245}}"
              "{/Helvetica e^{-{/Symbol m}^2/2} d}{/Symbol m = (p/2)^{1/2}}",
               PLOT_TEXT_LEFT, 0.5, -0.6, 0);
    plot.text("special characters \\\\^ \\\\_ \\\\@ \\\\&", PLOT_TEXT_LEFT, 0.5, -0.8, 0);
    plot.text("special character \\134 ", PLOT_TEXT_LEFT, 0.5, -1.0, 0);
    plot.text("special character a\\134b\\\\c", PLOT_TEXT_LEFT, 0.5, -1.2, 0);

    /* for display on the screen */
    plot.text("special characters \\\\{ \\\\} ", PLOT_TEXT_LEFT, 0.5, -1.4, 0);
    /* For postscript file use the format below to create ’{’ and ’}’ */
    //plot.text("special characters \\\\173  \\\\175", PLOT_TEXT_LEFT, 0.5, -1.6, 0);
    //plot.outputType(PLOT_OUTPUTTYPE_FILE, "postscript eps color",
                      "../output/outputOption.eps");
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::**label**(), **CPlot**::**legend**(), **CPlot**::**text**(), **CPlot**::).

CPlot::func2D

Synopsis
#include <**chplot.h**>
int func2D(**double** *x0*, **double** *xf*, **int** *n*, **double** (**func*)(**double** *x*));

Purpose
Add a set of data using a function for 2D curve to an instance of **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x0 the initial value for the range of the function.

xf the final value for the range of the function.

n the number of points for the range of the function.

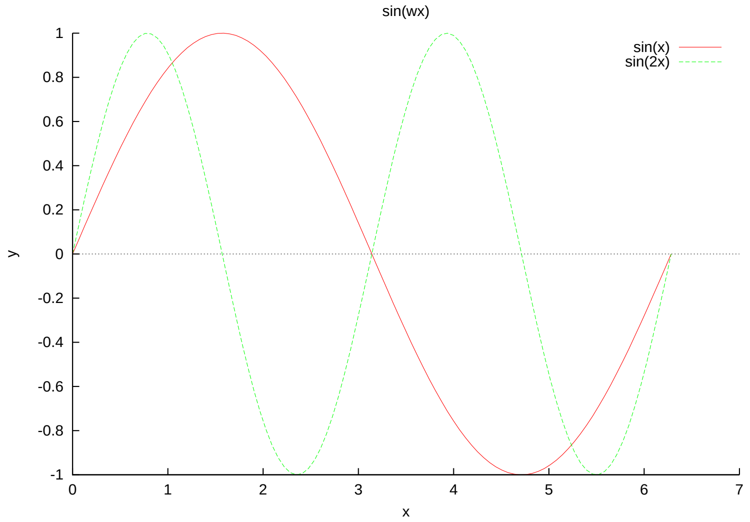
func a pointer to function for adding a set of data.

Description
This function adds a set of data using a function `func()` in the range from `x0` to `xf` with n points to a previously declared instance of the **CPlot** class.

Example 1

```
#include<math.h>
#include<chplot.h>

#define N 100

double omega;
double func(double x) {
   double y;

   y = sin(omega*x);
   return y;
}

int main() {
   double x0, xf;
   CPlot plot;

   x0 = 0;
   xf = 2*M_PI;
   plot.title("sin(wx)");
   plot.func2D(x0, xf, N, sin);
   plot.legend("sin(x)", 0);
   omega = 2;
   plot.func2D(x0, xf, N, func);
   plot.legend("sin(2x)", 1);
   plot.plotting();
}
```

Output



See Also

CPlot::func3D(), **CPlot::funcp2D**(), **CPlot::funcp3D**(), **CPlot::data2D**(), **CPlot::data3D**(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **fplotxy**().

CPlot::func3D

Synopsis

#include <chplot.h>
int func3D(double *x0*, **double** *xf*, **double** *y0*, **double** *yf*, **int** *nx*, **int** *ny*, **double** (**func*)(**double** *x*, **double** *y*));

Purpose

Add a set of data using a function for 3D surface to an instance of **CPlot** class.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

x0 the initial value for the x range of the function.

xf the final value for the x range of the function.

y0 the initial value for the y range of the function.

yf the final value for the y range of the function.

nx the number of points for the x range of the function.

ny the number of points for the y range of the function.

func a pointer to function for adding a set of data.

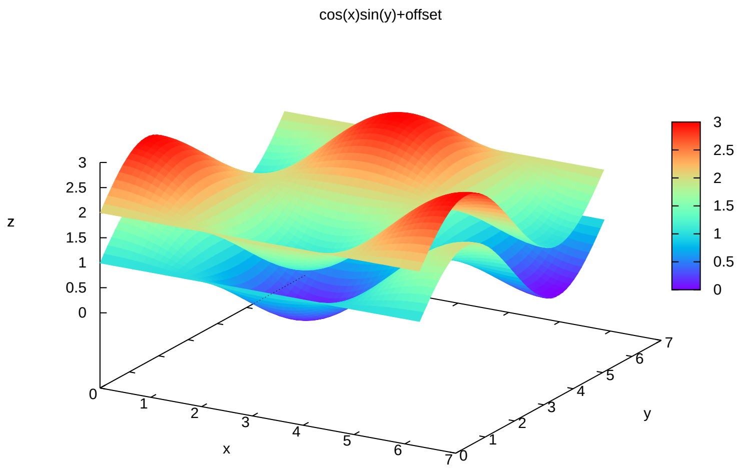
Description

This function adds a set of data using a function `func()` with nx points in the range from `x0` to `xf` for `x` and with ny points in the range from `y0` to `yf` for `y` to a previously declared instance of the **CPlot** class.

Example 1

```
#include<math.h>
#include<chplot.h>

#define NX 50
#define NY 50

double offset;

double func(double x, double y) {
   double z;

   z = cos(x)*sin(y) +offset;
   return z;
}

int main() {
   double x0, xf, y0, yf;
   CPlot plot;

   x0 = 0;
   xf = 2*M_PI;
   y0 = 0;
   yf = 2*M_PI;
   offset = 1;
   plot.title("cos(x)sin(y)+offset");
   plot.func3D(x0, xf, y0, yf, NX, NY, func);
   offset = 2;
   plot.func3D(x0, xf, y0, yf, NX, NY, func);
   plot.plotting();
}
```

Output



See Also

CPlot::func2D(), **CPlot::func3D**(), **CPlot::funcp2D**(), **CPlot::data2D**(), **CPlot::data3D**(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **fplotxy**().

CPlot::funcp2D

Synopsis

#include <**chplot.h**>
int funcp2D(**double** *x0*, **double** *xf*, **int** *n*, **double** (**func*)(**double** *x*, **void** * *param*, **void** * *param*);

Purpose

Add a set of data using a function for 2D curve to an instance of **CPlot** class.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

x0 the initial value for the range of the function.

xf the final value for the range of the function.

n the number of points for the range of the function.

func a pointer to function for adding a set of data.

param a parameter passed to the function. If this argument is not used inside function `func()`, NULL can be passed to this paramenter.

Description

This function adds a set of data using a function with a parameter `func()` in the range from `x0` to `xf` with n points to a previously declared instance of the **CPlot** class.

Example 1

```
#include<math.h>
#include<chplot.h>

#define N 100

double func(double x, void *param) {
   double omega, y;

   omega = *(double*)param;
   y = sin(omega*x);
   return y;
}

int main() {
   double x0, xf, omega;
   CPlot plot;

   x0 = 0;
   xf = 2*M_PI;
   omega = 1;
   plot.title("sin(wx)");
   plot.funcp2D(x0, xf, N, func, &omega);
   plot.legend("sin(x)", 0);
   omega = 2;
   plot.funcp2D(x0, xf, N, func, &omega);
   plot.legend("sin(2x)", 1);
   plot.plotting();
}
```

Output

See **CPlot::func2D**().

See Also

CPlot::func2D(), **CPlot::funcp3D**(), **CPlot::funcp3D**(), **CPlot::data2D**(), **CPlot::data3D**(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **fplotxy**().

CPlot::funcp3D

Synopsis

#include <chplot.h>
int funcp3D(double *x0*, **double** *xf*, **double** *y0*, **double** *yf*, **int** *nx*, **int** *ny*, **double** (**func*)(**double** *x*, **double** *y*, **void** * *param*, **void** * *param*);

Purpose

Add a set of data using a function for 3D surface to an instance of **CPlot** class.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

x0 the initial value for the x range of the function.

xf the final value for the x range of the function.

y0 the initial value for the y range of the function.

yf the final value for the y range of the function.

nx the number of points for the x range of the function.

ny the number of points for the y range of the function.

func a pointer to function for adding a set of data.

param a parameter passed to the function. If this argument is not used inside function `func()`, NULL can be passed to this paramenter.

Description

This function adds a set of data using a function `func()` with nx points in the range from `x0` to `xf` for `x` and with ny points in the range from `y0` to `yf` for `y` to a previously declared instance of the **CPlot** class. The function has an optional paramter.

Example 1

```
#include<math.h>
#include<chplot.h>

#define NX 50
#define NY 50

double func(double x, double y, void *param) {
   double offset, z;

   offset = *(double*)param;
   z = cos(x)*sin(y) +offset;
   return z;
}

int main() {
   double x0, xf, y0, yf, offset;
   CPlot plot;

   x0 = 0;
   xf = 2*M_PI;
   y0 = 0;
   yf = 2*M_PI;
   offset = 1;
   plot.title("cos(x)sin(y)+offset");
   plot.funcp3D(x0, xf, y0, yf, NX, NY, func, &offset);
   offset = 2;
   plot.funcp3D(x0, xf, y0, yf, NX, NY, func, &offset);
   plot.plotting();
}
```

Output
See **CPlot**::**func3D**().
See Also
CPlot::**func2D**(), **CPlot**::**func3D**(), **CPlot**::**funcp2D**(), **CPlot**::**data2D**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **fplotxy**().

CPlot::getLabel

Synopsis in Ch
#include <**chplot.h**>
string_t getLabel(**int** `axis`);

Synopsis in C++
#include <**chplot.h**>
char * **getLabel**(**int** `axis`);

Purpose
Get the label for a plot axis.

Return Value
The label of the axis.

Parameters
axis The axis with its label to be obtained. Valid values are:

- **PLOT_AXIS_X** Select the x axis only.
- **PLOT_AXIS_X2** Select the x2 axis only.
- **PLOT_AXIS_Y** Select the y axis only.
- **PLOT_AXIS_Y2** Select the y2 axis only.
- **PLOT_AXIS_Z** Select the z axis only.

Description
Get the label of a plot axis.

Example

```
#include <math.h>
#include <chplot.h>

int main() {
    class CPlot plot;
    char *xlabel, *ylabel, *zlabel, *title;

    plot.label(PLOT_AXIS_X, "xlabel");
    xlabel = plot.getLabel(PLOT_AXIS_X);
    ylabel = plot.getLabel(PLOT_AXIS_Y);
    zlabel = plot.getLabel(PLOT_AXIS_Z);
    printf("xlabel = %s\n", xlabel);
    printf("ylabel = %s\n", ylabel);
    printf("zlabel = %s\n", zlabel);
    title = plot.getTitle();
    printf("title = %p\n", title);
    plot.title("New Title");
    title = plot.getTitle();
    printf("title = %s\n", title);
}
```

Output

```
xlabel = xlabel
ylabel = y
zlabel = z
title = 0
title = New Title
```

See Also
CPlot::**label**(), **CPlot**::**title**(), **CPlot**::**getTitle**().

CPlot::getOutputType

Synopsis in Ch
#include <**chplot.h**>
plotinfo_t getOutputType();

Synopsis in C++
#include <**chplot.h**>
plotinfo_t getOutputType();

Purpose
Get the output type for a plot.

Return Value
The output type in one of the following forms:

PLOT_OUTPUTTYPE_DISPLAY Display the plot on the screen.

PLOT_OUTPUTTYPE_STREAM Output the plot as a standard output stream.

PLOT_OUTPUTTYPE_FILE Output the plot to a file in one of a variety of formats.

Parameters None
Description
Get the output type of a plot.

Example

```
#include <math.h>
#include <chplot.h>
```

```
int main() {
    class CPlot plot;
    plotinfo_t outputtype;

    outputtype = plot.getOutputType();
    if(outputtype == PLOT_OUTPUTTYPE_DISPLAY)
      printf("output is displayed\n");
    else if(outputtype == PLOT_OUTPUTTYPE_STREAM)
      printf("output is stdout stream\n");
    else if(outputtype == PLOT_OUTPUTTYPE_FILE)
      printf("output is in a file\n");
    return 0;
}
```

Output

```
output is displayed
```

See Also
CPlot::**outputType**().

CPlot::getSubplot

Synopsis
#include <chplot.h>
class CPlot* getSubplot(int *row*, **int** *col*);

Purpose
Get a pointer to an element of a subplot.

Return Value
Returns a pointer to the specified element of the subplot.

Parameters

row The row number of the desired subplot element. Numbering starts with zero.

col The column number of the desired subplot element. Numbering starts with zero.

Description
After the creation of a subplot using **CPlot**::**subplot**(), this function can be used to get a pointer to an element of the subplot. This pointer can be used as a **CPlot** pointer normally would be. Addition of data sets or selection of plotting options are done normally. However, each option only effects the subplot element currently pointed to.

Example

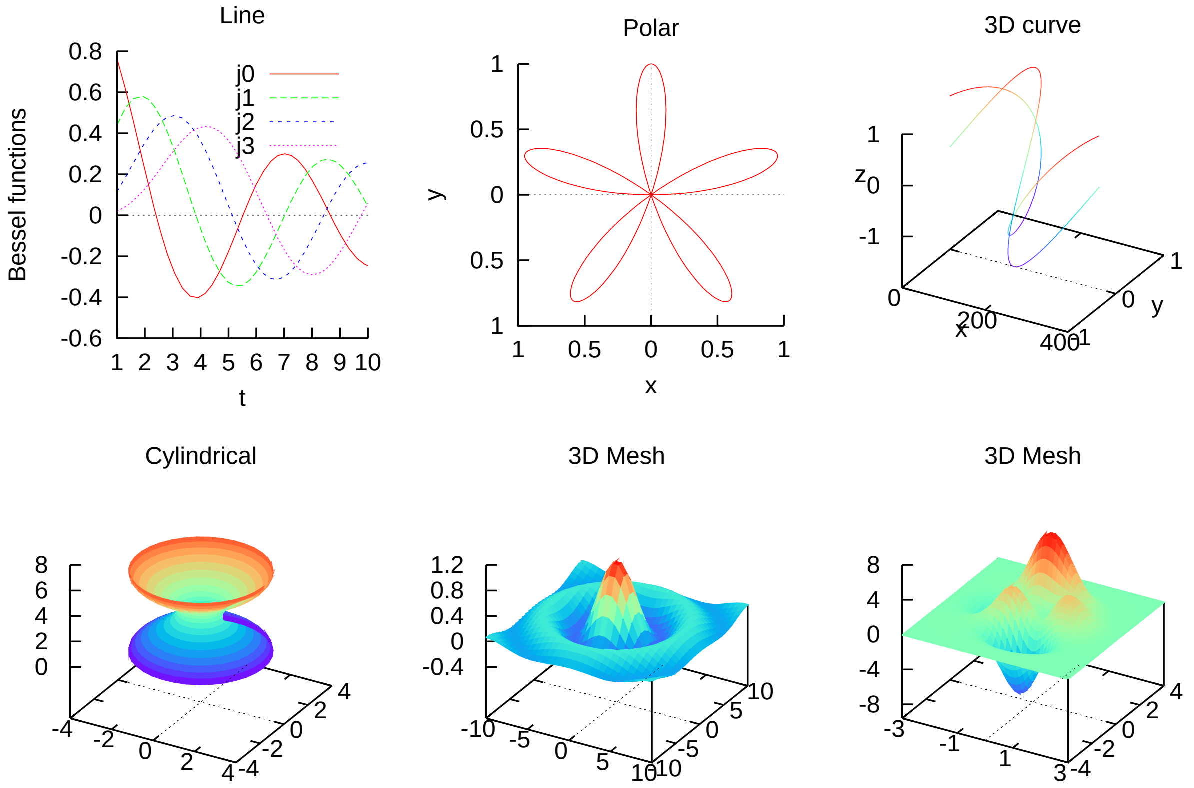
```
#include <chplot.h>
#include <math.h>

#define POINTS 50
```

```
#define NUM1 30
#define NUM2 360
#define NUMX1 36
#define NUMY1 20
#define NUMX2 30
#define NUMY2 30
#define NUMX3 40
#define NUMY3 60
int main() {
    double t[POINTS], b0[POINTS], b1[POINTS], b2[POINTS], b3[POINTS];
    int i, j, datasetnum=0, line_type=4, line_width = 2;
    double theta2[NUM2], r2[NUM2];
    double x3[NUM2], y3[NUM2], z3[NUM2], z32[NUM2];
    double theta4[NUMX1], z4[NUMY1], r4[NUMX1*NUMY1];
    double r, x5[NUMX2], y5[NUMY2], z5[NUMX2*NUMY2];
    double x6[NUMX3], y6[NUMY3], z6[NUMX3*NUMY3];
    class CPlot subplot, *spl;

    /* plot 1 */
    for(i=0; i< POINTS; i++) {
      t[i] = 1+i*(10.0-1)/(POINTS-1);
      b0[i] = j0(t[i]);
      b1[i] = j1(t[i]);
      b2[i] = jn(2, t[i]);
      b3[i] = jn(3, t[i]);
    }

    /* plot 2 */
    for(i=0; i<NUM2; i++) {
       theta2[i]= 0 + i*M_PI/(NUM2-1); // linspace(theta2, 0, M_PI);
       r2[i] = sin(5*theta2[i]); //  r2 = sin(5*theta2);
    }

    /* plot 3 */
    for(i=0; i<NUM2; i++) {
       x3[i]= 0 + i*360.0/(NUM2-1); // linspace(x3, 0, 360);
       y3[i] = sin(x3[i]*M_PI/180);
       z3[i] = cos(x3[i]*M_PI/180);
       z32[i] = y3[i];
    }

    /* plot 4 */
    for(i=0; i<NUMX1; i++) {
       theta4[i]= 0 + i*360.0/(NUMX1-1); // linspace(theta4, 0, 360);
    }
    for(i=0; i<NUMY1; i++) {
       z4[i]= 0 + i*2*M_PI/(NUMY1-1); // linspace(z4, 0, 2*M_PI);
    }
    for(i=0; i<NUMX1; i++) {
        for(j=0; j<NUMY1; j++) {
            r4[i*NUMY1+j] = 2+cos(z4[j]);
        }
    }

    /* plots 5 */
    for(i=0; i<NUMX2; i++) {
       x5[i]= -10 + i*20.0/(NUMX2-1); // linspace(x5, -10, 10);
    }
    for(i=0; i<NUMY2; i++) {
```

```
   y5[i]= -10 + i*20/(NUMY2-1); // linspace(y5, -10, 10);
}
for(i=0; i<NUMX2; i++) {
    for(j=0; j<NUMY2; j++) {
        r = sqrt(x5[i]*x5[i]+y5[j]*y5[j]);
        z5[NUMY2*i+j] = sin(r)/r;
    }
}

/* plots 6 */
for(i=0; i<NUMX3; i++) {
   x6[i]= -3 + i*6.0/(NUMX3-1); // linspace(x6, -3, 3);
}
for(i=0; i<NUMY3; i++) {
   y6[i]= -4 + i*8/(NUMY3-1); // linspace(y6, -4, 4);
}
for(i=0; i<NUMX3; i++) {
    for(j=0; j<NUMY3; j++) {
        z6[NUMY3*i+j] = 3*(1-x6[i])*(1-x6[i])*
                     exp(-(x6[i]*x6[i])-(y6[j]+1)*(y6[j]+1))
        - 10*(x6[i]/5 - x6[i]*x6[i]*x6[i]-pow(y6[j],5))*
                     exp(-x6[i]*x6[i]-y6[j]*y6[j])
        - 1/3*exp(-(x6[i]+1)*(x6[i]+1)-y6[j]*y6[j]);
    }
}



subplot.subplot(2,3); /* create 2 x 3 subplot */
spl = subplot.getSubplot(0,0); /* get subplot (0,0) */
spl->title("Line");
spl->label(PLOT_AXIS_X,"t");
spl->label(PLOT_AXIS_Y,"Bessel functions");
spl->data2DCurve(t, b0, POINTS);
spl->data2DCurve(t, b1, POINTS);
spl->data2DCurve(t, b2, POINTS);
spl->data2DCurve(t, b3, POINTS);
spl->legend("j0", 0);
spl->legend("j1", 1);
spl->legend("j2", 2);
spl->legend("j3", 3);

spl = subplot.getSubplot(0,1); /* get subplot (0,1) */
spl->title("Polar");
spl->axisRange(PLOT_AXIS_XY, -1, 1);
spl->ticsRange(PLOT_AXIS_XY, .5, -1, 1);
spl->data2DCurve(theta2, r2, NUM2);
spl->polarPlot(PLOT_ANGLE_RAD);
spl->sizeRatio(-1);

spl = subplot.getSubplot(0,2);      /* get subplot (0,2) */
spl->title("3D curve");
spl->data3DCurve(x3, y3, z3, NUM2);
spl->data3DCurve(x3, y3, z32, NUM2);
spl->axisRange(PLOT_AXIS_X, 0, 400);
spl->ticsRange(PLOT_AXIS_X, 200, 0, 400);
spl->axisRange(PLOT_AXIS_Y, -1, 1);
spl->ticsRange(PLOT_AXIS_Y, 1, -1, 1);
spl->axisRange(PLOT_AXIS_Z, -1, 1);
```

```
    spl->ticsRange(PLOT_AXIS_Z, 1, -1, 1);
    spl->colorBox(PLOT_OFF);

    spl = subplot.getSubplot(1,0); /* get subplot (1,0) */
    spl->title("Cylindrical");
    spl->data3DSurface(theta4, z4, r4, NUMX1, NUMY1);
    spl->coordSystem(PLOT_COORD_CYLINDRICAL, PLOT_ANGLE_DEG);
    spl->axisRange(PLOT_AXIS_Z, 0, 8);
    spl->ticsRange(PLOT_AXIS_Z, 2, 0, 8);
    spl->axisRange(PLOT_AXIS_XY, -4, 4);
    spl->colorBox(PLOT_OFF);
    spl->ticsRange(PLOT_AXIS_XY, 2, -4, 4);
    spl->label(PLOT_AXIS_XYZ, NULL);

    spl = subplot.getSubplot(1,1); /* get subplot (1,1) */
    spl->title("3D Mesh");
    spl->axisRange(PLOT_AXIS_X, -10, 10);
    spl->ticsRange(PLOT_AXIS_X, 5, -10, 10);
    spl->axisRange(PLOT_AXIS_Y, -10, 10);
    spl->ticsRange(PLOT_AXIS_Y, 5, -10, 10);
    spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
    spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
    spl->data3DSurface(x5, y5, z5, NUMX2, NUMY2);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);

    spl = subplot.getSubplot(1,2);      /* get subplot (1,2) */
    spl->title("3D Mesh");
    spl->data3DSurface(x6, y6, z6, NUMX3, NUMY3);
    spl->axisRange(PLOT_AXIS_X, -3, 3);
    spl->ticsRange(PLOT_AXIS_X, 2, -3, 3);
    spl->axisRange(PLOT_AXIS_Y, -4, 4);
    spl->ticsRange(PLOT_AXIS_Y, 2, -4, 4);
    spl->axisRange(PLOT_AXIS_Z, -8, 8);
    spl->ticsRange(PLOT_AXIS_Z, 4, -8, 8);
    spl->colorBox(PLOT_OFF);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->ticsLevel(0.1);
    subplot.plotting();
    return 0;
}
```

Output



See Also
CPlot::**subplot**().

CPlot::getTitle

Synopsis in Ch
#include <**chplot.h**>
string_t getTitle(void);

Synopsis in C++
#include <**chplot.h**>
char *getTitle(void);

Purpose
Get the plot title.

Return Value
The plot title.

Parameters
None.

Description

Get the title of the plot. If no title, `NULL` will be returned.

Example

see **CPlot**::**getLabel**().

See Also

CPlot::**label**(), **CPlot**::**getLabel**(), **CPlot**::**title**().

CPlot::grid

Synopsis in Ch

#include <**chplot.h**>
void grid(**int** *flag*, ... /* [**char** * *option*] */);

Synopsis in C++

#include <**chplot.h**>
void grid(**int** *flag*);
void grid(**int** *flag*, **char** * *option*);

Syntax in Ch and C++

grid(**PLOT_OFF**)
grid(**PLOT_ON**)
grid(**PLOT_ON**, option)

Purpose

Enable or disable the display of a grid on the xy plane.

Return Value

None.

Parameters

flag This parameter can be set to:

- **PLOT_ON** Enable the display of the grid.
- **PLOT_OFF** Disable the display of the grid.

option The option for the grid.

Description

Enable or disable the display of a grid on the xy plane. By default, the grid is off. For a polar plot, the polar grid is displayed. Otherwise, the grid is rectangular.

The optional argument `option` of string type with the following values can be used to fine tune the grid based on the argument for set grid command of the gnuplot.

```
{{no}{m}xtics} {{no}{m}ytics} {{no}{m}ztics}
{{no}{m}x2tics} {{no}{m}y2tics}
{{no}{m}cbtics}
```

```
{polar {<angle>}}
{layerdefault | front | back}
{ {linestyle <major_linestyle>}
  | {linetype | lt <major_linetype>}
    {linewidth | lw <major_linewidth>}
  { , {linestyle | ls <minor_linestyle>}
      | {linetype | lt <minor_linetype>}
        {linewidth | lw <minor_linewidth>} } }
```

The grid can be enabled and disabled for the major and/or minor tic marks on any axis, and the linetype and linewidth can be specified for major and minor grid lines.

Additionally, a polar grid can be selected for 2-d plots—circles are drawn to intersect the selected tics, and radial lines are drawn at definable intervals. The interval is given in degrees or radians, depending on the argument of **CPlot**::**polarPlot**(). The default polar angle is 30 degrees.

The pertinent tics must be enabled. Otherwise, the plotting engine will quietly ignore instructions to draw grid lines at non-existent tics, but they will appear if the tics are subsequently enabled.

The ‘linetype‘ is followed by an integer index representing the line type for drawing. The line type varies depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line or make it dashed or dotted. All terminals support at least six different line types. The ‘linewidth‘ is followed by a scaling factor for the line width. The line width is ‘linewidth‘ multiplied by the default width. Typically the default width is one pixel.

If no linetype is specified for the minor gridlines, the same linetype as the major gridlines is used.

If `"front"` is given, the grid is drawn on top of the graphed data. If `"back"` is given, the grid is drawn underneath the graphed data. Using `"front"` will prevent the grid from being obscured by dense data. The default setup, `"layerdefault"`, is equivalent to `"back"` for 2d plots. In 3D plots the default is to split up the grid and the graph box into two layers: one behind, the other in front of the plotted data and functions.

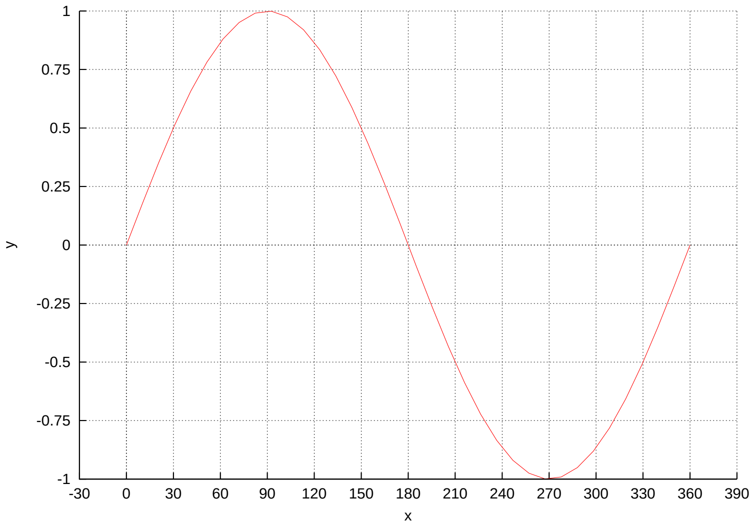
For 3D plot, Z grid lines are drawn on the bottom of the plot.

Example 1

Compare with the output for the example in **CPlot**::**axisRange**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    double x[NUM], y[NUM];
    int i;
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.axisRange(PLOT_AXIS_X, -30, 390);
    plot.ticsRange(PLOT_AXIS_X, 30, -30, 390);
    plot.axisRange(PLOT_AXIS_Y, -1, 1);
    plot.ticsRange(PLOT_AXIS_Y, .25, -1, 1);
    plot.grid(PLOT_ON);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output

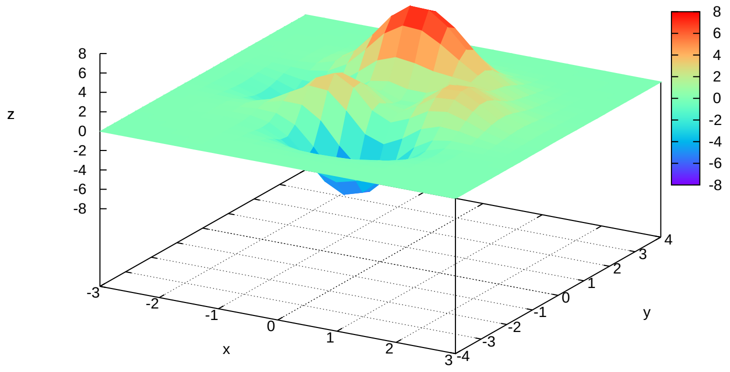


Example 2

Compare with the output for examples in **CPlot**::**data3D**() and **CPlot**::**data3DSurface**().

```
#include <math.h>
#include <chplot.h>

#define NUMX 20
#define NUMY 30
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    int i,j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -3 + i*6.0/(NUMX-1); // linspace(x, -3, 3);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -4 + i*8.0/(NUMY-1); // linspace(y, -4, 4);
    }
    for(i=0; i<NUMX; i++) {
       for(j=0; j<NUMY; j++) {
           z[NUMY*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.grid(PLOT_ON);
    plot.plotting();
    return 0;
}
```

Output



Example 3
An example with grids for both y and y2 axes on page 68 for **CPlot**:**axes**().
Example 4
An example with grids on the front for a filled curve on page 187 for **CPlot**:**plotType**().

CPlot::isUsed

Synopsis
#include <**chplot.h**>
int isUsed();

Purpose
Test if an instance of the **CPlot** class has been used.

Return Value
Returns `true` if the **CPlot** instance has been previously used, returns `false` otherwise.

Parameters
None.

Description
This function determines if an instance of the **CPlot** class has previously been used. The function actually tests if data have previously been added to the instance, by checking if an internal pointer in the CPlot class is NULL. If the pointer is not NULL, then the instance has been used. This function is used internally by **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyf**(), and **plotxyzf**().

See Also
fplotxy(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyf**(), and **plotxyzf**().

CPlot::label

Synopsis in Ch
#include <chplot.h>
void label(**int** `axis`, **string_t** *label*);

Synopsis in C++
#include <chplot.h>
void label(**int** `axis`, **char** * *label*);

Purpose
Set the labels for the plot axes.

Return Value
None.

Parameters
axis The axis to be set. Valid values are:

> **PLOT_AXIS_X** Select the x axis only.
>
> **PLOT_AXIS_X2** Select the x2 axis only.
>
> **PLOT_AXIS_Y** Select the y axis only.
>
> **PLOT_AXIS_Y2** Select the y2 axis only.
>
> **PLOT_AXIS_Z** Select the z axis only.
>
> **PLOT_AXIS_XY** Select the x and y axes.
>
> **PLOT_AXIS_XYZ** Select the x, y, and z axes.

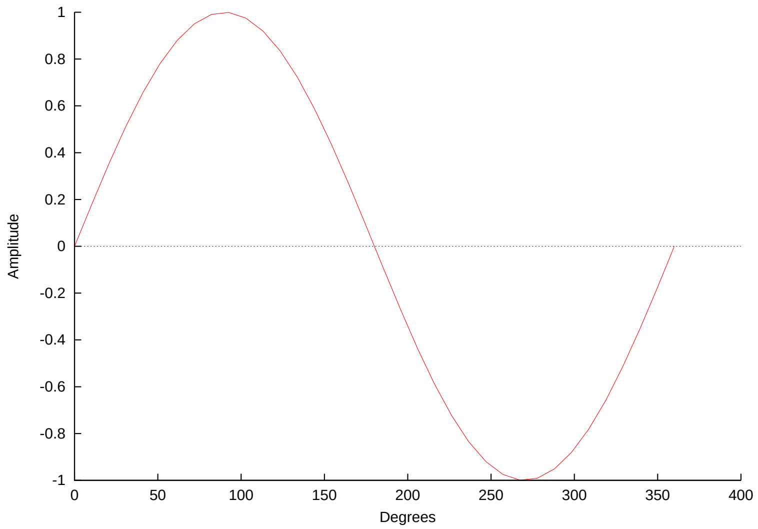
label label of the axis.

Description
Set the plot axis labels. The label for the z-axis can be set only for a 3D plot. If no label is desired, the value of `NULL` can be used as an argument. For example, function call `plot.label(PLOT_AXIS_XY, NULL)` will suppress the labels for both x and y axes of the plot. By default, labels are “x”, “y”, and “z” for the first x, y, and z axes, respectively.

Example
Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    char *xlabel="Degrees", *ylabel="Amplitude";
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::getLabel(), **CPlot::title**().
CPlot::getTitle().

CPlot::legend

Synopsis in Ch

#include <**chplot.h**>
void legend(**string_t** *legend*, **int** *num*);

Synopsis in C++

#include <**chplot.h**>
void legend(**char** * *legend*, **int** *num*);

Purpose

Specify a legend string for a previously added data set.

Return Value

None.

Parameters

legend The legend string.

num The data set the legend is added to.

Description

The legend string is added to a plot legend located in the upper-right corner of the plot by default. Numbering of the data sets starts with zero. New legends will replace previously specified legends.
This member function shall be called after plotting data have been added by member functions **CPlot::data2D**(), **CPlot::data2DCurve**(), **CPlot::data3D**(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **CPlot::dataFile**().

Bugs

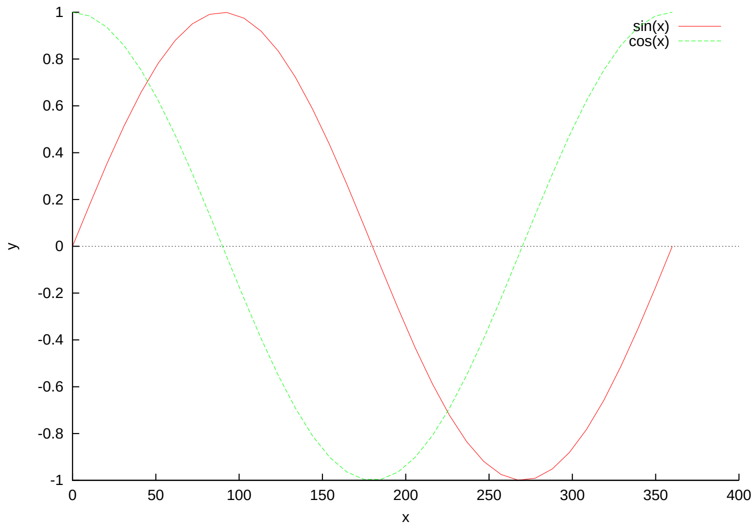
The legend string may not be displayed for a data set with a single point. To suppress the legend string completely, passs value of `" "` to the argument `legend`. Use **CPlot::text**() to add a legend for the data set.

Example

Compare with the output for the example in **CPlot::legendLocation**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM], y2[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
       y2[i] = cos(x[i]*M_PI/180); // Y-axis data
    }

    plot.data2DCurve(x, y, NUM);
    plot.data2DCurve(x, y2, NUM);
    plot.legend("sin(x)", 0);
    plot.legend("cos(x)", 1);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**dataFile**(), **CPlot**::**legendLocation**(), **CPlot**::**legendOption**().

CPlot::legendLocation

Synopsis in Ch
#include <**chplot.h**>
void legendLocation(**double** *x*, **double** *y*, ... /* [**double** *z*] */);

Synopsis in C++
#include <**chplot.h**>
void legendLocation(**double** *x*, **double** *y*);
void legendLocation(**double** *x*, **double** *y*, **double** *z*);

Syntax in Ch and C++
legendLocation(*x*, *y*)
legendLocation(*x*, *y*, *z*)

Purpose
Specify the plot legend (if any) location

Return Value
None.

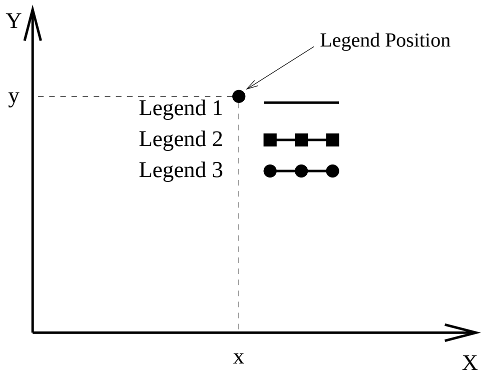
Parameters

x The x coordinate of the legend.

y The y coordinate of the legend.

z The z coordinate of the legend.

Description
This function specifies the position of the plot legend using plot coordinates. The position specified is the location of the top right of the box for the markers and labels of the legend, as shown below. By default, the location of the legend is near the upper-right corner of the plot. For a two-dimensional plot, the third argument is not necessary.

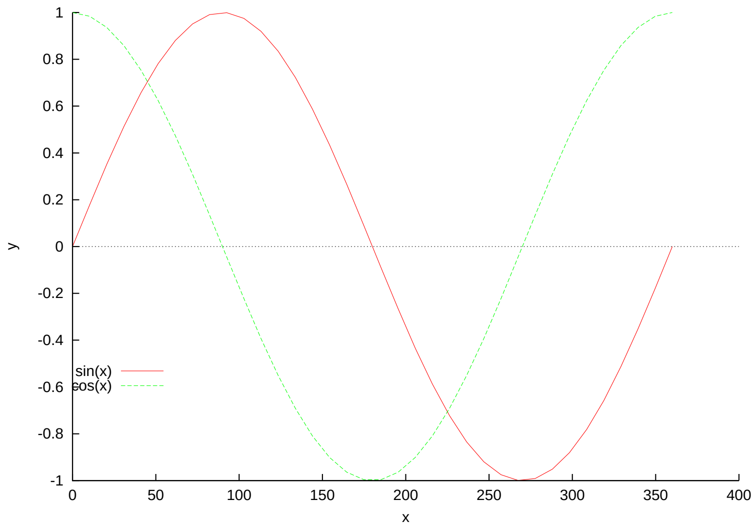


Example

Compare with the output for the example in **CPlot::legend()**.

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM], y2[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
       y2[i] = cos(x[i]*M_PI/180); // Y-axis data
    }
    plot.data2DCurve(x, y, NUM);
    plot.data2DCurve(x, y2, NUM);
    plot.legend("sin(x)", 0);
    plot.legend("cos(x)", 1);
    plot.legendLocation(60, -.5);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::legend(), **CPlot::legendOption()**.

CPlot::legendOption

Synopsis in Ch and C++

#include <chplot.h>

void legendOption(**char** * *option*);

Syntax in Ch and C++
legendOption(*option*)

Purpose
Set options for legends of a plot.

Return Value
None.

Parameters
option The options for legends of a plot.

Description
The optional argument `option` of string type with the following values can be used to fine tune the legends of a plot.

```
{{inside | outside} | {lmargin | rmargin | tmargin | bmargin}
 | {at <position>}}
{left | right | center} {top | bottom | center}
{vertical | horizontal} {Left | Right}
{{no}reverse} {{no}invert}
{samplen <sample_length>} {spacing <vertical_spacing>}
{width <width_increment>}
{height <height_increment>}
{{no}autotitle {columnheader}}
{{no}box { {linestyle | ls <line_style>}
           | {linetype | lt <line_type>}
             {linewidth | lw <line_width>}}}
```

Legends are stacked according to ‘vertical‘ or ‘horizontal‘. In the case of ‘vertical‘, the legendskey occupy as few columns as possible. That is, legends are aligned in a column until running out of vertical space at which point a new column is started. In the case of ‘horizontal‘, the legends occupy as few rows as possible.

By default the legends are placed in the upper right inside corner of the graph. The keywords ‘left‘, ‘right‘, ‘top‘, ‘bottom‘, ‘center‘, ‘inside‘, ‘outside‘, ‘lmargin‘, ‘rmargin‘, ‘tmargin‘, ‘bmargin‘(, ‘above‘, ‘over‘, ‘below‘ and ‘under‘) may be used to automatically place the legends in other positions of the graph. Also an ‘at `<positionx>`‘ may be given to indicate precisely where the plot should be placed. In this case, the keywords ‘left‘, ‘right‘, ‘top‘, ‘bottom‘ and ‘center‘ serve an analogous purpose for alignment.

To understand positioning, the best concept is to think of a region, i.e., inside/outside, or one of the margins. Along with the region, keywords ‘left/center/right‘ (l/c/r) and ‘top/center/bottom‘ (t/c/b) control where within the particular region the legends should be placed.

When in ‘inside‘ mode, the keywords ‘left‘ (l), ‘right‘ (r), ‘top‘ (t), ‘bottom‘ (b), and ‘center‘ (c) push the legends out toward the plot boundary as illustrated:

```
t/l   t/c   t/r

c/l    c    c/r
```

```
    b/l   b/c   b/r
```

When in ‘outside‘ mode, automatic placement is similar to the above illustration, but with respect to the view, rather than the graph boundary. That is, a border is moved inward to make room for the key outside of the plotting area, although this may interfere with other labels and may cause an error on some devices. The particular plot border that is moved depends upon the position described above and the stacking direction. For options centered in one of the dimensions, there is no ambiguity about which border to move. For the corners, when the stack direction is ‘vertical‘, the left or right border is moved inward appropriately. When the stack direction is ‘horizontal‘, the top or bottom border is moved inward appropriately.

The margin syntax allows automatic placement of legends regardless of stack direction. When one of the margins ‘lmargin‘ (lm), ‘rmargin‘ (rm), ‘tmargin‘ (tm), and ‘bmargin‘ (bm) is combined with a single, non-conflicting direction keyword, the following illustrated positions may contain the legends:

```
        l/tm  c/tm  r/tm

    t/lm                 t/rm

    c/lm                 c/rm

    b/lm                 b/rm

        l/bm  c/bm  r/bm
```

Keywords ‘above‘ and ‘over‘ are synonymous with ‘tmargin‘. For version compatibility, ‘above‘ or ‘over‘ without an additional l/c/r or stack direction keyword uses ‘center‘ and ‘horizontal‘. Keywords ‘below‘ and ‘under‘ are synonymous with ‘bmargin‘. For compatibility, ‘below‘ or ‘under‘ without an additional l/c/r or stack direction keyword uses ‘center‘ and ‘horizontal‘. A further compatibility issue is that ‘outside‘ appearing without an additional t/b/c or stack direction keyword uses ‘top‘, ‘right‘ and ‘vertical‘ (i.e., the same as t/rm above).

The `<position>` can be a simple x,y,z as in previous versions, but these can be preceded by one of five keywords (‘first‘, ‘second‘, ‘graph‘, ‘screen‘, ‘character‘) which selects the coordinate system in which the position of the first sample line is specified. See ‘coordinates‘ for more details. The effect of ‘left‘, ‘right‘, ‘top‘, ‘bottom‘, and ‘center‘ when `<position>` is given is to align the legends as though it were text positioned using the label command, i.e., ‘left‘ means left align with key to the right of ¡position¿, etc.

Justification of the labels within the key is controlled by ‘Left‘ or ‘Right‘ (default is ‘Right‘). The text and sample can be reversed (‘reverse‘) and a box can be drawn around the legend `(‘box {...}‘)` in a specified ‘linetype‘ and ‘linewidth‘. Note that not all terminal drivers support linewidth selection, though.

By default the first plot label is at the top of the legends and successive labels are entered below it. The ‘invert‘ option causes the first label to be placed at the bottom of the legends, with successive labels entered above it. This option is useful to force the vertical ordering of labels in the legends to match the order of box types in a stacked histogram.

The length of the sample line can be controlled by ‘samplen‘. The sample length is computed as the sum of the tic length and `<sample_length>` times the character width. ‘samplen‘ also affects the positions of point samples in the legends since these are drawn at the midpoint of the sample line, even if the sample line itself is not drawn.

The vertical spacing between lines is controlled by ‘spacing‘. The spacing is set equal to the product of the pointsize, the vertical tic size, and `<vertical_spacing>`. The program will guarantee that the vertical spacing is no smaller than the character height.

The `<width_increment>` is a number of character widths to be added to or subtracted from the length of the string. This is useful only when you are putting a box around the legends and you are using control characters in the text. CPlot class simply counts the number of characters in the string when computing the box width; this allows you to correct it.

The `<height_increment>` is a number of character heights to be added to or subtracted from the height of the key box. This is useful mainly when you are putting a box around the legends, otherwise it can be used to adjust the vertical shift of automatically chosen legend position by `<height_increment>/2`.

The defaults for legends are ‘on‘, ‘right‘, ‘top‘, ‘vertical‘, ‘Right‘, ‘noreverse‘, ‘noinvert‘, ‘samplen 4‘, ‘spacing 1.25‘, and ‘nobox‘. The default `<linetype>` is the same as that used for the plot borders.

The legends are drawn as a sequence of lines, with one plot described on each line. On the right-hand side (or the left-hand side, if ‘reverse‘ is selected) of each line is a representation that attempts to mimic the way the curve is plotted. On the other side of each line is the text description (the line title), obtained from the member function **CPlot**::legend(). The lines are vertically arranged so that an imaginary straight line divides the left- and right-hand sides of the key. It is the coordinates of the top of this line that are specified in the argument of `option` or member function **CPlot**::**legendLocation**(). For a 2D plot, only the x and y coordinates are used to specify the line position. For a 3D plot, x, y and z are all used as a 3-d location mapped using the same mapping as the graph itself to form the required 2-d screen position of the imaginary line.

When using the TeX or PostScript drivers, or similar drivers where formatting information is embedded in the string, CPlot class is unable to calculate correctly the width of the string for the legend positioning. If the legends are to be positioned at the left, it may be convenient to use the combination ‘left Left reverse‘. The box and gap in the grid will be the width of the literal string.

For a 3D countour, the contour labels will be listed in the legends.

Examples:

This places the legends at coordinates 2,3.5,2 in the default (first) coordinate system:

```
plot.legendOption("2,3.5,2");
```

This places the legends below the graph:

```
plot.legendOption("below");
```

This places the legends in the bottom left corner, left-justifies the text, and draws a box around it in linetype 3:

```
plot.legendOption("left bottom Left box 3");
```

Examples

See three examples on pages 184, 185, and 186 for **CPlot**:**plotType**().

See Also

CPlot::**legend**(), **CPlot**::**legendLocation**().

CPlot::line

Synopsis in C++

#include <**chplot.h**>

int line(**double** *x1*, **double** *y1*, **double** *x2*, **double** *y2*);

int line(**double** *x1*, **double** *y1*, **double** *z1*, **double** *x2*, **double** *y2*, **double** *z2*);

Synopsis in Ch
#include <chplot.h>
int line(**double** *x1*, **double** *y1*, ... /* **double** *x2*, **double** *y2*; **double** *z1*, **double** *x2*, **double** *y2*, **double** *z2* */);

Syntax in Ch and C++
line(*x1*, *y1*, *x2*, *y2*);
line(*x1*, *y1*, *z1*, *x2*, *y2*, *z2*);

Purpose
Add a line to a plot.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x1 The x coordinate of the first endpoint of the line.

y1 The y coordinate of the first endpoint of the line.

z1 The z coordinate of the first endpoint of the line. This argument is ignored for 2D plots.

x2 The x coordinate of the second endpoint of the line.

y2 The y coordinate of the second endpoint of the line.

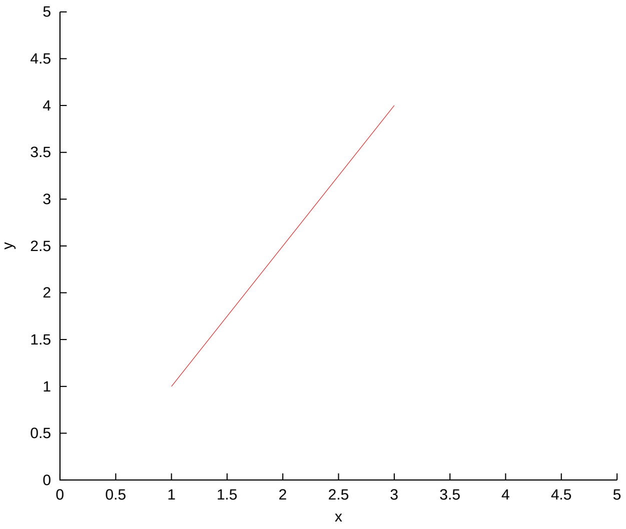
z2 The z coordinate of the second endpoint of the line. This argument is ignored for 2D plots.

Description
This function adds a line to a plot. It is a convenience function for creation of geometric primitives. A line added with this function is counted as a data set for later calls to **CPlot**::**legend**() and **CPlot**::**plotType**(). For 2D rectangular and 3D cartesian plots, (*x1*, *y1*, *z1*) and (*x2*, *y2*, *z2*) are the coordinates of the endpoints of the line, specified in units of the x, y, and z axes. However, for 2D plots, *z1* and *z2* are ignored. For 2D polar and 3D cylindrical plots, the endpoints are specified in polar coordinates where *x* is θ, *y* is r, and *z* is unchanged. Again, for 2D plots, *z1* and *z2* are ignored. For 3D plots with spherical coordinates *x* is θ, *y* is ϕ and *z* is r.

Example 1

```
#include <chplot.h>

int main(){
    double x1 = 1, y1 = 1, x2 = 3, y2 = 4;
    class CPlot plot;

    plot.line(x1, y1, x2, y2);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_XY, 0, 5, .5);
    plot.plotting();
    return 0;
}
```

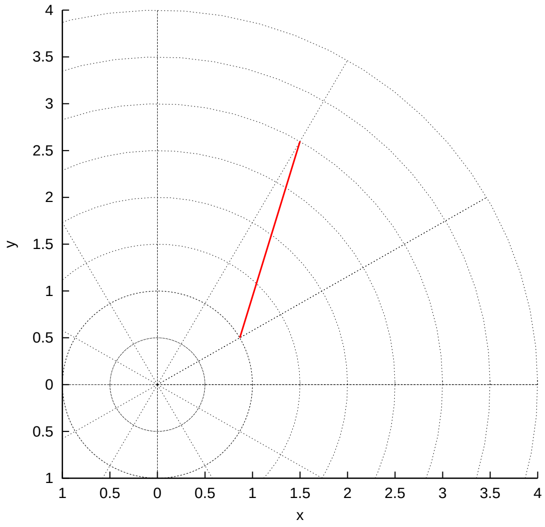
Output



Example 2

```
#include <chplot.h>

int main(){
    double theta1 = 30, theta2 = 60, r1 = 1, r2 = 3;
    class CPlot plot;

    plot.grid(PLOT_ON);
    plot.polarPlot(PLOT_ANGLE_DEG);
    plot.line(theta1, r1, theta2, r2);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.lineType(0, 1, 3);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_XY, -1, 4, .5);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::**data2D**(), **CPlot**::**data2DCurve**(), **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **CPlot**::**circle**(), **CPlot**::**outputType**(), **CPlot**::**plotType**(), **CPlot**::**point**(), **CPlot**::**polygon**(), **CPlot**::**rectangle**().

CPlot::lineType

Synopsis in Ch
#include <**chplot.h**>
void lineType(**int** *num*, **int** *line_type*, **int** *line_width*], ... /* [**char** *line_color*] */);

Synopsis in C++
#include <**chplot.h**>
void lineType(**int** *num*, **int** *line_or_point_type*, **int** *line_width*);
void lineType(**int** *num*, **int** *line_or_point_type*, **int** *line_width*, **char** *line_color*);

Syntax in Ch and C++
lineType(*num*, *line_type*, *line_width*)
lineType(*num*, *line_type*, *line_width*, *line_color*)

Purpose
Set the line type, width, and color for lines, impulses, steps, etc.

Return Value
None.

Parameters

num The data set to which the line type, width, and color apply.

line_type An integer index representing the line type for drawing. Use the same value for different curves so that each curve with the same style, and same color by default.

line_width A scaling factor for the line width. The line width is *line_width* multiplied by the default width.

line_color color for the line.

Description
Set the desired line type, width, and color for a previously added data set.

Numbering of the data sets starts with zero. The line style and/or marker type for the plot are selected automatically. The default line type, width, and color can be changed by this member function.

The *line_type* specifies an index for the line type used for drawing the line. The line type varies depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line or make it dashed or dotted. All terminals support at least six different line types. By default, the line type is 1. The *line_width* specifies the line width. The line width is *line_width* multiplied by the default width. Typically the default width is one pixel.

An optional fourth argument can specify the color of a line by a color name or RGB value, such as `"blue"` or `"#0000ff"` for color blue. The default line type, width, and color can be changed by the function call

```
    plot.lineType(num, linetype, linewidth, "blue");
```

The color of the line is specified as blue in this example. The valid color names and their corresponding GRB values are listed below.

```
 Color Name        Hexadecimal R   G    B
                               values
-------------------------------------------
 white              #ffffff = 255 255 255
 black              #000000 =   0   0   0
 gray0              #000000 =   0   0   0
 grey0              #000000 =   0   0   0
 gray10             #1a1a1a =  26  26  26
 grey10             #1a1a1a =  26  26  26
 gray20             #333333 =  51  51  51
 grey20             #333333 =  51  51  51
 gray30             #4d4d4d =  77  77  77
 grey30             #4d4d4d =  77  77  77
 gray40             #666666 = 102 102 102
 grey40             #666666 = 102 102 102
 gray50             #7f7f7f = 127 127 127
 grey50             #7f7f7f = 127 127 127
 gray60             #999999 = 153 153 153
 grey60             #999999 = 153 153 153
 gray70             #b3b3b3 = 179 179 179
 grey70             #b3b3b3 = 179 179 179
 gray80             #cccccc = 204 204 204
 grey80             #cccccc = 204 204 204
 gray90             #e5e5e5 = 229 229 229
 grey90             #e5e5e5 = 229 229 229
 gray100            #ffffff = 255 255 255
 grey100            #ffffff = 255 255 255
 gray               #bebebe = 190 190 190
 grey               #bebebe = 190 190 190
 light-gray         #d3d3d3 = 211 211 211
 light-grey         #d3d3d3 = 211 211 211
 dark-gray          #a9a9a9 = 169 169 169
 dark-grey          #a9a9a9 = 169 169 169
 red                #ff0000 = 255   0   0
 light-red          #f03232 = 240  50  50
 dark-red           #8b0000 = 139   0   0
 yellow             #ffff00 = 255 255   0
 light-yellow       #ffffe0 = 255 255 224
 dark-yellow        #c8c800 = 200 200   0
 green              #00ff00 =   0 255   0
 light-green        #90ee90 = 144 238 144
 dark-green         #006400 =   0 100   0
 spring-green       #00ff7f =   0 255 127
```

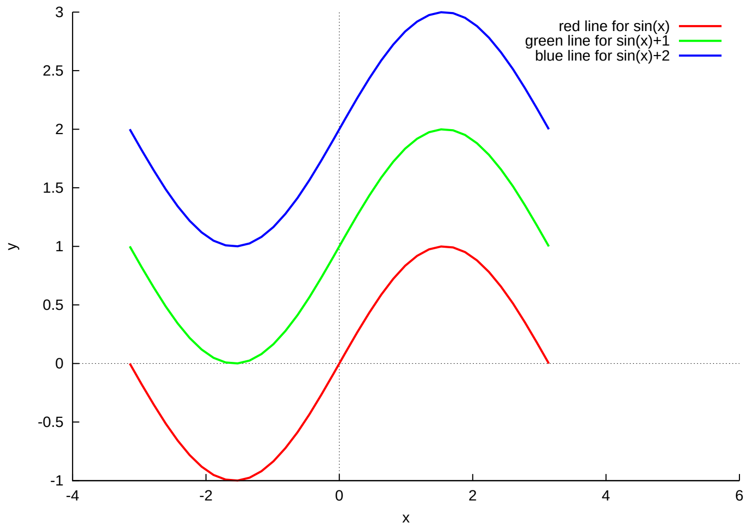
```
  forest-green       #228b22 =  34 139  34
  sea-green          #2e8b57 =  46 139  87
  blue               #0000ff =   0   0 255
  light-blue         #add8e6 = 173 216 230
  dark-blue          #00008b =   0   0 139
  midnight-blue      #191970 =  25  25 112
  navy               #000080 =   0   0 128
  medium-blue        #0000cd =   0   0 205
  royalblue          #4169e1 =  65 105 225
  skyblue            #87ceeb = 135 206 235
  cyan               #00ffff =   0 255 255
  light-cyan         #e0ffff = 224 255 255
  dark-cyan          #008b8b =   0 139 139
  magenta            #ff00ff = 255   0 255
  light-magenta      #f055f0 = 240  85 240
  dark-magenta       #8b008b = 139   0 139
  turquoise          #40e0d0 =  64 224 208
  light-turquoise    #afeeee = 175 238 238
  dark-turquoise     #00ced1 =   0 206 209
  pink               #ffc0cb = 255 192 203
  light-pink         #ffb6c1 = 255 182 193
  dark-pink          #ff1493 = 255  20 147
  coral              #ff7f50 = 255 127  80
  light-coral        #f08080 = 240 128 128
  orange-red         #ff4500 = 255  69   0
  salmon             #fa8072 = 250 128 114
  light-salmon       #ffa07a = 255 160 122
  dark-salmon        #e9967a = 233 150 122
  aquamarine         #7fffd4 = 127 255 212
  khaki              #f0e68c = 240 230 140
  dark-khaki         #bdb76b = 189 183 107
  goldenrod          #daa520 = 218 165  32
  light-goldenrod    #eedd82 = 238 221 130
  dark-goldenrod     #b8860b = 184 134  11
  gold               #ffd700 = 255 215   0
  beige              #f5f5dc = 245 245 220
  brown              #a52a2a = 165  42  42
  orange             #ffa500 = 255 165   0
  dark-orange        #ff8c00 = 255 140   0
  violet             #ee82ee = 238 130 238
  dark-violet        #9400d3 = 148   0 211
  plum               #dda0dd = 221 160 221
  purple             #a020f0 = 160  32 240
```

Example 1

```
/* File: lineType.cpp */
#include <math.h>
#include <chplot.h>
```

```
#define NUM 36
int main() {
    double x[NUM], y[NUM], y2[NUM], y3[NUM];
    int line_type = 1, line_width = 4, datasetnum = 0, i;
    CPlot plot;

    for(i=0; i<NUM; i++) {
      x[i] = -M_PI + i*2*M_PI/(NUM-1);  // lindata(-M_PI, M_PI, x);
      y[i] = sin(x[i]);
      y2[i] = sin(x[i])+0.5;
      y2[i] = sin(x[i])+1;
      y3[i] = sin(x[i])+2;
    }

    plot.data2DCurve(x, y, NUM);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width, "red");
    plot.legend("red line for sin(x)", datasetnum);

    plot.data2DCurve(x, y2, NUM);
    plot.plotType(PLOT_PLOTTYPE_LINES, ++datasetnum);
    plot.lineType(datasetnum, line_type, line_width, "green");
    plot.legend("green line for sin(x)+1", datasetnum);

    plot.data2DCurve(x, y3, NUM);
    plot.plotType(PLOT_PLOTTYPE_LINES, ++datasetnum);
    plot.lineType(datasetnum, line_type, line_width, "blue");
    plot.legend("blue line for sin(x)+2", datasetnum);

    plot.axisRange(PLOT_AXIS_X, -4, 6);
    plot.plotting();
}
```

Output



Additional Examples

See Program 2.13 and Figure 2.13, Program 2.14 and Figure 2.14, and programs and their generated figures for **CPlot**::**plotType**().

See Also
CPlot::**plotType**(), **CPlot**::**pointType**().

CPlot::margins

Synopsis
#include <**chplot.h**>
void margins(**double** *left*, **double** *right*, **double** *top*, **double** *bottom*);

Purpose
Set the size of the margins around the edge of the plot.

Return Value
None.

Parameters
left The size of the left margin in character width.

right The size of the right margin in character width.

top The size of the top margin in character height.

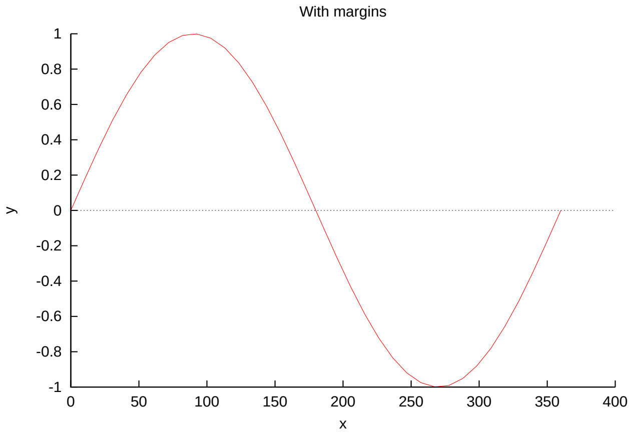
bottom The size of the bottom margin in character height.

Description
By default, the plot margins are calculated automatically. They can be set manually with this function. Specifying a negative value for a margin causes the default value to be used.

Example
Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.data2DCurve(x, y, NUM);
    plot.margins(15, 10, 5, 7);
    plot.title("With margins");
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::**borderOffsets**().

CPlot::origin

Synopsis

#include <chplot.h>
void origin(**double** *x_orig*, **double** *y_orig*);

Purpose

Set the location of the origin for the drawing of the plot.

Return Value

None.

Parameters

x_orig The x coordinate of the origin for the drawing of the plot.

y_orig The y coordinate of the origin for the drawing of the plot.

Description

This function specifies the location of the origin for the drawing of the plot. This is the location of the bottom left corner of the bounding box of the plot, not the location of the origin of the plot coordinate system. The values *x_org* and *y_org* are specified as numbers between 0 and 1, where (0,0) is the bottom left of the plot area (the plot window) and (1,1) is the top right. This function is used internally by method **CPlot**::**subplot**() in conjunction with method **CPlot**::**size**().

See Also

CPlot::**getSubplot**(), **CPlot**::**size**(), **CPlot**::**subplot**().

CPlot::outputType

Synopsis in Ch
#include <chplot.h>
void outputType(**int** *outputtype*, ... /* [**string_t** *terminal*, **string_t** *filename*] */);

Synopsis in C++
#include <chplot.h>
void outputType(**int** *outputtype*);
void outputType(**int** *outputtype*, [**char** * *terminal*);
void outputType(**int** *outputtype*, [**char** * *terminal*, **char** * *filename*]);

Syntax in Ch and C++
outputType(*outputtype*)
outputType(**PLOT_OUTPUTTYPE_DISPLAY**)
outputType(**PLOT_OUTPUTTYPE_STREAM**, *terminal*)
outputType(**PLOT_OUTPUTTYPE_FILE**, *terminal*, *filename*)

Purpose
Set the output type for a plot.

Return Value
None.

Parameters
outputtype This can have any of the following values:

PLOT_OUTPUTTYPE_DISPLAY Display the plot on the screen. The plot is displayed in its own separate window. A plot window can be closed by pressing the ‘q’ key in the X-Windows system.

PLOT_OUTPUTTYPE_STREAM Output the plot as a standard output stream. This output type is useful for CGI (Common Gateway Interface) when a Ch program is used as CGI script in a Web server.

PLOT_OUTPUTTYPE_FILE Output the plot to a file in one of a variety of formats. If this output option is selected two additional arguments are necessary: the *terminal* type and *filename*.

terminal Supported terminal types when gnuplot is used as a plotting engine are as follow:

Terminal	Description
aifm	Adobe Illustrator 3.0.
corel	EPS format for CorelDRAW.
dxf	AutoCAD DXF.
dxy800a	Roland DXY800A plotter.
eepic	Extended LaTeXpicture.
emtex	LaTeXpicture with emTeX specials.

epson-180dpi	Epson LQ-style 24-pin printer with 180dpi.
epson-60dpi	Epson LQ-style 24-pin printers with 60dpi.
epson-lx800	Epson LX-800, Star NL-10 and NX-100.
excl	Talaris printers.
fig	Xfig 3.1.
gif	GIF file format.
gpic	gpic/groff package.
hp2648	Hewlett Packard HP2647 an HP2648.
hp500c	Hewlett Packard DeskJet 500c.
hpdj	Hewlett Packard DeskJet 500.
hpgl	HPGL output.
hpljii	HP LaserJet II.
hppj	HP PaintJet and HP3630 printers.
latex	LaTeXpicture.
mf	MetaFont.
mif	Frame Maker MIF 3.00.
nec-cp6	NEC CP6 and Epson LQ-800.
okidata	9-pin OKIDATA 320/321 printers.
pcl5	Hewlett Packard LaserJet III.
pbm	Portable BitMap.
png	Portable Network Graphics.
postscript	Postscript.
pslatex	LaTeXpicture with postscript specials.
pstricks	LaTeXpicture with PSTricks macros.
starc	Star Color Printer.
tandy-60dpi	Tandy DMP-130 series printers.
texdraw	LaTeXtexdraw format.
tgif	TGIF X-Window drawing format.
tpic	LaTeXpicture with tpic specials.

`aifm` Output an Adobe Illustrator 3.0 file. The format for the *terminal* string is:

```
"aifm [colormode] [\"fontname\"] [fontsize]"
```

`colormode` can be `color` or `monochrome`.
`fontname` is the name of a valid PostScript font.
`fontsize` is the size of the font in points.

Defaults are `monochrome`, `"Helvetica"` and 14pt.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "aifm color", "plot.aif");`

`corel` Output EPS format for CorelDRAW.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "corel", "plot.eps");`

`dxf` Output AutoCAD DXF file.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "dxf", "plot.dxf");`

`dxy800a` Output file for Roland DXY800A plotter.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "dxy800a", "plot.dxy");`

`eepic` Output extended LATEXpicture.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "eepic", "plot.tex");`

`emtex` Output LATEXpicture with emTeX specials.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "emtex", "plot.tex");`

`epson-180dpi` Epson LQ-style 24-pin printers with 180dpi.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "epson-180dpi", "plot");`

`epson-60dpi` Epson LQ-style 24-pin printers with 60dpi.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "epson-60dpi", "plot");`

`epson-lx800` Epson LX-800, Star NL-10 and NX-100.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "epson-lx800", "plot");`

`excl` Talaris printers such as EXCL Laser printer and 1590.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "excl", "plot");`

`fig` Xfig 3.1 file. The format for the *terminal* string is:

```
"fig [colormode] [size] [pointsmax num_points] [orientation]
[units] [fontsize fntsize] [size xsize ysize] [thickness width]
[depth layer]"
```

`colormode` can be `monochrome` or `color`
`size` can be `small` or `big`
`num_points` is the maximum number of points per polyline.
`orientation` can be `landscape` or `portrait`. `units` can be `metric` or `inches`.
`fontsize` is the size of the text font in points. Must be preceded by the `fontsize` keyword.
`xsize` and `ysize` set the size of the drawing area. Must be preceded by the `size` keyword.

`width` is the line thickness. Must be preceded by the `thickness` keyword.
`layer` is the line depth. Must be preceded by the `depth` keyword.

Default values are: `monochrome small pointsmax 1000 landscape inches fontsize 10 thickness 1 depth 10`.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "fig color big", "plot.fig");`

`gif` GIF file format. The format for the *terminal* string is:

```
"gif [transparent] [interlace] [font_size] [size x,y]
[color0 color1 color2 ...]
```

Specifying the `transparent` keyword will generate a transparent GIF. By default, white is the transparent color.
Specifying the `interlace` key word will generate an interlaced GIF.
`font_size` is `small` (6x12 pixels), `medium` (7x13 pixels), or `large` (8x16 pixels).
`x` and `y` are the image size in pixels. Must be preceded by the `size` keyword.
`color`s are specified in the format ”xrrggbb” where x is the character ”x” and ”rrggbb” are the RGB components of the color in hexadecimal. A maximum of 256 colors can be set. If the GIF is transparent, the first color is used as the transparent color.

The default values are: `small size 640,480`

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "gif size 1024,768", "plot.gif");`

`gpic` Output for use with the Free Software Foundation gpic/groff p package. The format for the *terminal* string is:

```
"gpic [x] [y]"
```

where `x` and `y` are the location of the origin. Default is (0,0).

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "gpic 5 5", "plot.gpic");`

`hp2633a` Hewlett Packard HP2623A.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hp2633a", "plot");`

`hp2648` Hewlett Packard HP2647 an HP2648.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hp2648", "plot");`

`hp500c` Hewlett Packard DeskJet 500c. The format for the *terminal* string is:

```
"hp500c [resolution] [compression]"
```

`resolution` can be `75`, `100`, `150`, or `300` dpi.
`compression` can be `rle` or `tiff`.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hp500c 100 tiff", "plot");`

`"hpdj`: Hewlett Packard DeskJet 500. The format for the *terminal* string is:

```
"hp500c [resolution]"
```

`resolution` can be `75`, `100`, `150`, or `300` dpi. Default is `75`.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hpdj 100", "plot");`

`hpgl` Produces HPGL output for devices such as the HP7475A plotter. The format for the *terminal* string is:

```
"hpgl [num_of_pens] [eject]"
```

`num_of_pens` is the number of available pens. The default is 6.
`eject` is a keyword that tells the plotter to eject a page when done.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hpgl 4", "plot");`

`hpljii` HP LaserJet II

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hpljii", "plot");`

item[`hppj`] HP PaintJet and HP3630 printers. The format for the *terminal* string is:

```
"hppj [font]"
```

`font` can be `FNT5X9`, `FNT9x17`, or `FNT13X25`. Default is `FNT9x17`.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "hppj FNT5X9", "plot");`

`latex` Output a LaTeXpicture. The format of the *terminal* string is:

```
"latex [font] [size]"
```

where `font` can be `courier` or `roman` and `size` can be any point size. Default is `roman` 10pt.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "latex", "plot.tex");`

`mf` Output file for the MetaFont program.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "mf", "plot.mf");`

`mif` Output Frame Maker MIF file format version 3.00. The format of the *terminal* string is:

`"mif pentype curvetype"`

`pentype` can be: `colour` or `monochrome`.

`curvetype` can be:

`polyline` curves drawn as continuous lines
`vectors` curves drawn as a collection of vectors

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "mif colour vectors", "plot.mif");`

`nec-cp6` Generic 24-pin printer such as NEC CP6 and Epson LQ-800.

The format for the *terminal* string is:

`"nec-cp6a [option]`

`option` can be `monochrome`, `colour`, or `draft`.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "nec-cp6a draft", "plot");`

`okidata` 9-pin OKIDATA 320/321 printers.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "okidata", "plot");`

`pcl5` Hewlett Packard LaserJet III. This actually produces HPGL-2. The format of the *terminal* string is:

`"pcl5 [mode] [font] [fontsize]"`

`mode` is `landscape` or `portrait`.
`font` is `stick`, `univers`, or `cg_times`.
`fontsize` is the font size in points.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "pcl5 landscape", "plot");`

`pbm` Output a Portable BitMap. The format for the *terminal* string is:

```
"pbm [fontsize] [colormode]"
```

`fontsize` is `small`, `medium`, or `large`.
`colormode` is `monochrome`, `gray`, or `color`.

Example:

```
plot.outputType(PLOT_OUTPUTTYPE_FILE, "pbm medium gray",
"plot.pbm");
```

`png` Portable Network Graphics format. The format of the *terminal* string is:

`fontsize` can be `small`, `medium`, or `large`.
Default is `small` with the output size of 640x480 pixel. Use member function **CPlot**::**size**() to change the size of the plot.

Example:

```
plot.outputType(PLOT_OUTPUTTYPE_FILE, "png", "plot.png");
```

`postscript` This produces a postscript file. The format for the *terminal* string is:

```
"postscript [mode] [colormode] [dash] [\"fontname\"] [fontsize]"
```

`mode` can be `landscape`, `portrait`, `eps`, or `default`
`colormode` can be `color` or `monochrome`.
`dash` can be `solid` or `dashed`.
`fontname` is the name of a valid PostScript font.
`fontsize` is the size of the font in points.

The `default` mode is `landscape`, `monochrome`, `dashed`, `"Helvetica"`, 14pt.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE,"postscript eps \"Times\" 11", "plot.eps");`

`pslatex` Output LaTeXpicture with postscript specials.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "pslatex", "plot.tex");`

`pstricks` Output LaTeXpicture with PSTricks macros.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "pstricks", "plot.tex");`

`starc` Star Color Printer.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "starc", "plot");`

`tandy-60dpi` Tandy DMP-130 series printers.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "tandy-60dpi", "plot");`

`texdraw` Output LaTeXtexdraw format.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "texdraw", "plot.tex");`

`tgif` Output TGIF X-Window drawing format.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "tgif", "plot.tgif");`

`tpic` Output LaTeXpicture with tpic specials.

Example: `plot.outputType(PLOT_OUTPUTTYPE_FILE, "tpic", "plot.tex");`

filename The filename the plot is saved to. On machines that support `popen()` functions, the output can also be piped to another program by placing the '`|`' character in front of the command name and using it as the *filename*. For example, on Unix systems, setting *terminal* to "`postscript`" and *filename* to "`|lp`" could be used to send a plot directly to a postscript printer.

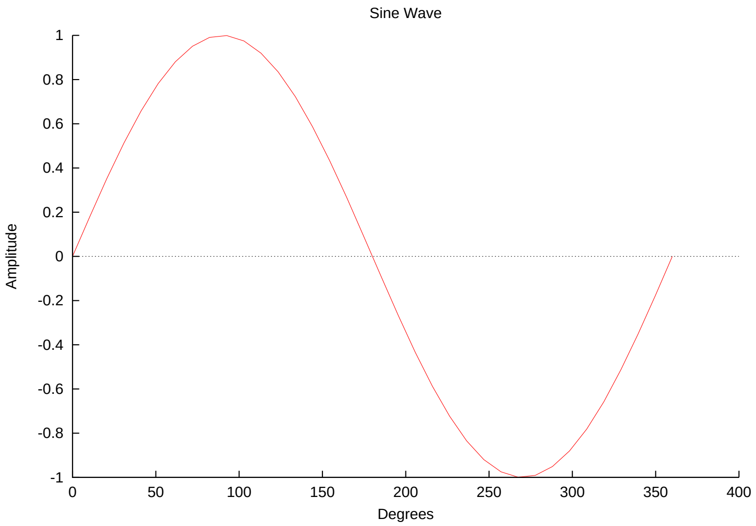
Description

This function is used to display a plot on the screen, save a plot to a file, or generate a plot to the stdout stream in GIF format for use on the World Wide Web. By default, the output type is **PLOT_OUTPUTTYPE_DISPLAY**.

Example 1

```
/* a plot is created in postscript file plot.eps */
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    char *title="Sine Wave", *xlabel="Degrees", *ylabel="Amplitude";
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.title(title);
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.data2DCurve(x, y, NUM);
    plot.outputType(PLOT_OUTPUTTYPE_FILE, "postscript eps color", "plot.eps");
    plot.plotting();
    return 0;
}
```

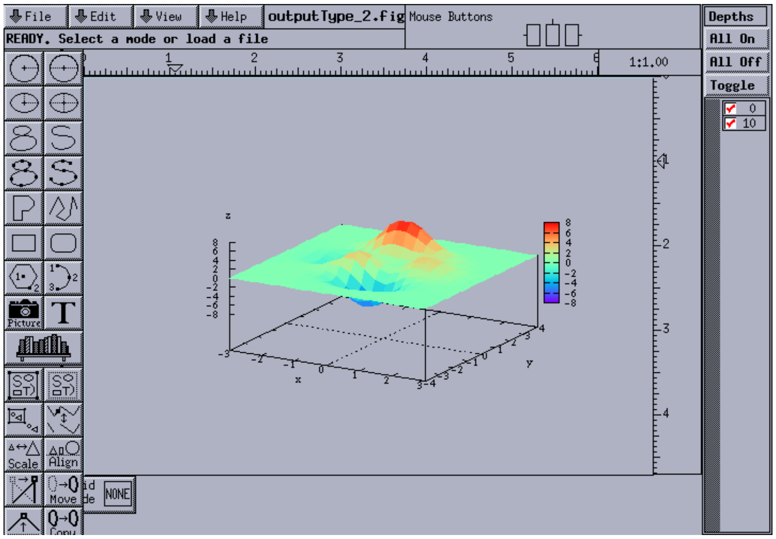
Output



Example 2

```
/* In this example, the plot is saved as an xfig file first.
   Next the xfig program is invoked. The plot can then be edited using xfig in Unix */
#include <math.h>
#include <chplot.h>

#define NUMX 20
#define NUMY 30
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    int i,j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -3 + i*6.0/(NUMX-1); // linspace(x, -3, 3);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -4 + i*8.0/(NUMY-1); // linspace(y, -4, 4);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[NUMY*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.outputType(PLOT_OUTPUTTYPE_FILE, "fig", "../output/outputType_2.fig");
    plot.plotting();
    return 0;
}
```

Output



Example 3

To run this code as a CGI program in a Web server, place outputType.ch in your `cgi-bin` directory. If you place outputType.ch in a different directory, change `/cgi-bin` to specify the correct location. In your Web browser, open the html file.
outputType.html

```
<! this file is called outputType.html >
<html>
<head>
<title>
Example Plot
</title>
</head>

<img src="/cgi-bin/outputType.ch">

</html>
```

CPlot::plotType

Synopsis in Ch
#include <chplot.h>
void plotType(int *plot_type*, **int** *num*, ... /* [**char** *option*]] */);

Obsolete Synopsis in Ch
#include <chplot.h>
void plotType(int *plot_type*, **int** *num*, ... /* [[**int** *line_type*, **int** *line_width*],
[**int** *point_type*, **int** *point_size*], [**char** *option*]] */);

Synopsis in C++
#include <chplot.h>
void plotType(int *plot_type*, **int** *num*);
void plotType(int *plot_type*, **int** *num*, **char** *option*);

Obsolete Synopsis in C++

#include <chplot.h>
void plotType(int *plot_type*, **int** *num*, **int** *point_type*);
void plotType(int *plot_type*, **int** *num*, **int** *line_or_point_type*, **int** *line_width*);
void plotType(int *plot_type*, **int** *num*, **int** *point_type*, **int** *point_size*);
void plotType(int *plot_type*, **int** *num*, **int** *line_or_point_type*, **int** *line_width*, **int** *point_type*, **int** *point_size*);

Syntax in Ch and C++

plotType(PLOT_PLOTTYPE_LINES, *num*)
plotType(PLOT_PLOTTYPE_IMPULSES, *num*)
plotType(PLOT_PLOTTYPE_FSTEPS, *num*)
plotType(PLOT_PLOTTYPE_HISTEPS, *num*)
plotType(PLOT_PLOTTYPE_POINTS, *num*)
plotType(PLOT_PLOTTYPE_LINESPOINTS, *num*)
plotType([PLOT_PLOTTYPE_STEPS, *num*)
plotType(PLOT_PLOTTYPE_DOTS, *num*)
plotType(PLOT_PLOTTYPE_SURFACES, *num*)
plotType(PLOT_PLOTTYPE_FINANCEBARS, *num*)
plotType(PLOT_PLOTTYPE_BOXES, *num*)
plotType(PLOT_PLOTTYPE_BOXERRORBARS, *num*)
plotType(PLOT_PLOTTYPE_BOXXYERRORBARS, *num*)
plotType(PLOT_PLOTTYPE_XERRORBARS, *num*)
plotType(PLOT_PLOTTYPE_XYERRORBARS, *num*)
plotType(PLOT_PLOTTYPE_YERRORBARS, *num*)
plotType(PLOT_PLOTTYPE_XERRORLINES, *num*)
plotType(PLOT_PLOTTYPE_XYERRORLINES, *num*)
plotType(PLOT_PLOTTYPE_YERRORLINES, *num*)
plotType(PLOT_PLOTTYPE_VECTORS, *num*)
plotType(PLOT_PLOTTYPE_VECTORS, *num*, *option*)
plotType(PLOT_PLOTTYPE_CANDLESTICKS, *num*)
plotType(PLOT_PLOTTYPE_CANDLESTICKS, *num*, *option*)
plotType(PLOT_PLOTTYPE_FILLEDCURVES, *num*)
plotType(PLOT_PLOTTYPE_FILLEDCURVES, *num*, *option*)

Obsolte Syntax in Ch and C++

plotType(PLOT_PLOTTYPE_LINES, *num*, *line_type*, *line_width*)
plotType(PLOT_PLOTTYPE_IMPULSES, *num*, *line_type*, *line_width*)
plotType([PLOT_PLOTTYPE_STEPS, *num*, *line_type*, *line_width*)
plotType(PLOT_PLOTTYPE_FSTEPS, *num*, *line_type*, *line_width*)
plotType(PLOT_PLOTTYPE_HISTEPS, *num*, *line_type*, *line_width*)
plotType(PLOT_PLOTTYPE_POINTS, *num*, *point_type*, *point_size*)
plotType(PLOT_PLOTTYPE_LINESPOINTS, *num*, *line_type*, *line_width*, *point_type*, *point_size*)
plotType(PLOT_PLOTTYPE_DOTS, *num*, *point_type*)
plotType(PLOT_PLOTTYPE_FINANCEBARS, *num*, *line_type*, *line_width*)
plotType(PLOT_PLOTTYPE_BOXES, *num*, *line_type*, *line_width*)
plotType(PLOT_PLOTTYPE_BOXERRORBARS, *num*, *line_type*, *line_width*)
plotType(PLOT_PLOTTYPE_BOXXYERRORBARS, *num*, *line_type*, *line_width*)
plotType(PLOT_PLOTTYPE_XERRORBARS, *num*, *line_type*, *line_width*)

plotType(**PLOT_PLOTTYPE_XYERRORBARS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_YERRORBARS**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_XERRORLINES**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_XYERRORLINES**, *num*, *line_type*, *line_width*)
plotType(**PLOT_PLOTTYPE_YERRORLINES**, *num*, *line_type*, *line_width*)

Use
lineType(*num*, *line_type*, *line_width*)
lineType(*num*, *line_type*, *line_width*, *line_color*)
pointType(*num*, *pint_type*, *point_size*)
pointType(*num*, *point_type*, *point_size*, *point_color*)

Purpose
Set the plot type for a data set.

Return Value
None.

Parameters
plot_type The plot type. Valid values are:

PLOT_PLOTTYPE_LINES Data points are connected with a line. This is the default for 2D plots. When this plot type used for 3D plot, the surface is meshed with wire frames.

PLOT_PLOTTYPE_IMPULSES Display vertical lines from the x-axis (for 2D plots) or the xy plane (for 3D plots) to the data points.

PLOT_PLOTTYPE_STEPS Adjacent points are connected with two line segments, one from (x1,y1) to (x2,y1), and a second from (x2,y1) to (x2,y2). This type is available only for 2D plots.

PLOT_PLOTTYPE_FSTEPS Adjacent points are connected with two line segments, one from (x1,y1) to (x1,y2), and a second from (x1,y2) to (x2,y2). This type is available only for 2D plots.

PLOT_PLOTTYPE_HISTEPS This type is intended for plotting histograms. The point x1 is represented by a horizontal line from ((x0+x1)/2,y1) to ((x1+x2)/2,y1). Adjacent lines are connected with a vertical line from ((x1+x2)/2,y1) to ((x1+x2)/2,y2). This type is available only for 2D plots.

PLOT_PLOTTYPE_POINTS Markers are displayed at each data point.

PLOT_PLOTTYPE_LINESPOINTS Markers are displayed at each data point and connected with a line.

PLOT_PLOTTYPE_DOTS A small dot is displayed at each data point. The optional *point_type* argument effects only the color of the dot.

PLOT_PLOTTYPE_SURFACES Data points are meshed as a smooth surface. This is the default for 3D plots.

PLOT_PLOTTYPE_FILLEDCURVES The `filledcurves` plot type is only relevant to 2D plotting. Three variants are possible. The first two variants require two columns of input data, and may be further modified by the options listed below. The first variant, `closed`, treats the curve itself as a closed polygon. This is the default. The second variant is to fill the area between the curve and a given axis, a horizontal or vertical line, or a point. The third variant requires three columns of input data: the x coordinate and two y coordinates corresponding to two curves sampled at the same set of x coordinates; the area between the two curves is filled.

PLOT_PLOTTYPE_VECTORS For a 2D plot, this plot type draws a vector from (x,y) to (x+xdelta, y+ydelta). Thus it requires four columns of data. It also draws a small arrowhead at the end of the vector. A 3D plot of this plot type is similar, but requires six columns of data. It only works for Cartesian 3D plot. The `option` for this plot type is the same as that for the arrow style defined on page 55 for member function **CPlot**::**arrow**().

PLOT_PLOTTYPE_BOXES The boxes plot type is only relevant to 2-d plotting. It draws a box centered about the given x coordinate from the x axis (not the graph border) to the given y coordinate. The width of the box is obtained in one of three ways. If it is a data plot and the data file has a third column, this will be used to set the width of the box. If not, if a width has been set using the member function **CPlot**::**boxWidth**(), this will be used. If neither of these is available, the width of each box will be calculated automatically so that it touches the adjacent boxes. The interior of the boxes is drawn based on the member function **CPlot**::**boxFill**(). By default, the the box is filled with the background color.

PLOT_PLOTTYPE_BOXERRORBARS The boxerrorbars plot type is only relevant to 2-d data plotting. It is a combination of the `boxes` **PLOT_PLOTTYPE_BOXES** and `yerrorbars` **PLOT_PLOTTYPE_YERRORBARS** plot types. The boxwidth will come from the fourth column if the y errors are in the form of `"ydelta"` and the boxwidth was not previously set equal to -2.0 by the member function **CPlot**::**boxWidth**() or from the fifth column if the y errors are in the form of `"ylow yhigh"`. The special case **CPlot**::**boxWidth**(-2.0) is for four-column data with y errors in the form `"ylow yhigh"`. In this case the boxwidth will be calculated so that each box touches the adjacent boxes. The width will also be calculated in cases where three-column data are used. The box height is determined from the y error in the same way as it is for the yerrorbars style—either from y-ydelta to y+ydelta or from ylow to yhigh, depending on how many data columns are provided. The interior of the boxes is drawn based on the specification by **CPlot**::**boxFill**().

PLOT_PLOTTYPE_BOXXYERRORBARS The `boxxyerrorbars` plot type is only relevant to 2-d data plotting. It is a combination of the `boxes` **PLOT_PLOTTYPE_BOXES** and `xyerrorbars` **PLOT_PLOTTYPE_XYERRORBARS** plot types. The box width and height are determined from the x and y errors in the same way as they are for the `xyerrorbars` plot type — either from xlow to xhigh and from ylow to yhigh, or from x-xdelta to x+xdelta and from y-ydelta to y+ydelta , depending on how many data columns are provided. The interior of the boxes is drawn based on the specification by **CPlot**::**boxFill**().

PLOT_PLOTTYPE_CANDLESTICKS The candlesticks plot type can be used for 2-d data plotting of financial data or for generating box-and-whisker plots of statistical data. Five columns of data are required; in order, these should be the x coordinate (most likely a date) and the opening, low, high, and closing prices. The symbol is a rectangular box, centered horizontally at the x coordinate and limited vertically by the opening and closing prices. A vertical line segment at the x coordinate extends up from the top of the rectangle to the high price and another down to the low. The vertical line will be unchanged if the low and high prices are interchanged.

The width of the rectangle can be controlled by the member function **CPlot**::**boxWidth**().

By default the vertical line segments have no crossbars at the top and bottom. If you want crossbars, which are typically used for box-and-whisker plots, then add the keyword `whiskerbars` to the `option` parameter of the function. By default these whiskerbars extend the full horizontal width of the candlestick, but you can modify this by specifying a fraction of the full width.

By default the rectangle is empty if (open > close), and filled with three vertical bars if (close > open). If filled-boxes support is present, then the rectangle can be colored by the member function **CPlot**::**boxFill**().

Note: To place additional symbols, such as the median value, on a box-and-whisker plot requires additional function call as shown in the example below.

```
plot.dataFile("candlesticks.dat", "using 1:3:2:6:5");
plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 0,
              "linetype 1 linewidth 2 whiskerbars 0.5");
plot.dataFile("candlesticks.dat", "using 1:4:4:4:4");
plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 1, "linetype -1 linewidth 2");
```

It assumed that the data in file `candlesticks.dat` contains the following entries

```
# X   Min   1stQuartile Median 3rdQuartile Max
  1   1.5   2           2.4    4           6.
  2   1.5   3           3.5    4           5.5
  3   4.5   5           5.5    6           6.5
  ...
```

The plot will have crossbars on the whiskers and crossbars are 50% of full width.

PLOT_PLOTTYPE_FINANCEBARS The `financebars` plot type is only relevant for 2-d data plotting of financial data. Five columns of data are required; in order, these should be the x coordinate (most likely a date) and the opening, low, high, and closing prices. The symbol is a vertical line segment, located horizontally at the x coordinate and limited vertically by the high and low prices. A horizontal tic on the left marks the opening price and one on the right marks the closing price. The length of these tics may be changed by **CPlot**::**barSize**(). The symbol will be unchanged if the high and low prices are interchanged.

PLOT_PLOTTYPE_XERRORBARS The `xerrorbars` plot type is only relevant to 2D data plotting. The `xerrorbars` is like `dots`, except that a horizontal error bar is also drawn. At each point (x,y), a line is drawn from (xlow,y) to (xhigh,y) or from (x-xdelta,y) to (x+xdelta,y), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

PLOT_PLOTTYPE_XYERRORBARS The `xyerrorbars` plot type is only relevant to 2D data plotting. The `xyerrorbars` is like `dots`, except that horizontal and vertical error bars are also drawn. At each point (x,y), lines are drawn from (x,y-ydelta) to (x,y+ydelta) and from (x-xdelta,y) to (x+xdelta,y) or from (x,ylow) to (x,yhigh) and from (xlow,y) to (xhigh,y), depending upon the number of data columns provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

If data in a file are provided in an unsupported mixed form, the option `using` filter for **CPlot**::dataFile() should be used to set up the appropriate form. For example, if the data are of the form (x,y,xdelta,ylow,yhigh), then you can use

```
plot.dataFile("datafile", "using 1:2:($1-$3):($1+$3):4:5");
plot.plotType(PLOT_PLOTTYPE_XYERRORBARS, plot.dataSetNum());
```

PLOT_PLOTTYPE_YERRORBARS The `yerrorbars` (or `errorbars`) plot type is only relevant to 2D data plotting. The `yerrorbars` is like `points`, except that a vertical error bar is also drawn. At each point (x,y), a line is drawn from (x,y-ydelta) to (x,y+ydelta) or from (x,ylow) to (x,yhigh), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

PLOT_PLOTTYPE_XERRORLINES The `xerrorlines` plot type is only relevant to 2D data plotting. The `xerrorlines` is like `linespoints`, except that a horizontal error line is also drawn. At each point (x,y), a line is drawn from (xlow,y) to (xhigh,y) or from (x-xdelta,y) to (x+xdelta,y), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

PLOT_PLOTTYPE_XYERRORLINES The `xyerrorlines` plot type is only relevant to 2D data plotting. The `xyerrorlines` is like `linespoints`, except that a horizontal and vertical error lines are also drawn. At each point (x,y), lines are drawn from (x,y-ydelta) to (x,y+ydelta) and from (x-xdelta,y) to (x+xdelta,y) or from (x,ylow) to (x,yhigh) and from (xlow,y) to (xhigh,y), depending upon the number of data columns provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

If data in a file are provided in an unsupported mixed form, the option `using` filter for **CPlot**::dataFile() should be used to set up the appropriate form. For example, if the data are of the form (x,y,xdelta,ylow,yhigh), then you can use

```
plot.dataFile("datafile", "using 1:2:($1-$3):($1+$3):4:5");
plot.plotType(PLOT_PLOTTYPE_XYERRORLINES, plot.dataSetNum());
```

PLOT_PLOTTYPE_YERRORLINES The `yerrorlines` (or `errorlines`) plot type is only relevant to 2D data plotting. The `yerrorlines` is like `linespoints`, except that a vertical error line is also drawn. At each point (x,y), a line is drawn from (x,y-ydelta) to (x,y+ydelta) or from (x,ylow) to (x,yhigh), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless **CPlot**::**barSize**() is called).

num The data set to which the plot type applies.

line_type An integer index representing the line type for drawing. Use the same value for different curves so that each curve with the same color and style.

line_width A scaling factor for the line width. The line width is *line_width* multiplied by the default width.

point_type An integer index representing the desired point type.

point_size A scaling factor for the size of the point used. The point size is *point_size* multiplied by the default size.

option An option string for a plot type to fine tune the plot.

1. The `option` for the plot type **PLOT_PLOTTYPE_VECTORS** is the same as that for the arrow style defined on page 55 for member function **CPlot**::**arrow**().
2. The `option` for the plot type **PLOT_PLOTTYPE_CANDLESTICKS** is as follows.

```
{ {linetype | lt <line_type>}
  {linewidth | lw <line_width}
  {whiskerbars [fraction_value]}
}
```

It specifies line type and line width. The `whiskerbars` extend the full horizontal width of the candlestick. The optional `fraction_value` in the range [0, 1.0] specifies a fraction of the full width of whiskerbars.

3. The `option` for the plot type **PLOT_PLOTTYPE_FILLEDCURVES** is as follows.

   ```
   { [closed | {above | below} {x1 | x2 | y1 | y2}[=<a>] | xy=<x>,<y>]
     {linetype | lt <line_type>}
   }
   ```

   The option `linetype` can be used to change the color for the filled area. For example,

   ```
   plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, num, "y1=0 linetype 1");
   ```

   The first two plot variants for **PLOT_PLOTTYPE_FILLEDCURVES** can be further modified by the options

   ```
   closed   ... just filled closed curve,
   x1       ... x1 axis,
   x2       ... x2 axis, etc for y1 and y2 axes,
   y1=0     ... line y=0 (at y1 axis) ie parallel to x1 axis,
   y2=42    ... line y=42 (at y2 axis) ie parallel to x2, etc,
   xy=10,20 ... point 10,20 of x1,y1 axes (arc-like shape).
   ```

   An example of filling the area between two input curves using three columns of data is as follows.

   ```
     plot.dataFile("datafile", "using 1:2:3");
     plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, plot.dataSetNum());
   ```

   The `above` and `below` in the form

   ```
     above {x1|x2|y1|y2}=<val>
     below {x1|x2|y1|y2}=<val>
   ```

   limit the filled area to one side of the bounding line or curve.

   If the values of `<a>, <x>, <y>` are out of the drawing boundary, then they are moved to the graph boundary. Then the actually filled area in the case of option `xy=<x>,<y>` will depend on the x-range and y-range.

Description

Set the desired plot type for a previously added data set. For 3D plots, only **PLOT_PLOTTYPE_LINES**, **PLOT_PLOTTYPE_IMPULSES**, **PLOT_PLOTTYPE_POINTS**, and **PLOT_PLOTTYPE_LINESPOINTS** are valid. If other types are specified, **PLOT_PLOTTYPE_POINTS** is used.

Some 2D plot types need data with with more than two columns. If the data are in a file, the `using` option of **CPlot**::**dataFile**() can be used to set up the correct columns for the plot type you want. In this discussion, "column" will be used to refer to a column in a data file, an entry in the `using` list, or a column in a two-dimensional array. The data for plotting in a two-dimensonal array can be added to an instance of **CPlot** class by the member function **CPlot**::**data**().

For three columns data, only `xerrorbars, yerrorbars, xerrorlines, yerrorlines, boxes, boxerrorbars`, and `filledcurves` are allowed. If other plot type is used, the type will be changed to `yerrorbars`.

For four columns, only `xerrorbars, yerrorbars, xyerrorbars, xerrorlines`, `yerrorlines, xyerrorlines, boxxyerrorbars`, and `boxerrorbars` are allowed. An illegal plot type will be changed to `yerrorbars`.

Five-column data allow only the `boxerrorbars, financebars`, and `candlesticks` plot types. An illegal style will be changed to `boxerrorbars` before plotting.

Six- and seven-column data only allow the `xyerrorbars`, `xyerrorlines`, and `boxxyerrorbars` plot types. Illegal styles will be changed to `xyerrorbars` before plotting.

Numbering of the data sets starts with zero. New plot types replace previously specified types. The line style and/or marker type for the plot are selected automatically, unless the appropriate combination of *line_type*, *line_width*, *point_type*, and *point_size* are specified. The *line_type* is an optional argument specifying an index for the line type used for drawing the line. The line type varies depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line or make it dashed or dotted. All terminals support at least six different line types. By default, the line type is 1. The *line_width* is an optional argument used to specify the line width. The line width is *line_width* multiplied by the default width. Typically the default width is one pixel. *point_type* is an optional argument used to change the appearance (color and/or marker type) of a point. It is specified with an integer representing the index of the desired point type. All terminals support at least six different point types. *point_size* is an optional argument used to change the size of the point. The point size is *point_size* multiplied by the default size. If *point_type* and *point_size* are set to zero or a negative number, a default value is used.

Portability

The *line_width* and *point_size* options is not supported by all terminal types.

For 3D plots on some systems with output type set to `postscript` (see **CPlot**::**outputType**()), data may not be displayed for **PLOT_PLOTTYPE_DOTS**.

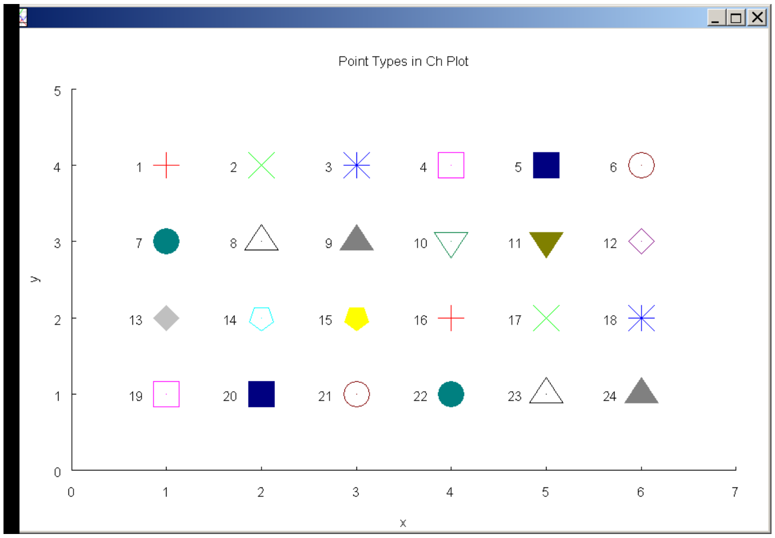
Example 1

This example shows some of the different point types for the default X-window and the `postscript` terminal types (see **CPlot**::**outputType**). In this example the points have a point size of five times the default. The appearance of points for different terminal types may be different.

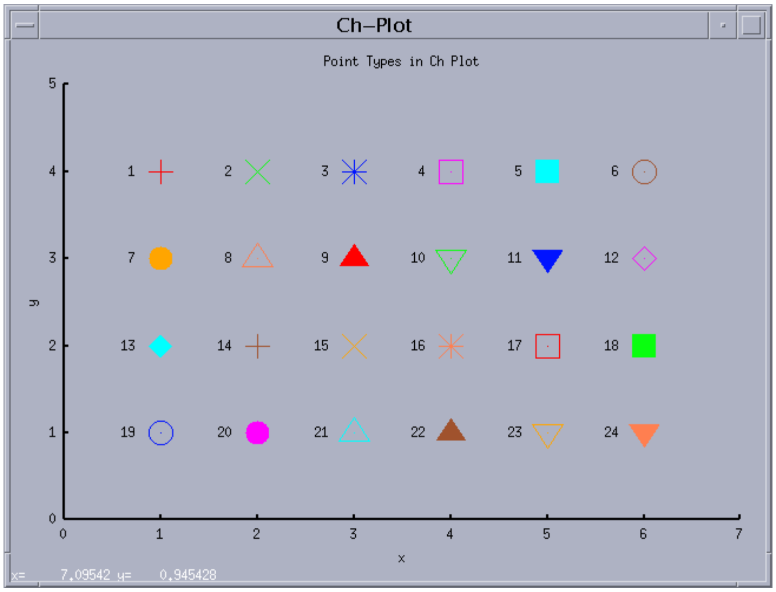
```
/* File: plotType4.cpp */
#include <chplot.h>

int main() {
    double x, y;
    char text[10];
    int datasetnum=0, point_type = 1, point_size = 5;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_X, 0, 7, 1);
    plot.axisRange(PLOT_AXIS_Y, 0, 5, 1);
    plot.title("Point Types in Ch Plot");
    for (y = 4; y >= 1; y--) {
        for (x = 1; x <= 6; x++) {
            sprintf(text, "%d", point_type);
            plot.point(x, y, 0);
            plot.plotType(PLOT_PLOTTYPE_POINTS, datasetnum);
            plot.pointType(datasetnum,  point_type, point_size);
            plot.text(text, PLOT_TEXT_RIGHT, x-.25, y, 0);
            datasetnum++; point_type++;
        }
    }
    plot.plotting();
    return 0;
}
```

Output

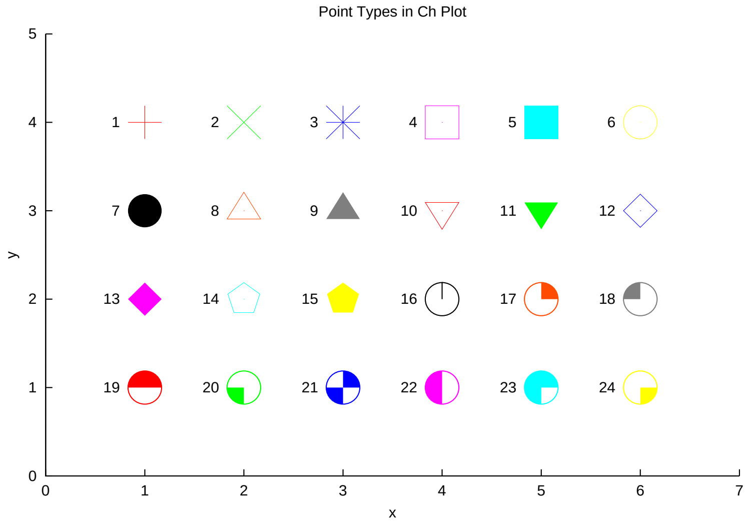
Output displayed in Window.



Output
Output displayed in X-window.

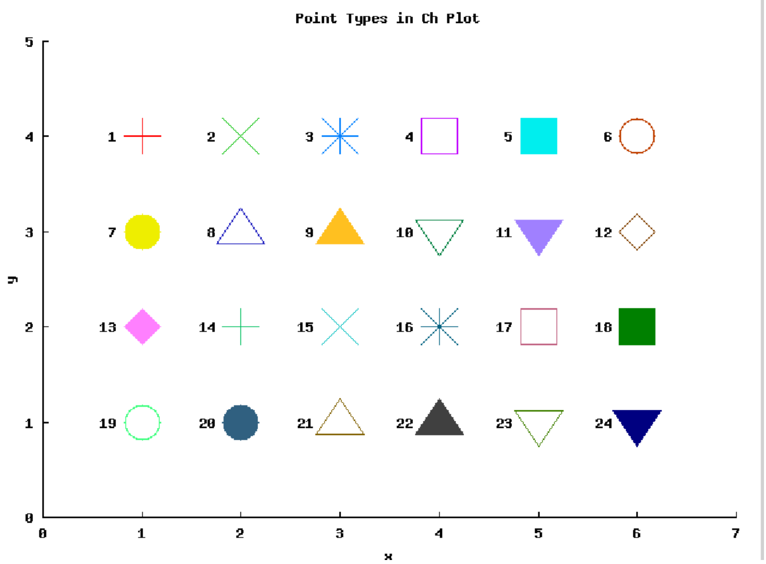


Output
Output as a postscript file.



Output

Output as a PNG file.

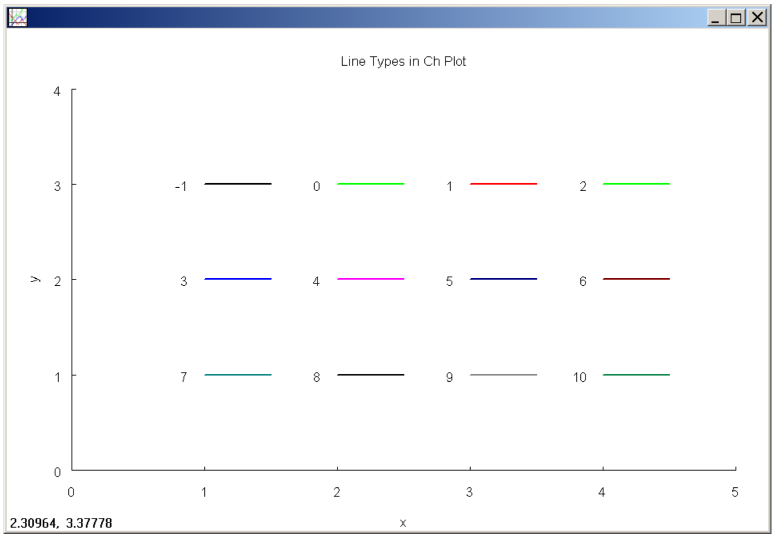


Example 2

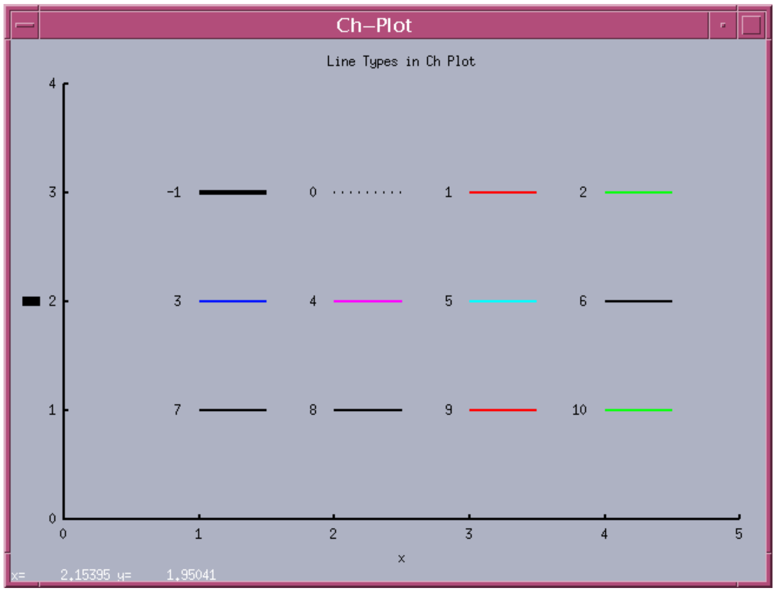
This example shows some of the different line types for the `postscript` terminal type (see **CPlot**::**outputType**). The appearance of lines for different terminal types may be different.

```
/* File: plotType6.cpp */
#include <chplot.h>

int main() {
    double x, y, xx[2], yy[2];
    char text[10];
    int line_type = -1, line_width = 2, datasetnum = 0;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_X, 0, 5, 1);
    plot.axisRange(PLOT_AXIS_Y, 0, 4, 1);
    plot.title("Line Types in Ch Plot");
    for (y = 3; y >= 1; y--) {
        for (x = 1; x <= 4; x++) {
            sprintf(text, "%d", line_type);
            xx[0] = x; xx[1] = x+0.5; // linspace(xx, x, x+.5);
            yy[0] = y; yy[1] = y; // linspace(yy, y, y);
            plot.data2DCurve(xx, yy, 2);
            plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
            plot.lineType(datasetnum, line_type, line_width);
            plot.text(text, PLOT_TEXT_RIGHT, x-.125, y, 0);
            datasetnum++;
            line_type++;
         }
    }
    plot.plotting();
    return 0;
}
```

Output
Output displayed in Window.

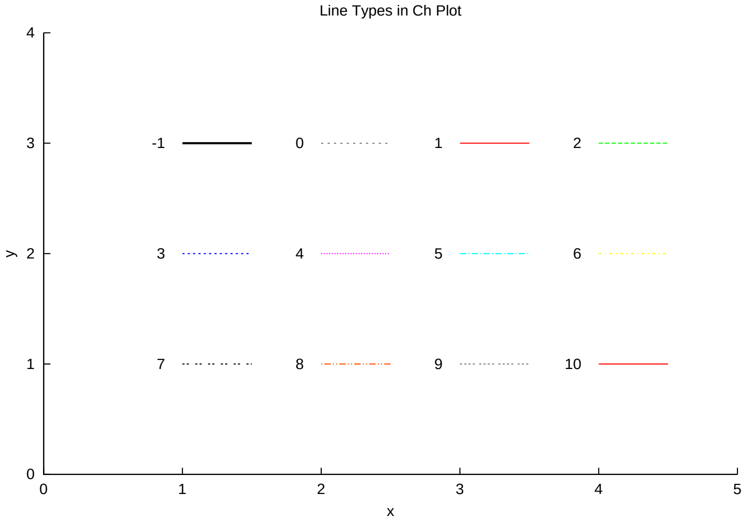


Output
Output displayed in X-window.



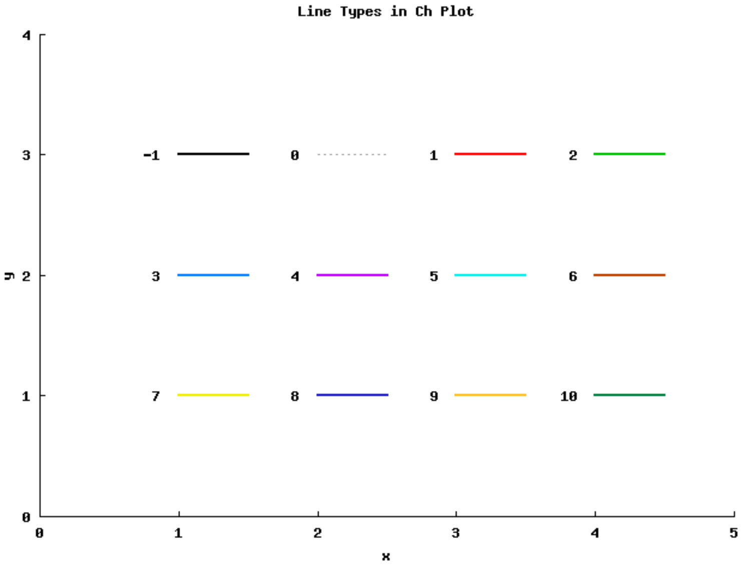
Output

Output as a postscript file.



Output

Output as a PNG file.

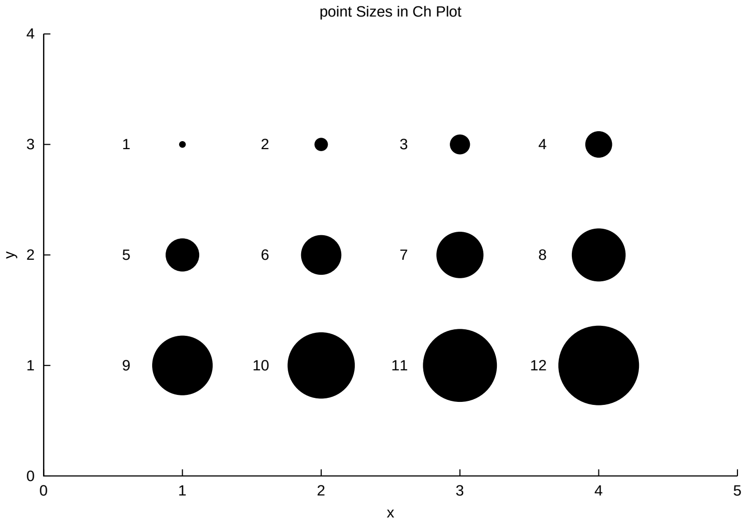


Example 3

This example shows some of the different point sizes for the `postscript` terminal type (see **CPlot::outputType**). The appearance of points for different terminal types may be different.

```
#include <chplot.h>

int main() {
    double x, y;
    char text[10];
    int datasetnum=0, point_type = 7, point_size = 1;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_X, 0, 5, 1);
    plot.axisRange(PLOT_AXIS_Y, 0, 4, 1);
    plot.title("point Sizes in Ch Plot");
    for (y = 3; y >= 1; y--) {
        for (x = 1; x <= 4; x++) {
            sprintf(text, "%d", point_size);
            plot.point(x, y, 0);
            plot.plotType(PLOT_PLOTTYPE_POINTS, datasetnum);
            plot.pointType(datasetnum,  point_type, point_size);
            plot.text(text, PLOT_TEXT_RIGHT, x-.375, y, 0);
            datasetnum++; point_size++;
        }
    }
    plot.plotting();
    return 0;
}
```

Output

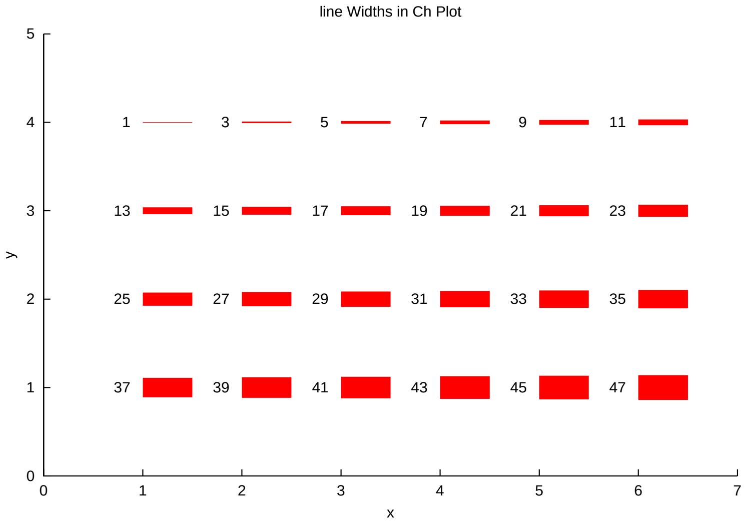


Example 4

This example shows some of the different line sizes for the `postscript` terminal type (see **CPlot::outputType**). The appearance of lines for different terminal types may be different.

```
#include <chplot.h>

int main() {
    double x, y, xx[2], yy[2];
    char text[10];
    int line_type = 1, line_width = 1, datasetnum = 0;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_X, 0, 7, 1);
    plot.axisRange(PLOT_AXIS_Y, 0, 5, 1);
    plot.title("line Widths in Ch Plot");
    for (y = 4; y >= 1; y--) {
        for (x = 1; x <= 6; x++) {
            sprintf(text, "%d", line_width);
            xx[0] = x; xx[1] = x+0.5; // linspace(xx, x, x+.5);
            yy[0] = y; yy[1] = y; //  linspace(yy, y, y);
            plot.data2DCurve(xx, yy, 2);
            plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
            plot.lineType(datasetnum, line_type, line_width);
            plot.text(text, PLOT_TEXT_RIGHT, x-.125, y, 0);
            datasetnum++;
            line_width+=2;
         }
    }
    plot.plotting();
    return 0;
}
```

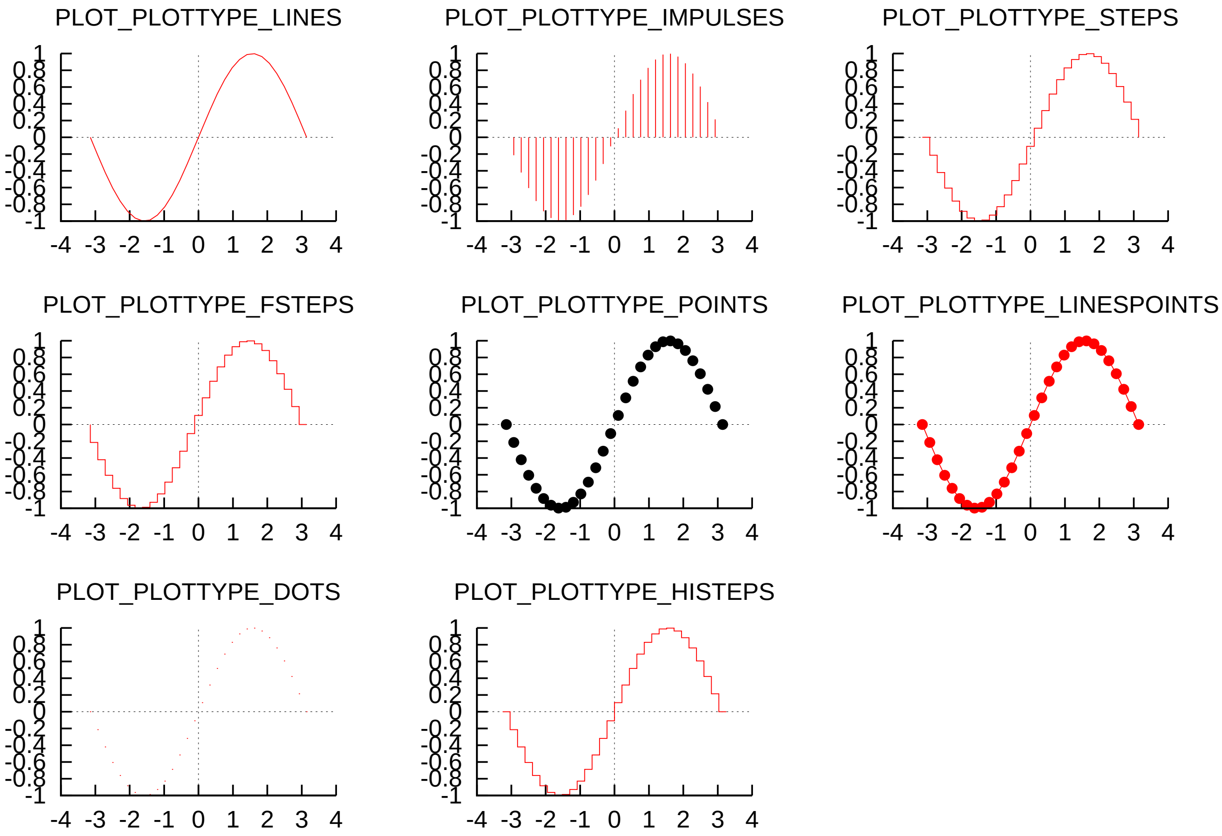
Output



Example 5

```
#include <chplot.h>
#include <math.h>

#define NUM 30
int main() {
    int i;
    class CPlot subplot, *spl;
    double x1[NUM], y1[NUM];
    int line_type = 1, line_width = 1;
    int point_type = 7, point_size = 1;

    for(i=0; i<NUM; i++) {
      x1[i] = -M_PI + i*2*M_PI/(NUM-1);  // linspace(x1, -PI, PI)
      y1[i] = sin(x1[i]);
    }
    subplot.subplot(3,3);
    spl = subplot.getSubplot(0,0);
    spl->data2DCurve(x1, y1, NUM);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_LINES, 0);
    spl->lineType(0, line_type, line_width);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_LINES");
    spl = subplot.getSubplot(0,1);
    spl->data2DCurve(x1, y1, NUM);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_IMPULSES, 0);
    spl->lineType(0, line_type, line_width);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_IMPULSES");
    spl = subplot.getSubplot(0,2);
    spl->data2DCurve(x1, y1, NUM);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_STEPS, 0);
    spl->lineType(0, line_type, line_width);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_STEPS");
    spl = subplot.getSubplot(1,0);
    spl->data2DCurve(x1, y1, NUM);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_FSTEPS, 0);
    spl->lineType(0, line_type, line_width);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_FSTEPS");
    spl = subplot.getSubplot(1,1);
    spl->data2DCurve(x1, y1, NUM);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_POINTS, 0);
    spl->pointType(0, point_type, point_size);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_POINTS");
    spl = subplot.getSubplot(1,2);
    spl->data2DCurve(x1, y1, NUM);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_LINESPOINTS, 0);
    spl->lineType(0, line_type, line_width);
    spl->pointType(0, point_type, point_size);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_LINESPOINTS");
    spl = subplot.getSubplot(2,0);
    spl->data2DCurve(x1, y1, NUM);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_DOTS, 0, point_type);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_DOTS");
    spl = subplot.getSubplot(2,1);
    spl->data2DCurve(x1, y1, NUM);
    spl->axisRange(PLOT_AXIS_Y, -1, 1);
    spl->ticsRange(PLOT_AXIS_Y, 0.5, -1, 1);
    spl->plotType(PLOT_PLOTTYPE_HISTEPS, 0);
    spl->lineType(0, line_type, line_width);
    spl->label(PLOT_AXIS_XY, NULL);
    spl->title("PLOT_PLOTTYPE_HISTEPS");
    subplot.plotting();
    return 0;
}
```

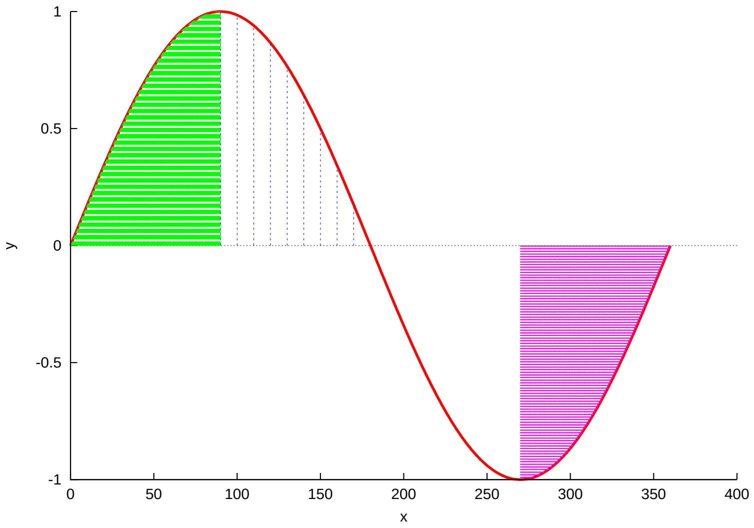
Output



Example 6

```
#include <math.h>
#include <chplot.h>

#define NUM1 360
#define NUM2 10
int main() {
    int i;
    double x0[NUM1], y0[NUM1];
    double x1[NUM1], y1[NUM1];
    double x2[NUM2], y2[NUM2];
    double x3[NUM1], y3[NUM1];
    int line_type = 1, line_width = 5;
    class CPlot plot;

    for(i=0; i<NUM1; i++) {
       x0[i]= 0 + i*360.0/(NUM1-1); // linspace(x0, 0, 360);
       y0[i] = sin(x0[i]*M_PI/180); // y0 = sin(x0*M_PI/180);
       x1[i]= 0 + i*90.0/(NUM1-1); // linspace(x1, 0, 90);
       y1[i] = sin(x1[i]*M_PI/180); // y1 = sin(x1*M_PI/180);
       x3[i]= 270 + i*90.0/(NUM1-1); // linspace(x3, 270, 360);
       y3[i] = sin(x3[i]*M_PI/180); // y3 = sin(x3*M_PI/180);
    }
    for(i=0; i<NUM2; i++) {
       x2[i]= 90 + i*90.0/(NUM2-1); // linspace(x2, 90, 180);
       y2[i] = sin(x2[i]*M_PI/180); // y2 = sin(x2*M_PI/180);
    }
    plot.data2DCurve(x0, y0, NUM1);
    plot.data2DCurve(x1, y1, NUM1);
    plot.data2DCurve(x2, y2, NUM2);
    plot.data2DCurve(x3, y3, NUM1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);       // line for (x,y)
    plot.lineType(0, line_type, line_width);
    line_width = 1;
    // impulse plot for (x1, y1)
    plot.plotType(PLOT_PLOTTYPE_IMPULSES, 1);
    plot.lineType(1, line_type, line_width);
    // impulse plot for (x2, y2)
    plot.plotType(PLOT_PLOTTYPE_IMPULSES, 2);
    plot.lineType(2, line_type, line_width);
   // impulse plot for (x3, y3)
    plot.plotType(PLOT_PLOTTYPE_IMPULSES, 3);
    plot.lineType(3, line_type, line_width);
    plot.plotting();                   // get the plotting job done
    return 0;
}
```

Output



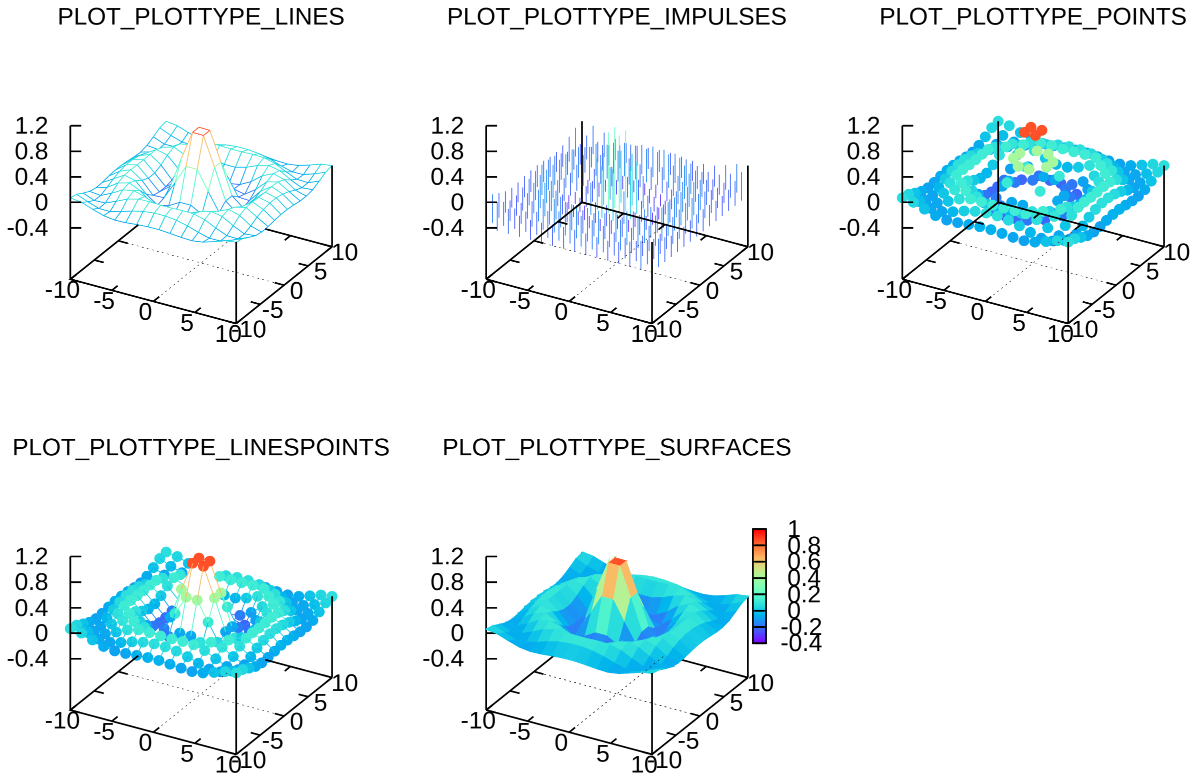
Example 7

```
#include <chplot.h>
#include <math.h>


#define NUMX 16
#define NUMY 16
int main() {
    int line_type = 1, line_width = 1;
    int point_type = 7, point_size = 1;
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double r;
    int i, j;
    class CPlot subplot, *spl;

    for(i=0; i<NUMX; i++) {
      x[i] = -10 + i*20.0/(NUMX-1);  // linspace(x, -10, 10)
    }
```

```
for(i=0; i<NUMY; i++) {
  y[i] = -10 + i*20.0/(NUMY-1);  // linspace(y, -10, 10)
}
for(i=0; i<NUMX; i++) {
    for(j=0; j<NUMY; j++) {
        r = sqrt(x[i]*x[i]+y[j]*y[j]);
        z[NUMY*i+j] = sin(r)/r;
    }
}
subplot.subplot(2,3);
spl = subplot.getSubplot(0,0);
spl->data3DSurface(x, y, z, NUMX, NUMY);
spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
spl->axisRange(PLOT_AXIS_XY, -10, 10);
spl->ticsRange(PLOT_AXIS_XY, 5, -10, 10);
spl->plotType(PLOT_PLOTTYPE_LINES, 0);
spl->lineType(0, line_type, line_width);
spl->colorBox(PLOT_OFF);
spl->label(PLOT_AXIS_XYZ, NULL);
spl->title("PLOT_PLOTTYPE_LINES");
spl = subplot.getSubplot(0,1);
spl->data3DSurface(x, y, z, NUMX, NUMY);
spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
spl->axisRange(PLOT_AXIS_XY, -10, 10);
spl->ticsRange(PLOT_AXIS_XY, 5, -10, 10);
spl->plotType(PLOT_PLOTTYPE_IMPULSES, 0);
spl->lineType(0, line_type , line_width);
spl->colorBox(PLOT_OFF);
spl->label(PLOT_AXIS_XYZ, NULL);
spl->title("PLOT_PLOTTYPE_IMPULSES");
spl = subplot.getSubplot(0,2);
spl->data3DSurface(x, y, z, NUMX, NUMY);
spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
spl->axisRange(PLOT_AXIS_XY, -10, 10);
spl->ticsRange(PLOT_AXIS_XY, 5, -10, 10);
spl->plotType(PLOT_PLOTTYPE_POINTS, 0);
spl->pointType(0, point_type, point_size);
spl->colorBox(PLOT_OFF);
spl->label(PLOT_AXIS_XYZ, NULL);
spl->title("PLOT_PLOTTYPE_POINTS");
spl = subplot.getSubplot(1,0);
spl->data3DSurface(x, y, z, NUMX, NUMY);
spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
spl->axisRange(PLOT_AXIS_XY, -10, 10);
spl->ticsRange(PLOT_AXIS_XY, 5, -10, 10);
spl->plotType(PLOT_PLOTTYPE_LINESPOINTS, 0);
spl->lineType(0, line_type, line_width);
spl->pointType(0, point_type, point_size);
spl->colorBox(PLOT_OFF);
spl->label(PLOT_AXIS_XYZ, NULL);
spl->title("PLOT_PLOTTYPE_LINESPOINTS");
spl = subplot.getSubplot(1,1);
spl->data3DSurface(x, y, z, NUMX, NUMY);
spl->axisRange(PLOT_AXIS_Z, -.4, 1.2);
spl->ticsRange(PLOT_AXIS_Z, .4, -.4, 1.2);
```

```
    spl->axisRange(PLOT_AXIS_XY, -10, 10);
    spl->ticsRange(PLOT_AXIS_XY, 5, -10, 10);
    spl->plotType(PLOT_PLOTTYPE_SURFACES, 0);
    spl->label(PLOT_AXIS_XYZ, NULL);
    spl->title("PLOT_PLOTTYPE_SURFACES");
    subplot.plotting();
    return 0;
}
```

Output



PLOT_PLOTTYPE_LINES

PLOT_PLOTTYPE_IMPULSES

PLOT_PLOTTYPE_POINTS

PLOT_PLOTTYPE_LINESPOINTS

PLOT_PLOTTYPE_SURFACES

Example 8

This example illustrates the plot type **PLOT_PLOTTYPE_VECTORS**. The arrow style is defined based on the specification in **CPlot::arrow**() on page 55.

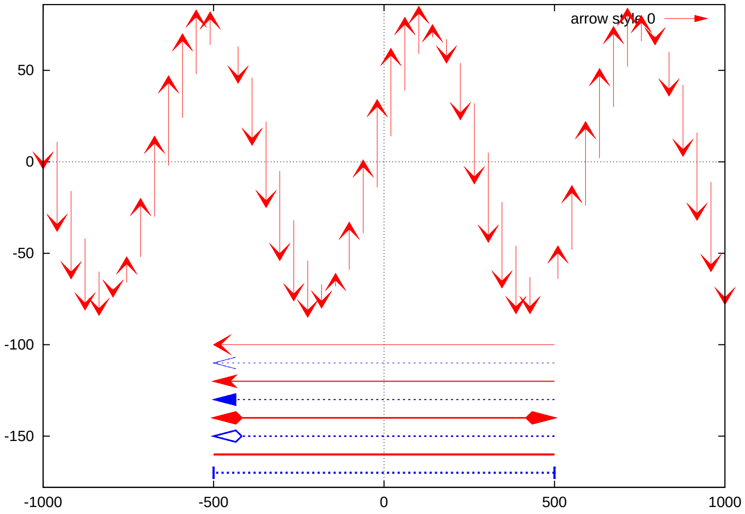
```
#include <chplot.h>

char *arrowstyle[] = {
             "head filled size screen 0.025,30,45 linetype 1 linewidth 1",
             "head nofilled size screen 0.03,15 linetype 3 linewidth 1",
             "head filled size screen 0.03,15,45 linetype 1 linewidth 2",
             "head filled size screen 0.03,15 linetype 3 linewidth 2",
             "heads filled size screen 0.03,15,135 linetype 1 linewidth 3",
             "head empty size screen 0.03,15,135 linetype 3 linewidth 3",
             "nohead linetype 1 linewidth 4",
             "heads size screen 0.008,90 linetype 3 linewidth 4"
};
```

```
int main () {
    class CPlot plot;
    int i;

    plot.label(PLOT_AXIS_X, "");
    plot.label(PLOT_AXIS_Y, "");
    plot.axisRange(PLOT_AXIS_X, -1000, 1000);
    plot.axisRange(PLOT_AXIS_Y,-178, 86);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);

    for(i=0; i<sizeof(arrowstyle)/sizeof(char*); i++) {
      plot.arrow(-500,-100-10*i,0, 500,-100-10*i,0, arrowstyle[i]);
    }
    plot.dataFile("arrowstyle.dat", "using 1:2:(0):3");
    plot.plotType(PLOT_PLOTTYPE_VECTORS, 0, arrowstyle[0]);
    plot.legend("arrow style 0", 0);
    plot.plotting();
}
```

The contents of data file `arrowstyle.dat` in Example 8

```
-1000   37      -41
-959    11      -49
-918    -16     -48
...
```

Output



Example 9

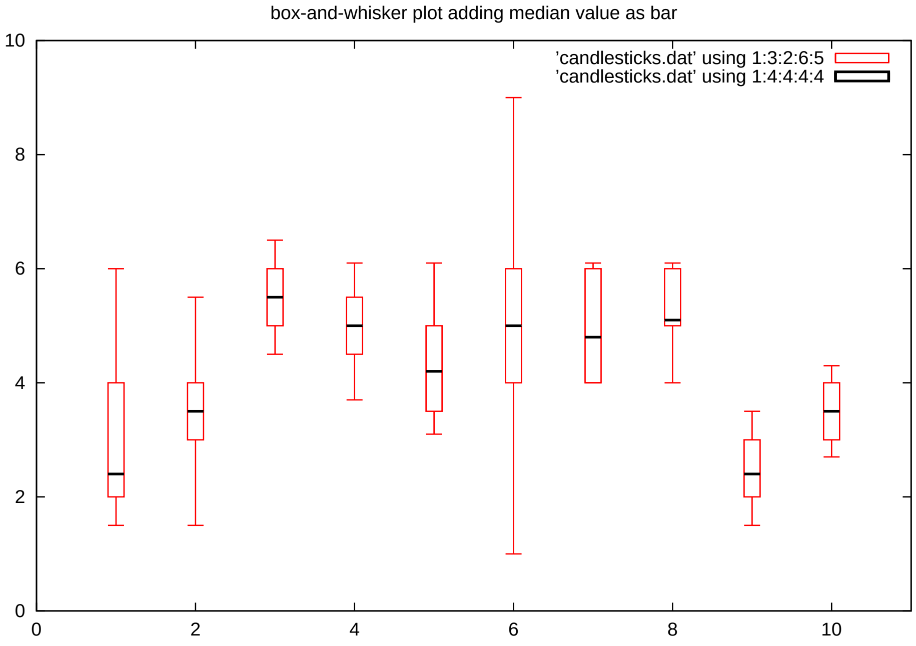
This example illustrates the plot type **PLOT_PLOTTYPE_CANDLESTICKS**.

```
/* File: plotTtype_cs.ch to process data in candlesticks.dat */
#include <chplot.h>

int main() {
    class CPlot plot;

    plot.label(PLOT_AXIS_X, "");
    plot.label(PLOT_AXIS_Y, "");
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot.title("box-and-whisker plot adding median value as bar");
    plot.axisRange(PLOT_AXIS_X, 0, 11);
    plot.axisRange(PLOT_AXIS_Y, 0, 10);
    plot.dataFile("candlesticks.dat", "using 1:3:2:6:5");
    plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 0, "linetype 1 linewidth 2 whiskerbars");
    plot.boxFill(0, PLOT_BOXFILL_EMPTY);
    plot.legend("'candlesticks.dat' using 1:3:2:6:5", 0);
    plot.dataFile("candlesticks.dat", "using 1:4:4:4:4");
    plot.plotType(PLOT_PLOTTYPE_CANDLESTICKS, 1, "linetype -1 linewidth 2");
    plot.legend("'candlesticks.dat' using 1:4:4:4:4", 1);
    plot.boxWidth(0.2);
    plot.plotting();
    return 0;
}
```

The contents of data file `candlesticks.dat` in Example 9

```
    1        1.5      2        2.4      4        6.
    2        1.5      3        3.5      4        5.5
    3        4.5      5        5.5      6        6.5
    ...
```

Output



Example 10

This example illustrates the plot type **PLOT_PLOTTYPE_FINANCEBARS**.

```
/* File: plotType_fb.ch to process data in finance.dat

  data and indicators in finance.dat
  # data file layout:
```

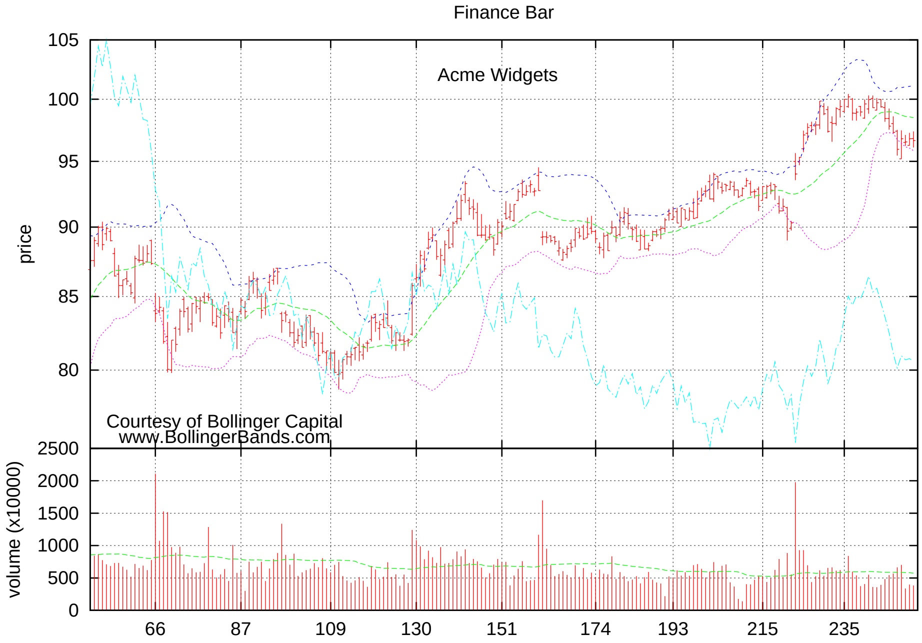
```
  # date, open, high, low, close, volume,
  # 50-day moving average volume, Intraday Intensity,
  # 20-day moving average close,
  # upper Bollinger Band, lower Bollinger Band
*/

#include <chplot.h>

int main() {
    class CPlot subplot, *plot1, *plot2;

    subplot.subplot(2,1);
    subplot.title("Finance Bar");
    plot1 = subplot.getSubplot(0,0);
    plot1->label(PLOT_AXIS_X, "");
    plot1->label(PLOT_AXIS_Y, "price");
    plot1->border(PLOT_BORDER_ALL, PLOT_ON);
    plot1->ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot1->dataFile("finance.dat", "using 0:2:3:4:5");
    plot1->plotType(PLOT_PLOTTYPE_FINANCEBARS, 0);
    plot1->axisRange(PLOT_AXIS_X, 50, 253);
    plot1->axisRange(PLOT_AXIS_Y, 75, 105);
    plot1->grid(PLOT_ON);
    //plot1->ticsPosition(PLOT_AXIS_X, 66, 87, 109, 130, 151, 174, 193, 215, 235);
    //plot1->ticsPosition(PLOT_AXIS_Y, 105, 100, 95, 90, 85, 80);
    /* use the code below for X-label for C++ or Ch */
      plot1->ticsLabel(PLOT_AXIS_X, "6/03", 66);
      plot1->ticsLabel(PLOT_AXIS_X, "7/03", 87);
      plot1->ticsLabel(PLOT_AXIS_X, "8/03", 109);
      plot1->ticsLabel(PLOT_AXIS_X, "9/03", 130);
      plot1->ticsLabel(PLOT_AXIS_X, "10/03", 151);
      plot1->ticsLabel(PLOT_AXIS_X, "11/03", 174);
      plot1->ticsLabel(PLOT_AXIS_X, "12/03", 193);
      plot1->ticsLabel(PLOT_AXIS_X, "1/04", 215);
      plot1->ticsLabel(PLOT_AXIS_X, "2/04", 235);
      plot1->ticsPosition(PLOT_AXIS_Y, 105);
      plot1->ticsPosition(PLOT_AXIS_Y, 100);
      plot1->ticsPosition(PLOT_AXIS_Y, 95);
      plot1->ticsPosition(PLOT_AXIS_Y, 90);
      plot1->ticsPosition(PLOT_AXIS_Y, 85);
      plot1->ticsPosition(PLOT_AXIS_Y, 80);
    plot1->ticsFormat(PLOT_AXIS_X, "");
    plot1->scaleType(PLOT_AXIS_Y, PLOT_SCALETYPE_LOG);
    plot1->dataFile("finance.dat", "using 0:9");
    plot1->dataFile("finance.dat", "using 0:10");
    plot1->dataFile("finance.dat", "using 0:11");
    plot1->dataFile("finance.dat", "using 0:8");
    plot1->axes(plot1->dataSetNum(), "x1y2");
    plot1->text("Courtesy of Bollinger Capital", PLOT_TEXT_CENTER, 83, 76.7, 0);
    plot1->text("www.BollingerBands.com", PLOT_TEXT_CENTER, 83, 75.7, 0);
    plot1->size(1, 0.7);           // 70% for the top plot
    plot1->margins(9, -1, -1, 0);  // left margin is 9, bottom margin is 0
    plot1->origin(0, 0.3);         // origin for Y is 0.3
    plot1->text("Acme Widgets", PLOT_TEXT_CENTER, 150, 102, 0);

    plot2 = subplot.getSubplot(1,0);
    plot2->label(PLOT_AXIS_X, "");
    plot2->label(PLOT_AXIS_Y, "volume (x10000)");
    plot2->dataFile("finance.dat", "using 0:($6/10000)");
    plot2->plotType(PLOT_PLOTTYPE_IMPULSES, 0);
    plot2->dataFile("finance.dat", "using 0:($7/10000)");
    plot2->border(PLOT_BORDER_ALL, PLOT_ON);
    plot2->ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot2->grid(PLOT_ON);
    plot2->axisRange(PLOT_AXIS_X, 50, 253);
    plot2->ticsRange(PLOT_AXIS_Y, 500);
    plot2->ticsFormat(PLOT_AXIS_Y, "%1.0f");
    //plot2->ticsPosition(PLOT_AXIS_X, 66, 87, 109, 130, 151, 174, 193, 215, 235);
    /* use the code below for X-label for C++ or Ch */
      plot2->ticsLabel(PLOT_AXIS_X, "6/03", 66);
      plot2->ticsLabel(PLOT_AXIS_X, "7/03", 87);
      plot2->ticsLabel(PLOT_AXIS_X, "8/03", 109);
      plot2->ticsLabel(PLOT_AXIS_X, "9/03", 130);
      plot2->ticsLabel(PLOT_AXIS_X, "10/03", 151);
      plot2->ticsLabel(PLOT_AXIS_X, "11/03", 174);
      plot2->ticsLabel(PLOT_AXIS_X, "12/03", 193);
      plot2->ticsLabel(PLOT_AXIS_X, "1/04", 215);
      plot2->ticsLabel(PLOT_AXIS_X, "2/04", 235);
    plot2->size(1, 0.3);          // 30% for the bottom plot
    plot2->margins(9, -1, 0, -1); // left margin is 9, top margin is 0
    subplot.plotting();
    return 0;
}
```

The contents of data file `finance.dat` in Example 10

```
# data and indicators in finance.dat
# data file layout:
# date, open, high, low, close, volume,
# 50-day moving average volume, Intraday Intensity,
# 20-day moving average close,
# upper Bollinger Band, lower Bollinger Band
2/27/2003 77.9  78.59 76.75 77.28 9927900 0 -4208566 0 0 0
2/28/2003 78.15 78.47 77    77.95 6556100 0 -2290796 0 0 0
3/3/2003  78.6  79    77.12 77.33 6618300 0 -7430539 0 0 0
...
```

This data file is also used to plot stock prices in Example 14 below.

Output



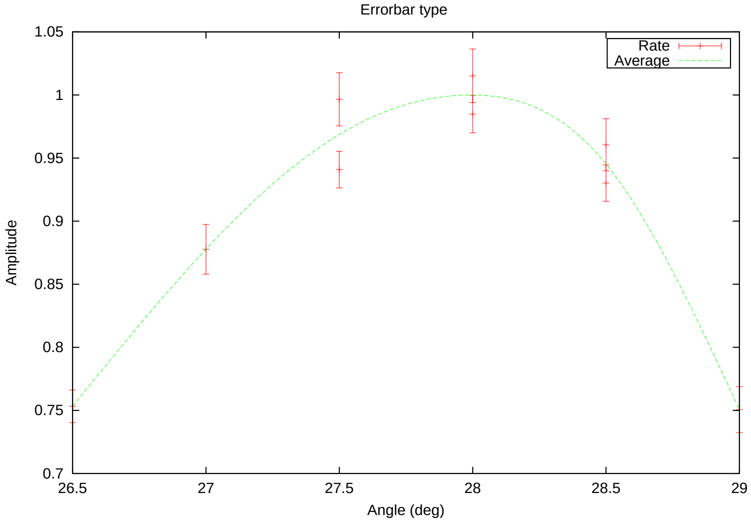
Example 11
This example illustrates the plot type **PLOT_PLOTTYPE_YERRORBARS**.

```
#include<chplot.h>

int main() {
   CPlot plot;

   plot.title("Errorbar type ");
   plot.label(PLOT_AXIS_X, "Angle (deg)");
   plot.label(PLOT_AXIS_Y, "Amplitude");
   plot.border(PLOT_BORDER_ALL, PLOT_ON);
   plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
   plot.legendOption("box");
   plot.dataFile("big_peak.dat");
   plot.plotType(PLOT_PLOTTYPE_YERRORBARS, 0);
   plot.legend("Rate", 0);
   plot.dataFile("big_peak.dat");
   plot.smooth(1, "csplines");
   plot.legend("Average", 1);
   plot.barSize(1.0);
   plot.plotting();
}
```

The contents of data file `big_peak.dat` **in Example 11**

```
26.500000 0.753252 0.012953
27.000000 0.877710 0.019712
27.500000 0.996531 0.021018
...
```

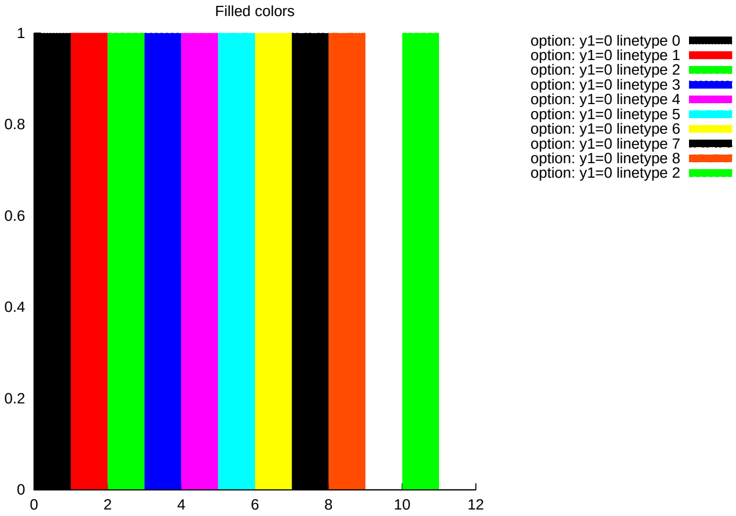
Output



Example 12
This example illustrates the plot type **PLOT_PLOTTYPE_FILLEDCURVES**.

```
#include<math.h>
#include<chplot.h>

#define N 100

double func(double x) {
   double y;

   y = 1;
   return y;
}

int main() {
   CPlot plot;
   double x0, xf;
   int i;
   char option[64], legend[64];

   plot.title("Filled colors");
   plot.legendOption("outside");
   plot.label(PLOT_AXIS_X, "");
   plot.label(PLOT_AXIS_Y, "");
   plot.axisRange(PLOT_AXIS_X, 0, 12);
   plot.axisRange(PLOT_AXIS_Y, 0, 1);

   for(i=0; i<9; i++) {
     x0 = i;
     xf = i+1;
     sprintf(option, "y1=0 linetype %d", i);
     sprintf(legend, "option: %s", option);
     plot.func2D(x0, xf, N, func);
     plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, i, option);
     plot.legend(legend, i);
   }
   x0 = i+1;
   xf = i+2;
   plot.func2D(x0, xf, N, func);
   plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, 9, "y1=0 linetype 2");
   plot.legend("option: y1=0 linetype 2", 9);
   plot.plotting();
}
```

Output

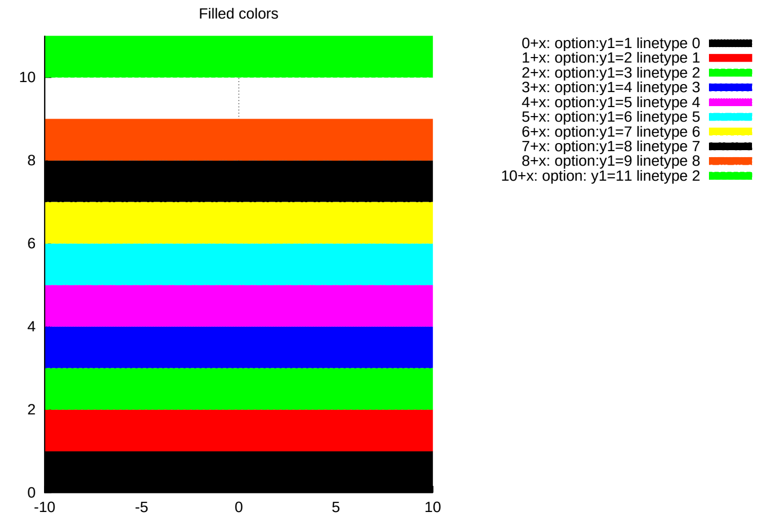


Example 13

This example illustrates the plot type **PLOT_PLOTTYPE_FILLEDCURVES**.

```
#include<math.h>
#include<chplot.h>

#define N 100

double func(double x, void *param) {
   double offset;
   double y;

   offset = *(double*)param;
   y = offset;
   return y;
}


int main() {
   CPlot plot;
   double offset, x0, xf;
   int i;
   char option[64], legend[64];

   x0 = -10;
   xf = 10;

   plot.title("Filled colors");
   plot.legendOption("outside");
   plot.label(PLOT_AXIS_X, "");
   plot.label(PLOT_AXIS_Y, "");
   plot.axisRange(PLOT_AXIS_X, -10, 10);
   plot.axisRange(PLOT_AXIS_Y, 0, 11);
   for(i=0; i<9; i++) {
     offset = i;
     sprintf(option, "y1=%d linetype %d", i+1, i);
     sprintf(legend, "%d+x: option:%s", i, option);
     plot.funcp2D(x0, xf, N, func, &offset);
     plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, i, option);
     plot.legend(legend, i);
   }
   offset = 10;
   plot.funcp2D(x0, xf, N, func, &offset);
   plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, 9, "y1=11 linetype 2");
   plot.legend("10+x: option: y1=11 linetype 2", 9);
   plot.plotting();

}
```

Output

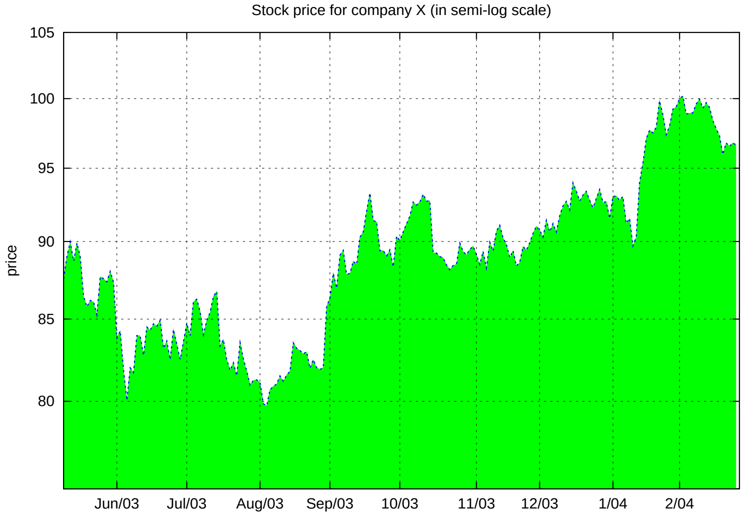


Example 14

This example illustrates the plot type **PLOT_PLOTTYPE_FILLEDCURVES**. This plot for stock prices uses the same data file in Example 10.

```
/* File: finance.ch to process data in finance.dat

  data and indicators in finance.dat
  # data file layout:
  # date, open, high, low, close, volume,
  # 50-day moving average volume, Intraday Intensity,
  # 20-day moving average close,
  # upper Bollinger Band, lower Bollinger Band
*/

#include <chplot.h>

int main() {
    class CPlot plot;
    int line_type, line_width;

    plot.title("Stock price for company X (in semi-log scale)");
    plot.label(PLOT_AXIS_X, "");
    plot.label(PLOT_AXIS_Y, "price");
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot.dataFile("finance.dat", "using 0:5");
    plot.plotType(PLOT_PLOTTYPE_FILLEDCURVES, 0, "y1=0 linetype 2");
    plot.dataFile("finance.dat", "using 0:5");
    line_type = 3;
    line_width = 2;
    plot.plotType(PLOT_PLOTTYPE_LINES, 1);
    plot.lineType(1, line_type, line_width);
    plot.axisRange(PLOT_AXIS_X, 50, 253);
    plot.axisRange(PLOT_AXIS_Y, 75, 105);
    plot.scaleType(PLOT_AXIS_Y, PLOT_SCALETYPE_LOG);
    plot.grid(PLOT_ON, "front linewidth 2");
    //plot.ticsPosition(PLOT_AXIS_Y, 105, 100, 95, 90, 85, 80);
      plot.ticsPosition(PLOT_AXIS_Y, 105);
      plot.ticsPosition(PLOT_AXIS_Y, 100);
      plot.ticsPosition(PLOT_AXIS_Y, 95);
      plot.ticsPosition(PLOT_AXIS_Y, 90);
      plot.ticsPosition(PLOT_AXIS_Y, 85);
      plot.ticsPosition(PLOT_AXIS_Y, 80);
    //plot.ticsLabel(PLOT_AXIS_X, "Jun/03", 66, "Jul/03", 87, "Aug/03", 109, "Sep/03", 130,
    //              "10/03", 151, "11/03", 174, "12/03", 193, "1/04", 215, "2/04", 235);
;
   /* use the code below for X-label for C++ or Ch */
      plot.ticsLabel(PLOT_AXIS_X, "6/03", 66);
      plot.ticsLabel(PLOT_AXIS_X, "7/03", 87);
      plot.ticsLabel(PLOT_AXIS_X, "8/03", 109);
      plot.ticsLabel(PLOT_AXIS_X, "9/03", 130);
      plot.ticsLabel(PLOT_AXIS_X, "10/03", 151);
      plot.ticsLabel(PLOT_AXIS_X, "11/03", 174);
      plot.ticsLabel(PLOT_AXIS_X, "12/03", 193);
      plot.ticsLabel(PLOT_AXIS_X, "1/04", 215);
      plot.ticsLabel(PLOT_AXIS_X, "2/04", 235);
    plot.plotting();
}
```

Output



Example 15
The example illustrates the plot type **PLOT_PLOTTYPE_BOXES** can be found on page 74.

See Also
CPlot::**arrow**(), **CPlot**::**lineType**(), **CPlot**::**pointType**().

CPlot::plotting

Synopsis
#include <**chplot.h**>
void plotting();

Purpose
Generate a plot file or display a plot.

Return Value
None.

Parameters
None.

Description
The plot is displayed or a file is generated containing the plot when this function is called. It shall be called after all the desired plot options are set.

Example
See **CPlot**::**data2D**().

CPlot::point

Synopsis in Ch
#include <**chplot.h**>
int point(**double** *x*, **double** *y*, ... /* **double** *z* */);

Synopsis in C++
#include <**chplot.h**>
int point(**double** *x*, **double** *y*);
int point(**double** *x*, **double** *y*, **double** *z*);

Syntax in Ch and C++
point(*x*, *y*);
point(*x*, *y*, *z*);

Purpose
Add a point to a 2D or 3D plot.

Return Value

This function returns 0 on success and -1 on failure.

Parameters

x The x coordinate of the point.

y The y coordinate of the point.

z The z coordinate of the point. This argument is ignored for 2D plots.

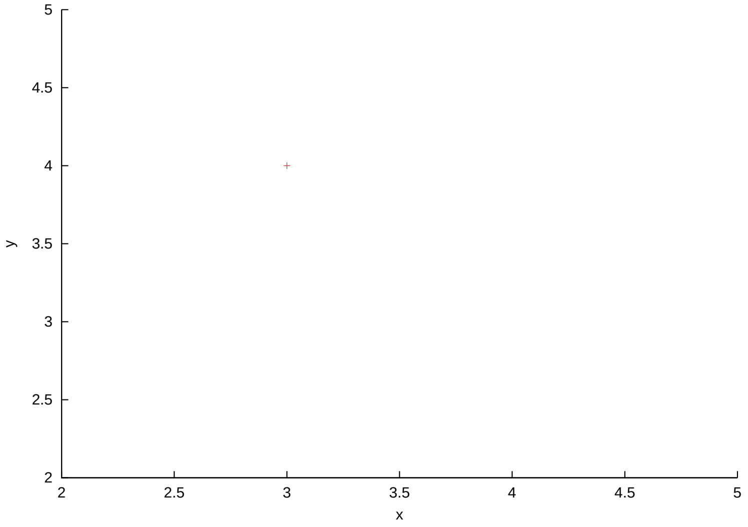
Description

This function adds a point to a plot. It is a convenience function for creation of geometric primitives. A point added with this function counts as a data set for later calls to **CPlot**::**legend**() and **CPlot**::**plotType**(). For 2D rectangular and 3D cartesian plots, *x*, *y*, and *z* are the coordinates of the point specified in units of the x, y and z axes. However, for 2D plots, *z* is ignored. For 2D polar and 3D cylindrical plots, the point is specified in polar coordinates where *x* is θ, *y* is r, and *z* is unchanged. Again, for 2D plots, *z* is ignored. For 3D plots with spherical coordinates *x* is θ, *y* is ϕ and *z* is r. For 3D plots with points, hidden line removal should be disabled (see **CPlot**::**removeHiddenLine**()) after all data are added.

Example 1

```
#include <chplot.h>

int main() {
    double x = 3, y = 4;
    class CPlot plot;

    plot.axisRange(PLOT_AXIS_XY, 2, 5, .5); /* one point cannot do autorange */
    plot.point(x, y);
    plot.plotting();
    return 0;
}
```

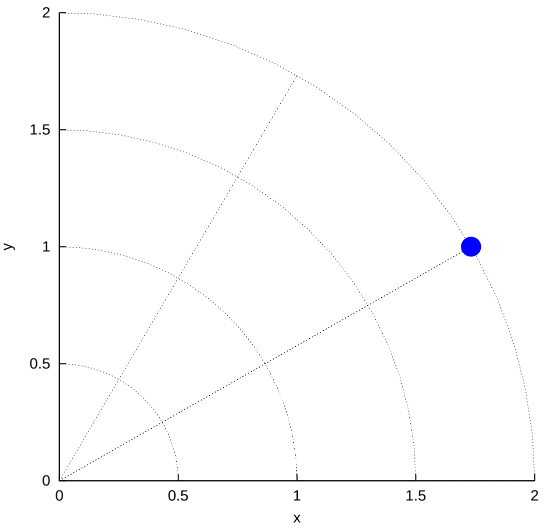
Output



Example 2

```
#include <chplot.h>

int main() {
    double theta1 = 30, r1 = 2;
    class CPlot plot;
    int point_type = 7, point_size = 3;
    char *point_color = "blue";

    plot.grid(PLOT_ON);
    plot.polarPlot(PLOT_ANGLE_DEG);
    plot.point(theta1, r1);
    plot.plotType(PLOT_PLOTTYPE_POINTS, 0);
    plot.pointType(0, point_type, point_size, point_color);
    plot.axisRange(PLOT_AXIS_XY, 0, 2, .5);
    plot.sizeRatio(-1);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::data2D(), **CPlot::data2DCurve**(), **CPlot::data3D**(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **CPlot::circle**(), **CPlot::line**(), **CPlot::outputType**(), **CPlot::plotType**(), **CPlot::polygon**(), **CPlot::rectangle**().

CPlot::pointType

Synopsis in Ch
#include <chplot.h>
void pointType(int *num*, **int** *point_type*, **int** *point_size*], ... /* [**char** *point_color*] */);

Synopsis in C++
#include <chplot.h>
void pointType(int *num*, **int** *point_or_point_type*, **int** *point_size*);

void pointType(int *num*, **int** *point_or_point_type*, **int** *point_size*, **char** *point_color*);

Syntax in Ch and C++
pointType(*num*, *point_type*, *point_size*)
pointType(*num*, *point_type*, *point_size*, *point_color*)

Purpose
Set the point type, size, and color for points, line-points, etc.

Return Value
None.

Parameters

num The data set to which the point type, size, and color apply.

point_type An integer index representing the desired point type.

point_size A scaling factor for the size of the point used. The point size is *point_size* multiplied by the default size.

point_color color for the point.

Description
Set the desired point type, size, and color for a previously added data set.

Numbering of the data sets starts with zero. The point style and/or marker type for the plot are selected automatically. The default point tye, size, and color can be changed by this member function.

The *point_type* specifies an index for the point type used for drawing the point. The point type varies depending on the terminal type used (see **CPlot**::**outputType**). The value *point_type* is used to change the appearance (color and/or marker type) of a point. It is specified with an integer representing the index of the desired point type. All terminals support at least six different point types. *point_size* is an optional argument used to change the size of the point. The point size is *point_size* multiplied by the default size. If *point_type* and *point_size* are set to zero or a negative number, a default value is used.

An optional fourth argument can specify the color of a point by a color name or RGB value, such as `"blue"` or `"#0000ff"` for color blue. The default point type, size, and color can be changed by the function call

```
plot.pointType(num, pointtype, pointsize, "blue");
```

The color of the point is specified as blue in this example. The valid color names and their corresponding GRB values are listed below.

```
  Color Name        Hexadecimal R   G    B
                                values
 ------------------------------------------
  white              #ffffff = 255 255 255
  black              #000000 =   0   0   0
  gray0              #000000 =   0   0   0
  grey0              #000000 =   0   0   0
  gray10             #1a1a1a =  26  26  26
  grey10             #1a1a1a =  26  26  26
  gray20             #333333 =  51  51  51
  grey20             #333333 =  51  51  51
  gray30             #4d4d4d =  77  77  77
  grey30             #4d4d4d =  77  77  77
  gray40             #666666 = 102 102 102
  grey40             #666666 = 102 102 102
  gray50             #7f7f7f = 127 127 127
  grey50             #7f7f7f = 127 127 127
  gray60             #999999 = 153 153 153
  grey60             #999999 = 153 153 153
  gray70             #b3b3b3 = 179 179 179
  grey70             #b3b3b3 = 179 179 179
  gray80             #cccccc = 204 204 204
  grey80             #cccccc = 204 204 204
  gray90             #e5e5e5 = 229 229 229
  grey90             #e5e5e5 = 229 229 229
  gray100            #ffffff = 255 255 255
  grey100            #ffffff = 255 255 255
  gray               #bebebe = 190 190 190
  grey               #bebebe = 190 190 190
  light-gray         #d3d3d3 = 211 211 211
  light-grey         #d3d3d3 = 211 211 211
  dark-gray          #a9a9a9 = 169 169 169
  dark-grey          #a9a9a9 = 169 169 169
  red                #ff0000 = 255   0   0
  light-red          #f03232 = 240  50  50
  dark-red           #8b0000 = 139   0   0
  yellow             #ffff00 = 255 255   0
  light-yellow       #ffffe0 = 255 255 224
  dark-yellow        #c8c800 = 200 200   0
  green              #00ff00 =   0 255   0
  light-green        #90ee90 = 144 238 144
  dark-green         #006400 =   0 100   0
  spring-green       #00ff7f =   0 255 127
  forest-green       #228b22 =  34 139  34
  sea-green          #2e8b57 =  46 139  87
  blue               #0000ff =   0   0 255
  light-blue         #add8e6 = 173 216 230
  dark-blue          #00008b =   0   0 139
  midnight-blue      #191970 =  25  25 112
  navy               #000080 =   0   0 128
  medium-blue        #0000cd =   0   0 205
  royalblue          #4169e1 =  65 105 225
  skyblue            #87ceeb = 135 206 235
  cyan               #00ffff =   0 255 255
  light-cyan         #e0ffff = 224 255 255
  dark-cyan          #008b8b =   0 139 139
  magenta            #ff00ff = 255   0 255
  light-magenta      #f055f0 = 240  85 240
  dark-magenta       #8b008b = 139   0 139
  turquoise          #40e0d0 =  64 224 208
  light-turquoise    #afeeee = 175 238 238
  dark-turquoise     #00ced1 =   0 206 209
  pink               #ffc0cb = 255 192 203
  light-pink         #ffb6c1 = 255 182 193
  dark-pink          #ff1493 = 255  20 147
  coral              #ff7f50 = 255 127  80
  light-coral        #f08080 = 240 128 128
  orange-red         #ff4500 = 255  69   0
  salmon             #fa8072 = 250 128 114
  light-salmon       #ffa07a = 255 160 122
  dark-salmon        #e9967a = 233 150 122
  aquamarine         #7fffd4 = 127 255 212
  khaki              #f0e68c = 240 230 140
  dark-khaki         #bdb76b = 189 183 107
  goldenrod          #daa520 = 218 165  32
  light-goldenrod    #eedd82 = 238 221 130
  dark-goldenrod     #b8860b = 184 134  11
  gold               #ffd700 = 255 215   0
  beige              #f5f5dc = 245 245 220
  brown              #a52a2a = 165  42  42
  orange             #ffa500 = 255 165   0
  dark-orange        #ff8c00 = 255 140   0
  violet             #ee82ee = 238 130 238
  dark-violet        #9400d3 = 148   0 211
  plum               #dda0dd = 221 160 221
  purple             #a020f0 = 160  32 240
```

Examples

See programs and their generated figures for **CPlot**::**point**() and **CPlot**::**plotType**().

See Also
CPlot::**lineType**(), **CPlot**::**point**().
CPlot::**plotType**().

CPlot::polarPlot

Synopsis
#include <**chplot.h**>
void polarPlot(**int** *angle_unit*);

Purpose
Set a 2D plot to use polar coordinates.

Return Value

None.

Parameter

angle_unit Specify the unit for mesurement of an angular position. The following options are available:

- **PLOT_ANGLE_DEG** Angles measured in degree.
- **PLOT_ANGLE_RAD** Angles measured in radian.

Description

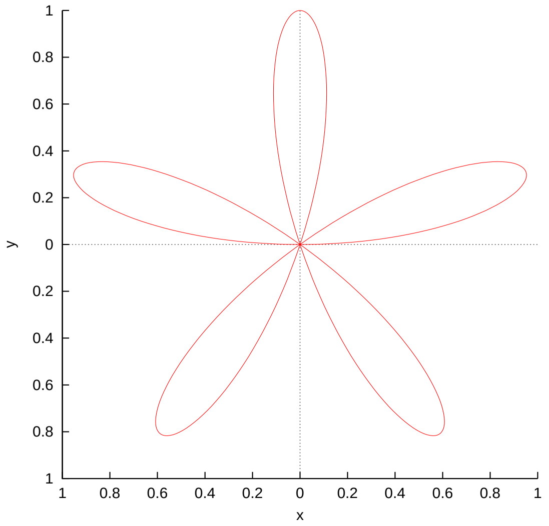
Set a 2D plot to polar coordinates. In polar mode, two numbers, θ and r, are required for each point. First two arguments in member functions **CPlot::data2D**() and **CPlot::data2DCurve**() are the phase angle θ and magnitude r of points to be plotted.

Example 1

Compare with the example output in **CPlot::border**().

```
#include <math.h>
#include <chplot.h>

#define NUM 360
int main() {
    int i;
    double theta[NUM], r[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
      theta[i] = 0 + i*M_PI/(NUM-1);  // linspace(theta, 0, PI)
      r[i] = sin(5*theta[i]);
    }
    plot.polarPlot(PLOT_ANGLE_RAD);
    plot.data2DCurve(theta, r, NUM);
    plot.sizeRatio(1);
    plot.plotting();
    return 0;
}
```

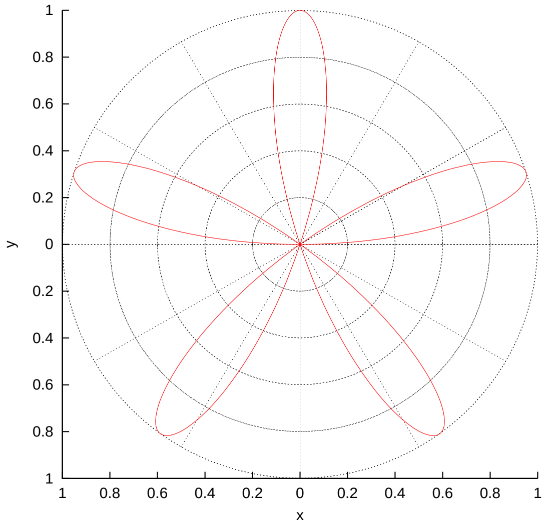
Output



Example 2

```
#include <math.h>
#include <chplot.h>

#define NUM 360
int main() {
    int i;
    double theta[NUM], r[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
      theta[i] = 0 + i*M_PI/(NUM-1);  // linspace(theta, 0, PI)
      r[i] = sin(5*theta[i]);
    }
    plot.polarPlot(PLOT_ANGLE_RAD);
    plot.data2DCurve(theta, r, NUM);
    plot.sizeRatio(1);
    plot.grid(PLOT_ON);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::**grid**().

CPlot::polygon

Synopsis in Ch
#include <**chplot.h**>
int polygon(double *x*[:], **double** *y*[:], ... /* **double** *z*[:]; [**int** *num*] */);

Synopsis in C++
#include <**chplot.h**>

int polygon(**double** *x*[], **double** *y*[], **int** *num*);
int polygon(**double** *x*[], **double** *y*[], **double** *z*[], **int** *num*);

Syntax in Ch
polygon(*x*, *y*)
polygon(*x*, *y*, *num*)
polygon(*x*, *y*, *z*)
polygon(*x*, *y*, *z*, *num*)

Syntax in C++
polygon(*x*, *y*, *num*)
polygon(*x*, *y*, *z*, *num*)

Purpose
Add a polygon to a plot.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
x An array containing the x coordinates of the vertices of the polygon.

y An array containing the y coordinates of the vertices of the polygon.

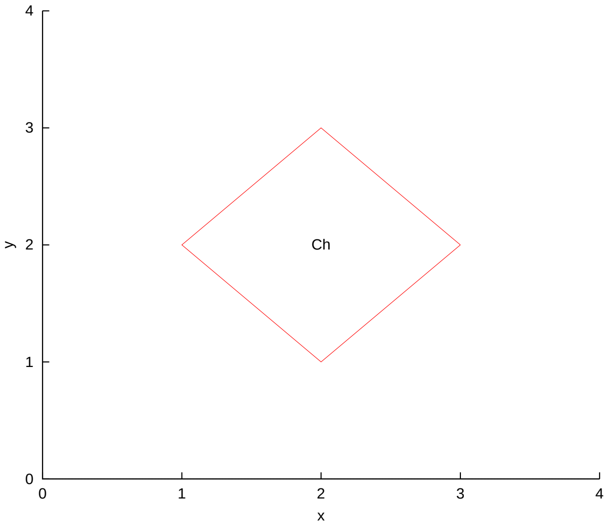
z An array containing the z coordinates of the vertices of the polygon.

num The number of elements of arrays *x*, *y*, and *z*.

Description
This function adds a polygon to a plot. It is a convenience function for creation of geometric primitives. A polygon added with this function is counted as a data set for later calls to **CPlot**::**legend**() and **CPlot**::**plotType**(). For 2D rectangular plots and 3D cartesian plots, *x*, *y*, and *z* contain the polygon vertices specified in units of the x, y, and z axes. However, for 2D plots, *z* is ignored. For 2D polar and 3D cylindrical plots, the locations of the vertices are specified in polar coordinates where *x* is θ, *y* is r, and *z* is unchanged. Again, for 2D plots, *z* is ignored. For 3D plots with spherical coordinates *x* is θ, *y* is ϕ and *z* is r. Each of the points is connected to the next in a closed chain. The line type and width vary depending on the terminal type used (see **CPlot**::**outputType**). Typically, changing the line type will change the color of the line or make it dashed or dotted. All terminals support at least six different line types.
Example 1

```
#include <chplot.h>

double x[5] = {3, 2, 1, 2, 3}, y[5] = {2, 3, 2, 1, 2};
int main() {
    class CPlot plot;

    plot.polygon(x, y, 5);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_XY, 0, 4, 1);
    plot.text("Ch", PLOT_TEXT_CENTER, 2, 2);
    plot.plotting();
    return 0;
}
```

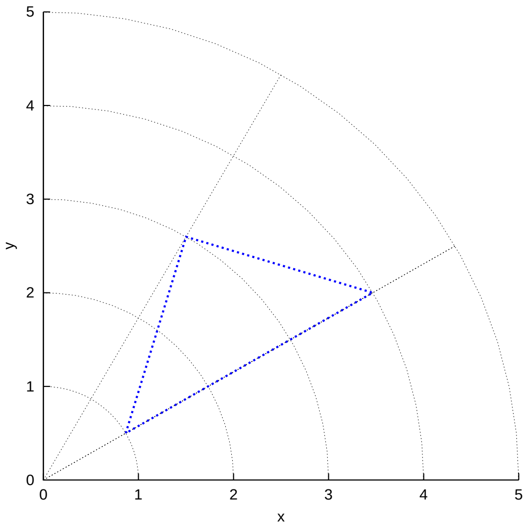
Output



Example 2

```
#include <chplot.h>
#include <math.h>

double theta[4] = {30, 60, 30, 30}, r[4] = {1, 3, 4, 1};
int main(){
    class CPlot plot;

    plot.grid(PLOT_ON);
    plot.polarPlot(PLOT_ANGLE_DEG);
    plot.polygon(theta, r, 4);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.lineType(0, 3, 4);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_XY, 0, 5, 1);
    plot.plotting();
    return 0;
}
```

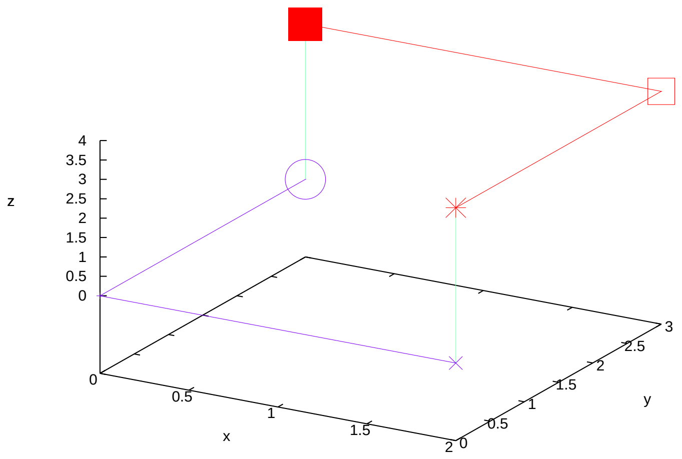
Output



Example 3

```
#include <chplot.h>

double x[7] = {0, 2, 2, 2, 0, 0, 0},
       y[7] = {0, 0, 0, 3, 3, 3, 0},
       z[7] = {0, 0, 4, 4, 4, 0, 0};
int main() {
    class CPlot plot;
    int datasetnum, pointtype, pointsize;

    plot.dimension(3);
    plot.polygon(x, y, z, 7);
    plot.point(0, 0, 0);
    plot.point(2, 0, 0);
    plot.point(2, 0, 4);
    plot.point(2, 3, 4);
    plot.point(0, 3, 4);
    plot.point(0, 3, 0);
    for(datasetnum=1, pointtype=1, pointsize=1;
        datasetnum <= 6;
        datasetnum++, pointtype++, pointsize++)
       plot.plotType(PLOT_PLOTTYPE_POINTS,
                     datasetnum, pointtype, pointsize);
    plot.removeHiddenLine(PLOT_OFF);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::data2D(), **CPlot::data2DCurve**(), **CPlot::data3D**(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **CPlot::circle**(), **CPlot::line**(), **CPlot::outputType**(), **CPlot::plotType**(), **CPlot::point**(), **CPlot::rectangle**().

CPlot::rectangle

Synopsis
#include <chplot.h>
int rectangle(double *x*, **double** *y*, **double** *width*, **double** *height*);
Purpose
Add a polygon to a 2D plot.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

x The x coordinate of the lower-left corner of the rectangle.

y The y coordinate of the lower-left corner of the rectangle.

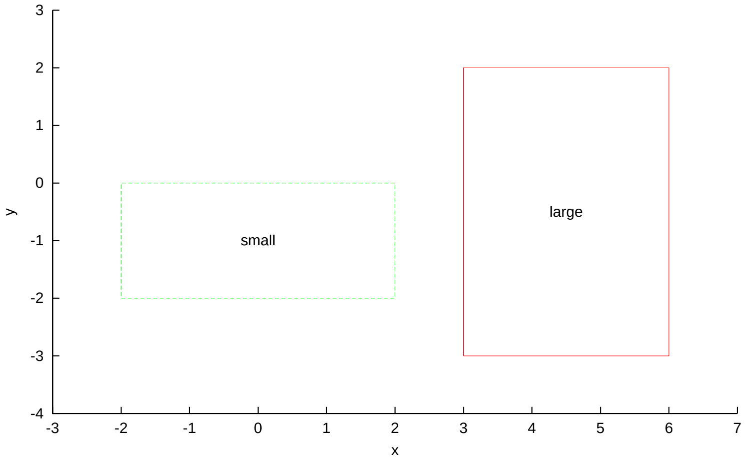
width The width of the rectangle.

height The height of the rectangle.

Description
This function adds a rectangle to a 2D plot. It is a convenience function for creation of geometric primitives. A rectangle added with this function is counted as a data set for later calls to **CPlot::legend**() and **CPlot::plotType**(). For rectangular plots, *x* and *y* are the coordinates of the lower-left corner of the rectangle. For polar plots, the coordinates of the lower-left corner are given in polar coordinates where *x* is theta and *y* is r. In both cases the *width* and *height* are the dimensions of the rectangle in rectangular coordinates.
Example 1

```
#include <chplot.h>

int main() {
    double x1 = 3, y1 = -3, width1 = 3, height1 = 5;
    double x2 = -2, y2 = -2, width2 = 4, height2 = 2;
    class CPlot plot;

    plot.rectangle(x1, y1, width1, height1);
    plot.rectangle(x2, y2, width2, height2);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_X, -3, 7, 1);
    plot.axisRange(PLOT_AXIS_Y, -4, 3, 1);
    plot.axis(PLOT_AXIS_XY, PLOT_OFF);
    plot.text("large", PLOT_TEXT_CENTER, 4.5, -0.5);
    plot.text("small", PLOT_TEXT_CENTER, 0, -1);
    plot.plotting();
    return 0;
}
```

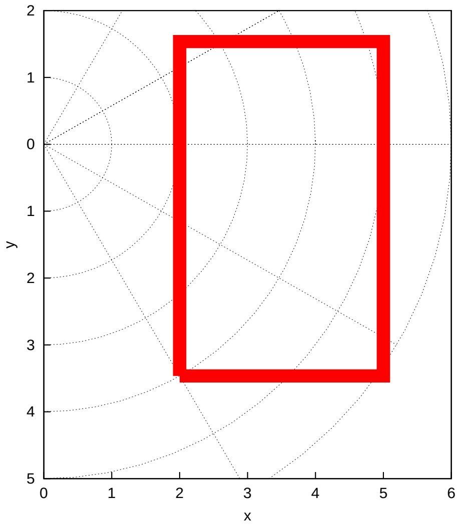
Output



Example 2

```
#include <chplot.h>

int main() {
    double theta1 = -60, r1 = 4, width1 = 3, height1 = 5;
    class CPlot plot;

    plot.grid(PLOT_ON);
    plot.polarPlot(PLOT_ANGLE_DEG);
    plot.rectangle(theta1, r1, width1, height1);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.lineType(0, 0, 25);
    plot.sizeRatio(-1);
    plot.axisRange(PLOT_AXIS_X, 0, 6, 1);
    plot.axisRange(PLOT_AXIS_Y, -5, 2, 1);
    plot.axis(PLOT_AXIS_XY, PLOT_OFF);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::data2D(), **CPlot::data2DCurve**(), **CPlot::data3D**(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **CPlot::circle**(), **CPlot::line**(), **CPlot::outputType**(), **CPlot::plotType**(), **CPlot::point**(), **CPlot::polygon**().

CPlot::removeHiddenLine

Synopsis
#include <**chplot.h**>
void removeHiddenLine(int *flag*);

Purpose
Enable or disable hidden line removal for 3D surface plots.

Return Value
None.

Parameter
flag This parameter can be set to:

PLOT_ON Enable hidden line removal.

PLOT_OFF Disable hidden line removal.

Description
Enable or disable hidden line removal for 3D surface plots. This option is only valid for 3D plots with a plot type set to **PLOT_PLOTTYPE_LINES** or **PLOT_PLOTTYPE_LINESPOINTS** with surface display enabled. By default hidden line removal is enabled. This function should be called after data set are added to the plot. The **PLOT_CONTOUR_SURFACE** option for **CPlot::contourMode**() does not work when

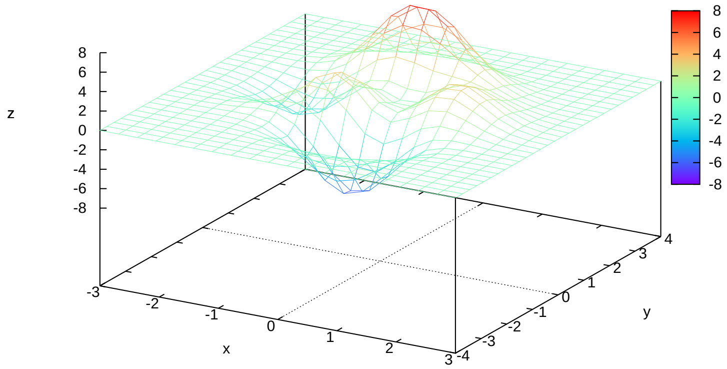
hidden line removal is enabled. **CPlot::point**() cannot be used when hidden line removal is enabled. By default, the hidden lines are removed.

Example 1

Compare with the output for the example in **CPlot::data3D**(), **CPlot::contourLabel**(), and **CPlot::contourLevels**().

```
#include <math.h>
#include <chplot.h>

#define NUMX 20
#define NUMY 30
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    int i,j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -3 + i*6.0/(NUMX-1); // linspace(x, -3, 3);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -4 + i*8.0/(NUMY-1); // linspace(y, -4, 4);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[NUMY*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.removeHiddenLine(PLOT_OFF);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
    return 0;
}
```

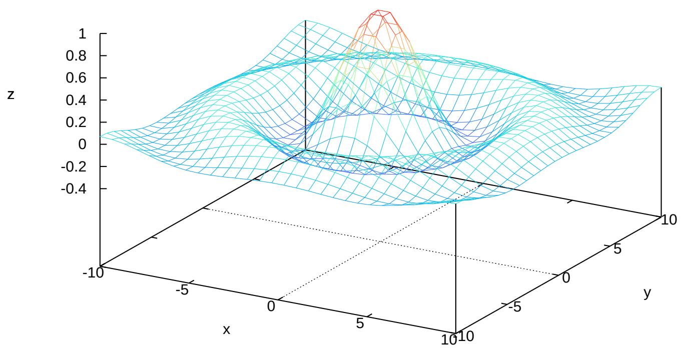
Output



Example 2

```
#include <math.h>
#include <chplot.h>

#define NUMX 30
#define NUMY 30

int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double r;
    int i, j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -10 + i*20.0/(NUMX-1); // linspace(x, -10, 10);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -10 + i*20.0/(NUMY-1); // linspace(y, -10, 10);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[NUMY*i+j] = sin(r)/r;
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0);
    plot.removeHiddenLine(PLOT_OFF);
    plot.colorBox(PLOT_OFF);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::data3D(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **CPlot::contourMode**(), **CPlot::plotType**(), **CPlot::point**(), **CPlot::showMesh**().

CPlot::scaleType

Synopsis in Ch
#include <chplot.h>
void scaleType(int *axis*, **int** *scale_type*, ... /* [**double** *base*] */);

Synopsis in C++
#include <chplot.h>
void scaleType(int *axis*, **int** *scale_type*);
void scaleType(int *axis*, **int** *scale_type*, **double** *base*);

Syntax in Ch and C++
scaleType(*axis*, *scale_type*)
scaleType(*axis*, *scale_type*, *base*)

Purpose
Set the axis scale type for a plot.

Return Value
None.

Parameters
axis The axis to be modified. Valid values are:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

scale_type The scale type for the specified axis. Valid values are:

PLOT_SCALETYPE_LINEAR Use a linear scale for a specified axis.

PLOT_SCALETYPE_LOG Use a logarithmic scale for a specified axis.

base The base for a log scale. For log scales the default base is 10.

Description
Using this function an axis can be modified to have a logarithmic scale. By default, axes are in linear scale. For a semilog scale in the x-coordinate, call the member function

```
plot.scaleType(PLOT_AXIS_X, PLOT_SCALETYPE_LOG);
```

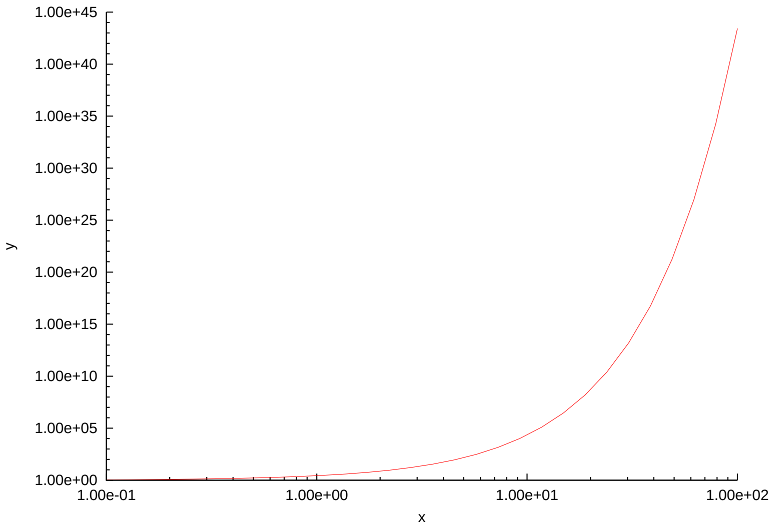
For a logarithmic base 2, call the member function

```
plot.scaleType(PLOT_AXIS_X, PLOT_SCALETYPE_LOG, 2); // log base = 2
```

Example

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= pow(10.0, -1 + i*3.0/(NUM-1));  // logspace(x, -1, 2);
       y[i] = exp(x[i]);                   // y = exp(x);
    }
    plot.scaleType(PLOT_AXIS_XY, PLOT_SCALETYPE_LOG, 10.0);
    plot.ticsFormat(PLOT_AXIS_XY, "%.2e");
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



CPlot::showMesh

Synopsis

#include <chplot.h>
void showMesh(int *flag*);

Purpose

Display 3D data.

Return Value

None.

Parameters

flag This parameter can be set to:

PLOT_ON Enable the display of 3D data.

PLOT_OFF Disable the display of 3D data.

Description

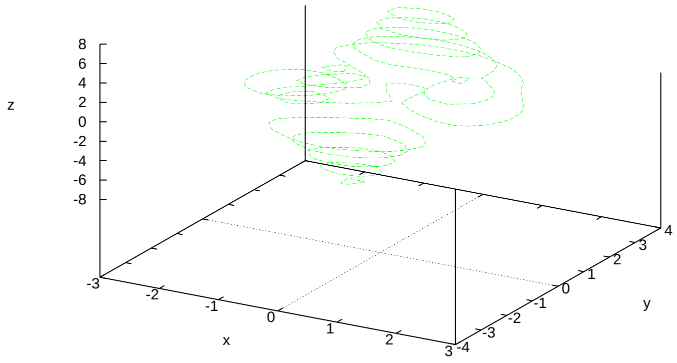
Enable or disable the display of 3D data. If this option is disabled, data points or lines will not be drawn. This option is useful with **CPlot**::**contourMode**() to display contour lines without the display of a surface grid.

Example 1

Compare with the output for the example in **CPlot**::**contourLabel**().

```
#include <math.h>
#include <chplot.h>

#define NUMX 20
#define NUMY 30
#define NUMLV 10
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double levels[NUMLV];
    int datasetnum =0, i, j;
    int line_type = 1, line_width = 1;
    class CPlot plot;

    for(i=0; i<NUMLV; i++) {
       levels[i]= -6 + i*12.0/(NUMLV-1); // linspace(levels, -6, 6);
    }
    for(i=0; i<NUMX; i++) {
       x[i]= -3 + i*6.0/(NUMX-1); // linspace(x, -3, 3);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -4 + i*8.0/(NUMY-1); // linspace(y, -4, 4);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[NUMY*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width);
    plot.contourLabel(PLOT_ON);
    plot.showMesh(PLOT_OFF);
    plot.contourMode(PLOT_CONTOUR_SURFACE);
    plot.contourLevels(levels, NUMLV);
    plot.plotting();
    return 0;
}
```

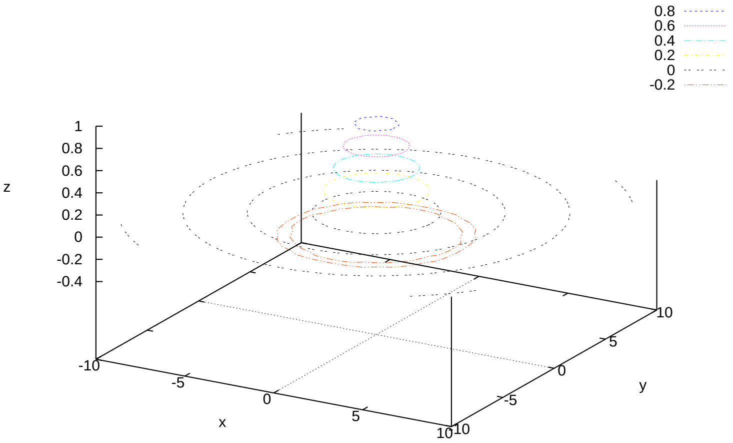
Output



Example 2

```
#include <math.h>
#include <chplot.h>

#define NUMX 30
#define NUMY 30
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    double r;
    int datasetnum =0, i, j;
    int line_type = 1, line_width = 1;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -10 + i*20.0/(NUMX-1); // linspace(x, -10, 10);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -10 + i*20.0/(NUMY-1); // linspace(y, -10, 10);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            r = sqrt(x[i]*x[i]+y[j]*y[j]);
            z[NUMY*i+j] = sin(r)/r;
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.plotType(PLOT_PLOTTYPE_LINES, datasetnum);
    plot.lineType(datasetnum, line_type, line_width);
    plot.contourLabel(PLOT_ON);
    plot.showMesh(PLOT_OFF);
    plot.contourMode(PLOT_CONTOUR_SURFACE);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::data3D(), **CPlot::data3DCurve**(), **CPlot::data3DSurface**(), **CPlot::contourMode**(), **CPlot::plotType**().

CPlot::size

Synopsis
#include <**chplot.h**>
void size(**double** *x_scale*, **double** *y_scale*);

Purpose
Scale the plot itself relative to the size of the output file or canvas.

Return Value
None.

Parameters

x_scale A positive multiplicative scaling factor in the range of (0, 1) for the x direction.

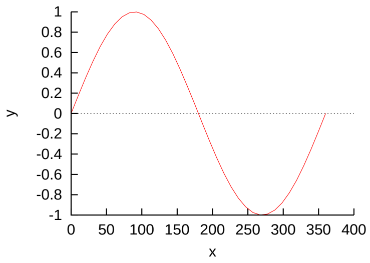
y_scale A positive multiplicative scaling factor in the range of (0, 1) for the y direction.

Description
This function can be used to scale a plot itself relative to the size of the output file or canvas. If the plot is displayed on a screen, the plot is scaled relative to the size of its canvas. If the plot is saved to a file, or output to the stdout stream, the size of the plot image is scaled relative to the output file. For a plot with subplots, this function should be called before **CPlot::subplot**() is called.

Example

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.data2DCurve(x, y, NUM);
    plot.size(.5, .5);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::**size3D**(), **CPlot**::**sizeOutput**(), and **CPlot**::**sizeRatio**().

CPlot::size3D

Synopsis

#include <**chplot.h**>
void size3D(**float** *scale*, **float** *z_scale*);

Purpose

Scale a 3D plot.

Return Value

None.

Parameters

scale A positive multiplicative scaling factor that is applied to the entire plot.

z_scale A positive multiplicative scaling factor that is applied to the z-axis only.

Description

This function can be used to scale a 3D plot to the desired size. By default, *scale* and *z_scale* are both 1.

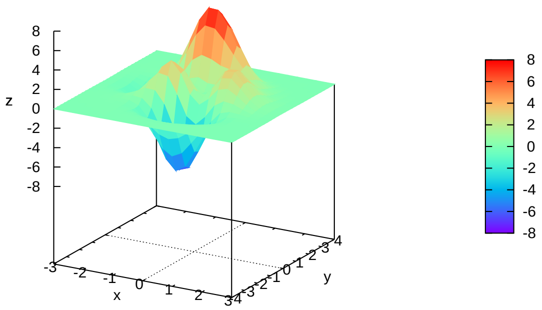
Example

Compare with the output for examples in **CPlot**::**data3D**() and **CPlot**::**data3DSurface**().

```
#include <math.h>
#include <chplot.h>
```

```
#define NUMX 20
#define NUMY 30
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    int i,j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -3 + i*6.0/(NUMX-1); // linspace(x, -3, 3);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -4 + i*8.0/(NUMY-1); // linspace(y, -4, 4);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[NUMY*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.size3D(0.5, 2);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::**size**().

CPlot::sizeOutput

Synopsis
#include <**chplot.h**>
void sizeOutput(int *xpixels*, **int** *ypixels*);

Purpose
Change the size of an output file.

Return Value
None.

Parameters

xpixels A positive integral number for the number of pixels in the x direction.

ypixels A positive integral number for the number of pixels in the y direction.

Description
This function can be used to change the default size (640x480) of an output image file for the plot.

Example

```
#include<math.h>
#include<chplot.h>

#define NUM 36
int main() {
    int i, xpixels = 400, ypixels=600;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.data2DCurve(x, y, NUM);
    plot.outputType(PLOT_OUTPUTTYPE_FILE, "png", "sizeOutput.png");
    plot.sizeOutput(xpixels, ypixels); // size of image is 400x600 pixels
    plot.plotting();
    return 0;
}
```

See Also
CPlot::**size**(), **CPlot**::**size3D**(), and **CPlot**::**sizeRatio**().

CPlot::**sizeRatio**

Synopsis
#include <**chplot.h**>
void sizeRatio(**float** *ratio*);

Purpose
Change the aspect ratio for a plot.

Return Value
None.

Parameter

ratio The aspect ratio for the plot.

Description

The function sets the aspect ratio for the plot. The meaning of *ratio* changes depending on its value. For a positive *ratio*, it is the ratio of the length of the y-axis to the length of the x-axis. So, if *ratio* is 2, the y-axis will be twice as long as the x-axis. If *ratio* is zero, the default aspect ratio for the terminal type is used. If it is negative, *ratio* is the ratio of the y-axis units to the x-axis units. So, if *ratio* is -2, one unit on the y-axis will be twice as long as one unit on the x-axis.

Portability

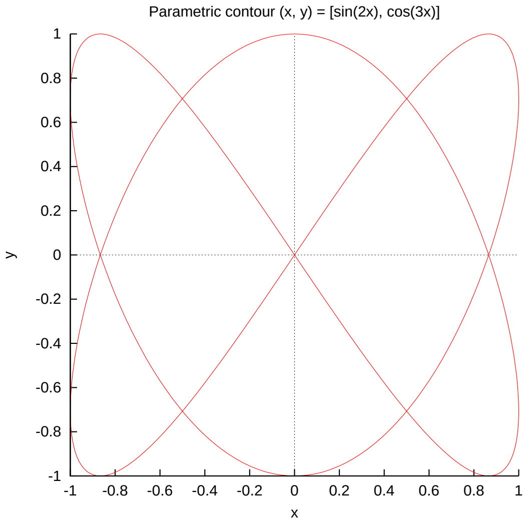
The aspect ratio specified is not exact on some terminals. To get a true ratio of 1, for example, some adjustment may be necessary.

Example

Compare with output for example in **plotxy**().

```
#include <math.h>
#include <chplot.h>

#define NUM 360
int main() {
    int i;
    double t[NUM], x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       t[i]= 0 + i*2*M_PI/(NUM-1); // linspace(t, 0, 2*M_PI);
       x[i] = sin(2 * t[i]); // x = sin(2*t);
       y[i] = cos(3 * t[i]); // y = cos(3*t);
    }
    plot.data2DCurve(x, y, NUM);
    plot.title("Parametric contour (x, y) = [sin(2x), cos(3x)]");
    plot.label(PLOT_AXIS_X, "x");
    plot.label(PLOT_AXIS_Y, "y");
    /* both x and y axes the same length */
    plot.sizeRatio(1);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::**size**(), **CPlot**::**size3D**().

CPlot::smooth

Synopsis
#include <**chplot.h**>
void smooth(**int** num, **char** * *option*);

Syntax
smooth(*num, option*)

Purpose
Smooth a plotting curve by interpolation or approximation of data.

Return Value
None.

Parameters

num The data set for the curve to be smooth.

option The options smooth a curve.

Description

This function can be used to readily plot a smooth curve through your data points for a 2D plotting by interpolation and approximation of data. However, sophisticated data processing may be performed by preprocessing the data in your Ch program.

The argument `option` of string type with the following values can be used to smooth the data points.

```
{unique | frequency | csplines | acsplines | bezier | sbezier}
```

The `unique` and `frequency` plot the data after making them monotonic. Each of the other options uses the data to determine the coefficients of a continuous curve between the end points of the data. This curve is then plotted in the same manner as a function, that is, by finding its value at uniform intervals along the abscissa and connecting these points with straight line segments (if a line style is chosen).

If the axis range is determined automatically, the ranges will be computed such that the plotted curve lies within the borders of the graph.

If **CPlot**::**axisRange**() is called, and the smooth option is either `acspline` or `cspline`, the sampling of the generated curve is done across the intersection of the x range covered by the input data and the fixed abscissa range as defined by **CPlot**::**axisRange**() for x-axis.

If too few points are available to allow the selected option to be applied, an error message is produced. The minimum number is one for `unique` and `frequency` four for `acsplines`, and three for the others.

`"acsplines"` The `acsplines` option approximates the data with a “natural smoothing spline”. After the data are made monotonic in x a curve is piecewise constructed from segments of cubic polynomials whose coefficients are found by the weighting the data points; the weights are taken from the third column in the data file or data in the memory. If the data is from a data file, that default can be modified by the third entry in the option `using` list for **CPlot**::**dataFile**() as shown below.

```
plot.dataFile("datafile", "using 1:2:(1.0)");
plot.smooth(plot.dataSetNum(), "acsplines");
```

`bezier` The `bezier` option approximates the data with a Bezier curve of degree n (the number of data points) that connects the endpoints.

`"csplines"` The `csplines` option connects consecutive points by natural cubic splines after rendering the data monotonic.

`"sbezier"` The `sbezier` option first renders the data monotonic and then applies the `bezier` algorithm.

`"unique"` The `unique` option makes the data monotonic in x; points with the same x-value are replaced by a single point having the average y-value. The resulting points are then connected by straight line segments.

`"frequency"` The `frequency` option makes the data monotonic in x; points with the same x-value are replaced by a single point having the summed y-values. The resulting points are then connected by straight line segments.

Examples

See an example on page 184 for **CPlot**:**plotType**() on how data for an error bar are smoothed by the option `csplines`.

CPlot::subplot

Synopsis

#include <**chplot.h**>

int subplot(**int** *row*, **int** *col*);

Purpose
Create a group of subplots.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
row The number of rows in the subplot.

col The number of columns in the subplot.

Description
This function allocates memory for (*row***col*)-1 instances of the **CPlot** class to be used in a subplot. The location of the drawing origin and the scale for each element of the subplot is set automatically. The first element of the subplot (an instance of the **CPlot** class) is created normally by the user before this function is called. The remaining elements of the subplot are created and stored internally when this function is called. These elements are accessible through a pointer returned by the **CPlot**::**getSubplot**() function. Calling **CPlot**::**subplot**() with *row* = *col* = 1 has no effect.

Example
See **CPlot**::**getSubplot**().

See Also
CPlot::**origin**(),**CPlot**::**getSubplot**(), **CPlot**::**size**().

CPlot::text

Synopsis in Ch
#include <chplot.h>
void text(**string_t** string, **int** *just*, **double** *x*, **double** *y*, ... /* **double** *z* */);

Synopsis in C++
#include <chplot.h>
void text(**char** * string, **int** *just*, **double** *x*, **double** *y*);
void text(**char** * string, **int** *just*, **double** *x*, **double** *y*, **double** *z*);

Syntax
text(*string*, *just*, *x*, *y*)
text(*string*, *just*, *x*, *y*, *z*)

Purpose
Add a text string to a plot.

Return Value
None.

Parameters
string The string to be added at location (*x*,*y*) for 2D plots or (*x*,*y*,*z*) for 3D plots. The location of the text is in plot coordinates.

just The justification of the text. Valid values are:

- **PLOT_TEXT_LEFT** The specified location is the left side of the text string.
- **PLOT_TEXT_RIGHT** The specified location is the right side of the text string.
- **PLOT_TEXT_CENTER** The specified location is the center of the text string.

x The x position of the text.

y The y position of the text.

z The z position of the text. This argument is ignored for a 2D plot.

Description

This function can be used to add text to a plot at an arbitrary location.

Example

See **CPlot**::**arrow**(), **CPlot**::**polygon**(), and **CPlot**::**rectangle**().

CPlot::tics

Synopsis

#include <chplot.h>
void tics(int *axis*, **int** *flag*)

Purpose

Enable or disable display of axis tics.

Return Value

None.

Parameters

axis The axis which labels are added to. This parameter can take one of the following values:

- **PLOT_AXIS_X** Select the x axis only.
- **PLOT_AXIS_X2** Select the x2 axis only.
- **PLOT_AXIS_Y** Select the y axis only.
- **PLOT_AXIS_Y2** Select the y2 axis only.
- **PLOT_AXIS_Z** Select the z axis only.
- **PLOT_AXIS_XY** Select the x and y axes.
- **PLOT_AXIS_XYZ** Select the x, y, and z axes.

flag This parameter can be set to:

- **PLOT_ON** Enable drawing of tics for the specified axis.
- **PLOT_OFF** Disable drawing of tics for the specified axis.

Description

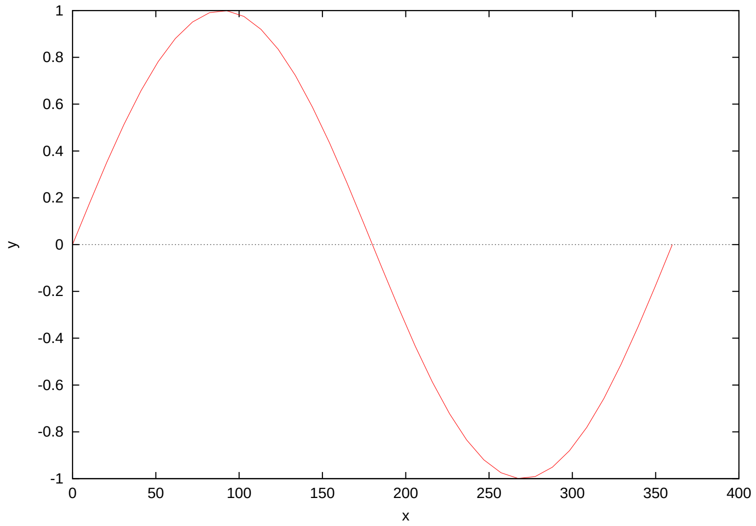
Enable or disable the display of tics and numerical labels for an axis. By default, tics are displayed for the x and y axes on 2D plots and for the x, y and z axes on 3D plots.

Example

Compare with the output for examples in **CPlot::data2D**() and **CPlot::data2DCurve**().

```
#include <math.h>
#include <chplot.h>


#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.data2DCurve(x, y, NUM);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.tics(PLOT_AXIS_XY, PLOT_OFF);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::ticsDirection(), **CPlot::ticsFormat**(), **CPlot::ticsLabel**(), **CPlot::ticsLevel**(), **CPlot::ticsLocation**(), and **CPlot::ticsMirror**().

CPlot::ticsDay

Synopsis

#include <chplot.h>
void ticsDay(int *axis*);

Purpose

Set axis tic-marks to days of the week.

Return Value

None.

Parameter

axis The *axis* parameter can take one of the following values:

- **PLOT_AXIS_X** Select the x axis only.
- **PLOT_AXIS_X2** Select the x2 axis only.
- **PLOT_AXIS_Y** Select the y axis only.
- **PLOT_AXIS_Y2** Select the y2 axis only.
- **PLOT_AXIS_Z** Select the z axis only.
- **PLOT_AXIS_XY** Select the x and y axes.
- **PLOT_AXIS_XYZ** Select the x, y, and z axes.

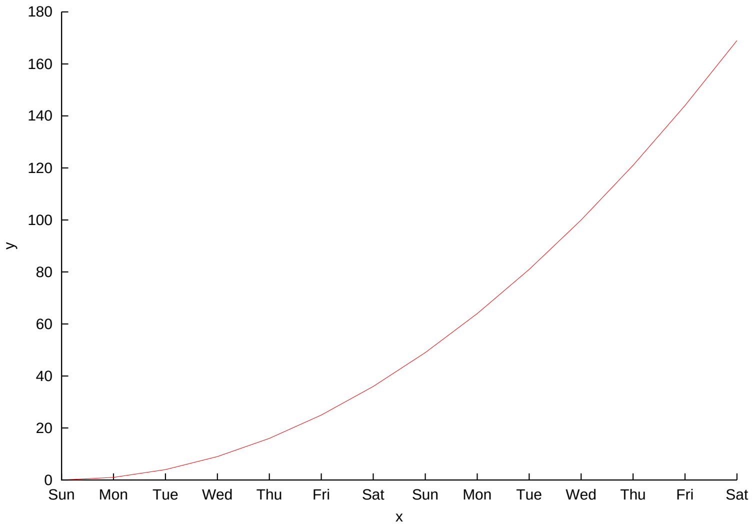
Description

Sets axis tic marks to days of the week (0=Sunday, 6=Saturday). Values greater than 6 are converted into the value of modulo 7.

Example

```
#include <math.h>
#include <chplot.h>


#define NUM 14
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= i; // linspace(x, 0, NUM-1)
       y[i] = x[i]*x[i]; // y = x.*x;
    }
    plot.ticsDay(PLOT_ON);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::ticsMonth(), **CPlot::ticsLabel**().

CPlot::ticsDirection

Synopsis
#include <**chplot.h**>
void ticsDirection(**int** *direction*);

Purpose
Set the direction in which the tic-marks are drawn for an axis.

Return Value
None.

Parameter
direction Direction tic-marks are drawn. Can be set to:

PLOT_TICS_IN Draw axis tic-marks inward.

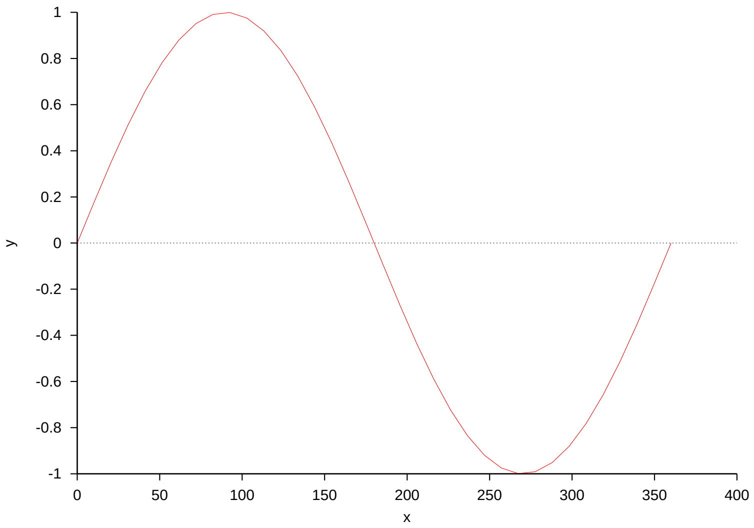
PLOT_TICS_OUT Draw axis tic-marks outward.

Description
Set the direction in which tic-marks are drawn in the inward or outward direction from the axes. The default is **PLOT_TICS_IN**.

Example
Compare with the output for examples in **CPlot::data2D**() and **CPlot::data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.ticsDirection(PLOT_TICS_OUT);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::**ticsDay**(), **CPlot**::**ticsLabel**(), **CPlot**::**ticsLocation**(), and **CPlot**::**ticsMonth**().

CPlot::ticsFormat

Synopsis in Ch

#include <**chplot.h**>
void ticsFormat(int *axis*, **string_t** *format*);

Synopsis in C++

#include <**chplot.h**>
void ticsFormat(int *axis*, **char** **format*);

Purpose

Set the number format for tic labels.

Return Value

None.

Parameters

axis The axis to be modified. Valid values are:

- **PLOT_AXIS_X** Select the x axis only.
- **PLOT_AXIS_X2** Select the x2 axis only.
- **PLOT_AXIS_Y** Select the y axis only.
- **PLOT_AXIS_Y2** Select the y2 axis only.
- **PLOT_AXIS_Z** Select the z axis only.
- **PLOT_AXIS_XY** Select the x and y axes.
- **PLOT_AXIS_XYZ** Select the x, y, and z axes.

format A C-style conversion specifier. Any conversion specifier suitable for double precision floats is acceptable, but other formats are available.

Description

This function allows for custom number formats for tic-mark labels. The default format is `"%g"`. The table below gives some of the available tics formats in C style.

Format	Effect
`%f`	Decimal notation for a floating point number. By default, 6 digits are displayed after the decimal point.
`%e` or `%E`	Scientific notation for a floating point value. There is alwayse one digit to the left of the decimal point. By default, 6 digits are displayed to the right of the decimal point.
`%g` or `%G`	A floating point number. If the value requires an exponent less than -4 or greater than the precision, e or E is used, otherwise f is used.

These format specifiers can be used with the standard C flag characters (-, +, #, etc.), minimum field width specifications and precision specifiers for function **fprintf**(). See **fprintf**() for details.

Examples:

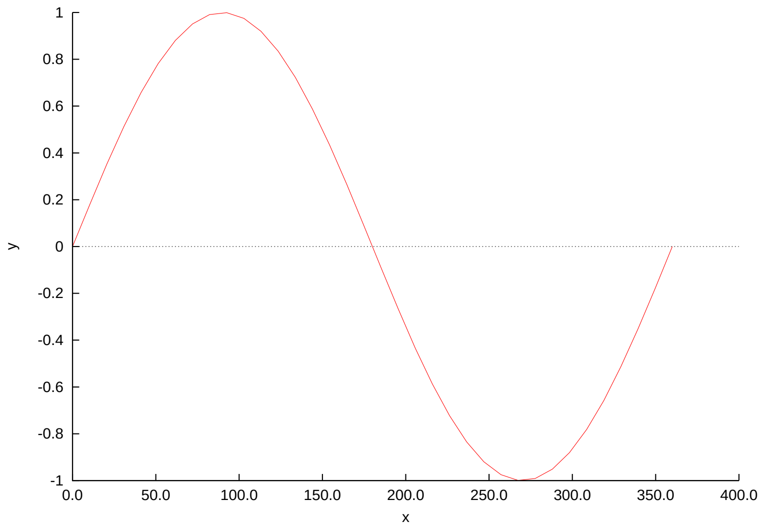
format	number	output
”%f”	234.5678	234.567800
”%.2f”	234.5678	234.57
”%e”	123.456	0.123456e+03
”%E”	123.456	0.123456E+03
”%5.0f”	125.0	125
”%#5.0f”	125.0	125.

Example

Compare with the output for examples in **CPlot::data2D**() and **CPlot::data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.data2DCurve(x, y, NUM);
    plot.ticsFormat(PLOT_AXIS_X, "%.1f");
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::ticsLabel(), **fprintf**().

CPlot::ticsLabel

Synopsis in Ch

#include <chplot.h>
void ticsLabel(int *axis*, ... /* [[**string_t** *label*, **double** *position*], ...] */);

Synopsis in C++

#include <chplot.h>
void ticsLabel(int *axis*, **char** **label*, **double** *position*);

Syntax in Ch

ticsLabel(*axis*, *label*, *position*)
ticsLabel(*axis*, *label1*, *position1*, *label2*, *position2*)
etc.

Syntax in C++

ticsLabel(*axis*, *label1*, *position1*)
ticsLabel(*axis*, *label2*, *position2*)

Purpose

Add tic-marks with arbitrary labels to an axis.

Return Value

None.

Parameters

axis The axis which labels are added to. This parameter can take one of the following values:

- **PLOT_AXIS_X** Select the x axis only.
- **PLOT_AXIS_X2** Select the x2 axis only.
- **PLOT_AXIS_Y** Select the y axis only.
- **PLOT_AXIS_Y2** Select the y2 axis only.
- **PLOT_AXIS_Z** Select the z axis only.
- **PLOT_AXIS_XY** Select the x and y axes.
- **PLOT_AXIS_XYZ** Select the x, y, and z axes.

label The tic-mark label.

position The position of the tic-mark on the axis.

Description

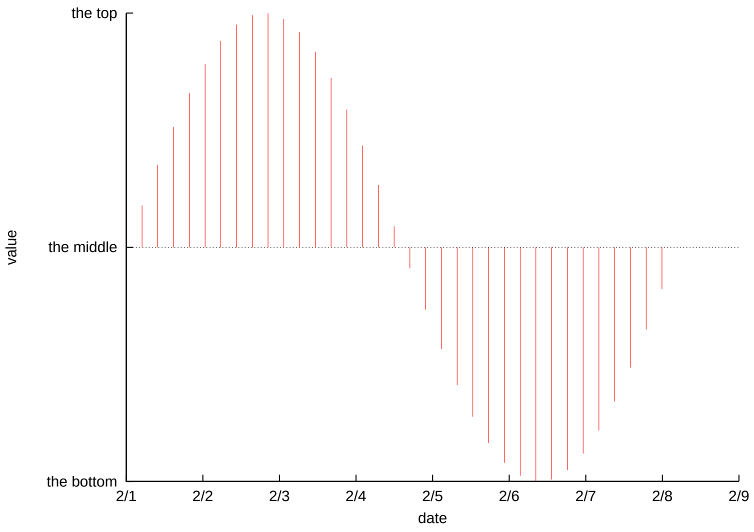
Add tic marks with arbitrary labels to an axis. The axis specification is followed by one or more pairs of arguments. Each pair consists of a label string and a double precision floating point position . This function disables numerical labels for the specified axis. This function can be called multiple times to set tic labels for an axis.

Example

Compare with the output for examples in **CPlot::data2D**() and **CPlot::data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.label(PLOT_AXIS_X, "date");
    plot.label(PLOT_AXIS_Y, "value");
    plot.data2DCurve(x, y, NUM);
    plot.plotType(PLOT_PLOTTYPE_IMPULSES, 0);
    plot.axisRange(PLOT_AXIS_X, 0, 400);
    plot.ticsLabel(PLOT_AXIS_X, "2/1", 0);
    plot.ticsLabel(PLOT_AXIS_X, "2/2", 50);
    plot.ticsLabel(PLOT_AXIS_X, "2/3", 100);
    plot.ticsLabel(PLOT_AXIS_X, "2/4", 150);
    plot.ticsLabel(PLOT_AXIS_X, "2/5", 200);
    plot.ticsLabel(PLOT_AXIS_X, "2/6", 250);
    plot.ticsLabel(PLOT_AXIS_X, "2/7", 300);
    plot.ticsLabel(PLOT_AXIS_X, "2/8", 350);
    plot.ticsLabel(PLOT_AXIS_X, "2/9", 400);
    //    plot.ticsLabel(PLOT_AXIS_Y, "the bottom", -1, "the middle", 0,
    //                        "the top", 1);
    plot.ticsLabel(PLOT_AXIS_Y, "the bottom", -1);
    plot.ticsLabel(PLOT_AXIS_Y, "the middle", 0);
    plot.ticsLabel(PLOT_AXIS_Y, "the top", 1);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::**ticsDirection**(), **CPlot**::**ticsFormat**(), **CPlot**::**ticsLevel**(), **CPlot**::**ticsLocation**().

CPlot::ticsLevel

Synopsis

#include <chplot.h>
void ticsLevel(double *level*);

Purpose

Set the z-axis offset for drawing of tics in 3D plots.

Return Value

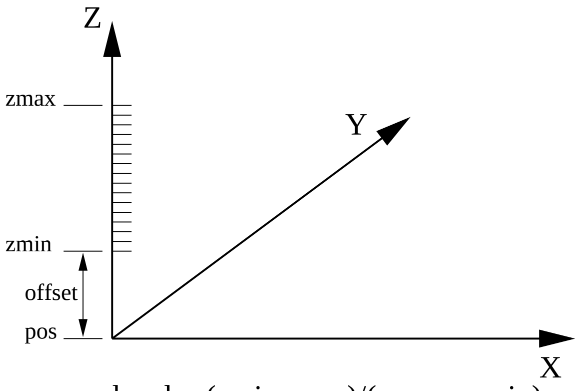
None.

Parameters

level The distance between the xy plane and the start of tic-marks on the z axis as a multiple of the full z range. This can be any non-negative number.

Description

This function specifies an offset between the xy plane and the start of z-axis tics-marks as a multiple of the full z range. By default the value for *level* is 0.5, so the z offset is a half of the z axis range. To place the xy-plane at the specified position `pos` on the z-axis, *level* shall equal (zmin-pos)/(zmax-zmin).



level = (zmin - pos)/(zmax - zmin)

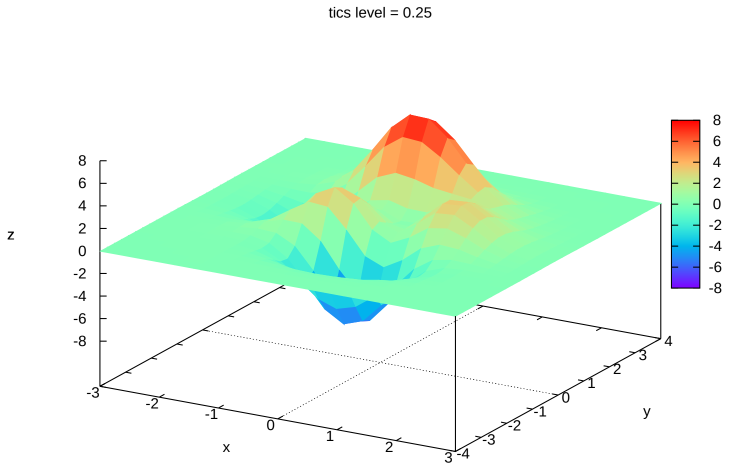
offset = level * (zmax - zmin)

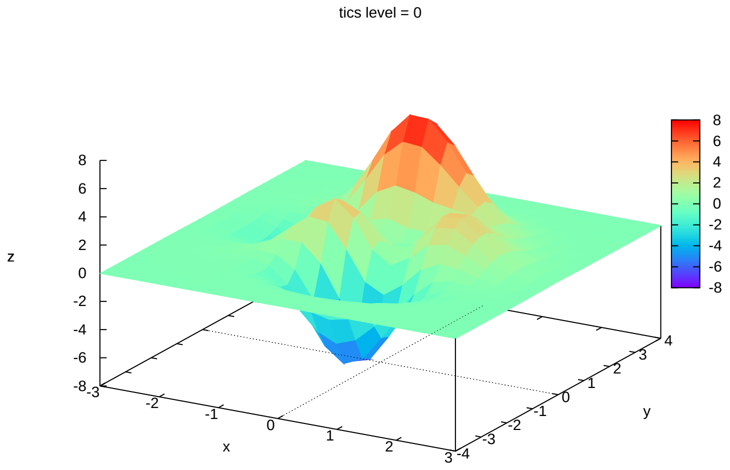
Example

Compare with the output for examples in **CPlot::data3D**() and **CPlot::data3DSurface**().

```
#include <math.h>
#include <chplot.h>

#define NUMX 20
#define NUMY 30
int main() {
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    int i,j;
    class CPlot plot;

    for(i=0; i<NUMX; i++) {
       x[i]= -3 + i*6.0/(NUMX-1); // linspace(x, -3, 3);
    }
    for(i=0; i<NUMY; i++) {
       y[i]= -4 + i*8.0/(NUMY-1); // linspace(y, -4, 4);
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[NUMY*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plot.data3DSurface(x, y, z, NUMX, NUMY);
    plot.ticsLevel(.25);
    plot.title("tics level = 0.25");
    plot.plotting();
    plot.ticsLevel(0);
    plot.title("tics level = 0");
    plot.plotting();
    return 0;
}
```

Output





CPlot::ticsLocation

Synopsis in Ch
#include <chplot.h>
void ticsLocation(int *axis*, **string_t** *location*)

Synopsis in C++
#include <chplot.h>
void ticsLocation(int *axis*, **char** * *location*)

Purpose
Specify the location of axis tic marks to be on the border or the axis.

Return Value

None.

Parameters

axis The *axis* parameter can take one of the following values:

- **PLOT_AXIS_X** Select the x axis only.
- **PLOT_AXIS_X2** Select the x2 axis only.
- **PLOT_AXIS_Y** Select the y axis only.
- **PLOT_AXIS_Y2** Select the y2 axis only.
- **PLOT_AXIS_XY** Select the x and y axes.

location Tic marks are placed on the plot border with `"border"` or on the axis itself with `"axis"`. By default, tic marks are on the border.

Description

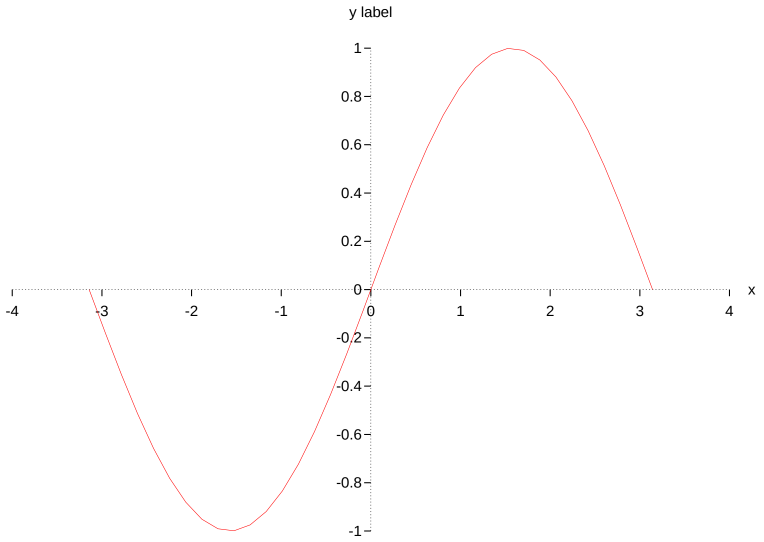
Specify the location of axis tic marks to be on the plot border or the axis itself.

Example

Compare with the output for examples in **CPlot**::**data2D**() and **CPlot**::**data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= -M_PI + i*2*M_PI/(NUM-1); // linspace(x, -M_PI, M_PI);
       y[i] = sin(x[i]); // Y-axis data
    }
    plot.data2DCurve(x, y, NUM);
    plot.ticsLocation(PLOT_AXIS_XY, "axis");
    plot.border(PLOT_BORDER_BOTTOM|PLOT_BORDER_LEFT, PLOT_OFF);
    plot.label(PLOT_AXIS_XY, NULL);
    plot.text("y label", PLOT_TEXT_CENTER, 0, 1.15, 0);
    plot.text("x", PLOT_TEXT_CENTER, 4.25, 0, 0);
    plot.margins(-1, -1, 2, -1); /* adjust top margin for y label */
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::tics(), **CPlot::ticsDirection**(), **CPlot::ticsFormat**(), **CPlot::ticsLabel**, **CPlot::ticsLevel**(), **CPlot::ticsLocation**, and **CPlot::ticsMirror**().

CPlot::ticsMirror

Synopsis

#include <**chplot.h**>
void ticsMirror(int *axis*, **int** *flag*)

Purpose

Enable or disable the display of axis tics on the opposite axis.

Return Value

None.

Parameters

axis The axis which labels are added to. This parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

flag This parameter can be set to:

PLOT_ON Enable drawing of tics for the specified axis.

PLOT_OFF Disable drawing of tics for the specified axis.

Description

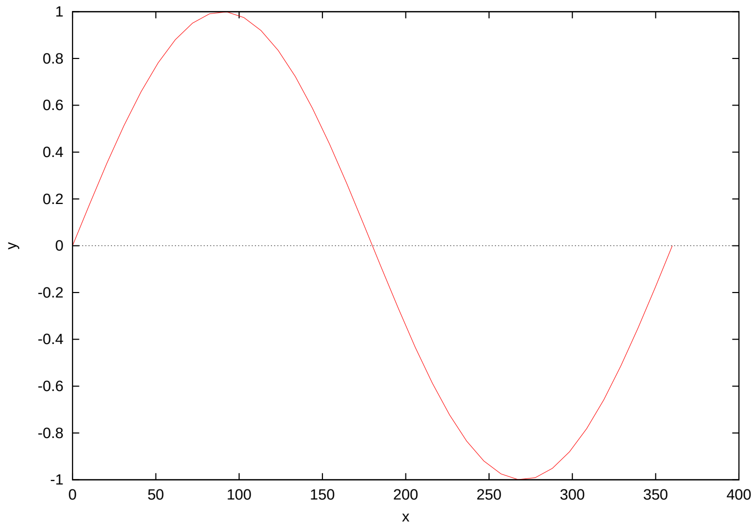
Enable or disable the display of tics on the opposite (mirror) axis. By default, on 2D plots tics on the opposite axis are not displayed. On 3D plots they are displayed

Example

Compare with output for example in **CPlot::border**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.data2DCurve(x, y, NUM);
    plot.border(PLOT_BORDER_ALL, PLOT_ON);
    plot.ticsMirror(PLOT_AXIS_XY, PLOT_ON);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::tics(), **CPlot::ticsDirection**(), **CPlot::ticsFormat**(), **CPlot::ticsLabel**(), **CPlot::ticsLevel**(), and **CPlot::ticsLocation**().

CPlot::ticsMonth

Synopsis

#include <chplot.h>

void ticsMonth(int *axis*);

Purpose

Set axis tic-marks to months.

Return Value

None.

Parameter

axis The axis to be changed. Valid values are:

- **PLOT_AXIS_X** Select the x axis only.
- **PLOT_AXIS_X2** Select the x2 axis only.
- **PLOT_AXIS_Y** Select the y axis only.
- **PLOT_AXIS_Y2** Select the y2 axis only.
- **PLOT_AXIS_Z** Select the z axis only.
- **PLOT_AXIS_XY** Select the x and y axes.
- **PLOT_AXIS_XYZ** Select the x, y, and z axes.

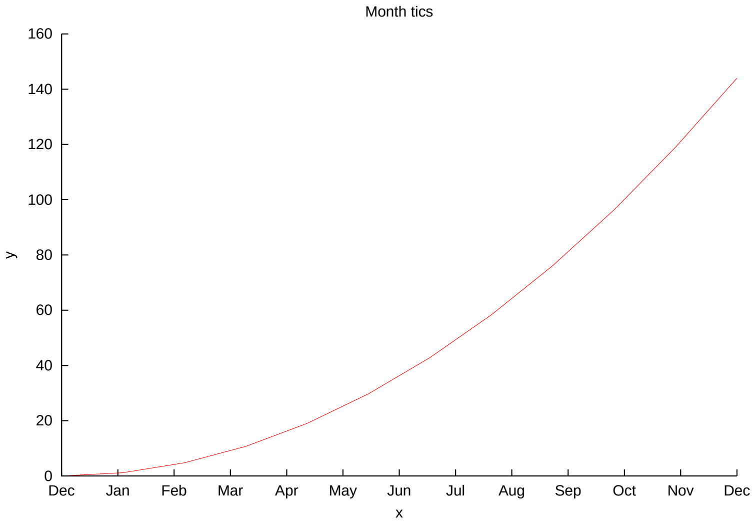
Description

Sets axis tic marks to months of the year (1=January, 12=December). Values greater than 12 are converted into the value of modulo 12.

Example

```
#include <math.h>
#include <chplot.h>

#define NUM 12
int main() {
    int i;
    double x[NUM], y[NUM];
    char *title="Month tics", *xlabel="x", *ylabel="y"; // Define labels.
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*12.0/(NUM-1); // linspace(x, 0, 12)
       y[i] = x[i]*x[i]; // y = x.*x
    }
    plot.ticsMonth(PLOT_AXIS_X);
    plot.title(title);
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



See Also
CPlot::ticsDay(), **CPlot::ticsLabel**().

CPlot::ticsPosition

Synopsis in Ch
#include <**chplot.h**>
void ticsPosition(**int** *axis*, **double** *position1*], ... /* [**double** *position2*], ...] */);

Synopsis in C++
#include <**chplot.h**>
void ticsLabel(**int** *axis*, **double** *position*);

Syntax in Ch
ticsPosition(*axis*, *position*)
ticsPosition(*axis*, *position1*, *position2*)
etc.

Syntax in C++
ticsPosition(*axis*, *position1*)
ticsPosition(*axis*, *position2*)

Purpose
Add tic-marks at the specified positions to an axis.

Return Value
None.

Parameters
axis The axis which tics are added to. This parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

- **PLOT_AXIS_X2** Select the x2 axis only.
- **PLOT_AXIS_Y** Select the y axis only.
- **PLOT_AXIS_Y2** Select the y2 axis only.
- **PLOT_AXIS_Z** Select the z axis only.
- **PLOT_AXIS_XY** Select the x and y axes.
- **PLOT_AXIS_XYZ** Select the x, y, and z axes.

position The position of the tic-mark on the axis.

Description
Add tic marks at the specified positions to an axis. The axis specification is followed by one or more position values of double precision floating point numbers. This function disables numerical labels for the specified axis. This function can be called multiple times to set tic positions for an axis. In this form, the tics do not need to be listed in numerical order.

Examples
See an example on page 183 using for **CPlot**:**ticsPosition**() for the x-axis for date.

See Also
CPlot::**ticsDirection**(), **CPlot**::**ticsFormat**(), **CPlot**::**ticsLevel**(), **CPlot**::**ticsLabel**(), **CPlot**::**ticsLocation**(), **CPlot**::**ticsRange**().

CPlot::ticsRange

Synopsis in Ch
#include <chplot.h>
void ticsRange(int *axis*, **double** *incr*, ... /* [**double** *start*], [**double** *end*] */);

Synopsis in C++
#include <chplot.h>
void ticsRange(int *axis*, **double** *incr*);
void ticsRange(int *axis*, **double** *incr*, **double** *start*]);
void ticsRange(int *axis*, **double** *incr*, **double** *start*], [**double** *end*]);

Syntax in Ch and C++
ticsRange(*axis*, *incr*) **ticsRange**(*axis*, *incr*, *start*)
ticsRange(*axis*, *incr*, *start*, *end*)

Purpose
Specify the range for a series of tics on an axis.

Return Value
None.

Parameters
axis The *axis* parameter can take one of the following values:

PLOT_AXIS_X Select the x axis only.

PLOT_AXIS_X2 Select the x2 axis only.

PLOT_AXIS_Y Select the y axis only.

PLOT_AXIS_Y2 Select the y2 axis only.

PLOT_AXIS_Z Select the z axis only.

PLOT_AXIS_XY Select the x and y axes.

PLOT_AXIS_XYZ Select the x, y, and z axes.

incr The increment between tic marks. By default or when *incr* is 0, the increment between tic marks is calculated internally.

start The starting value for tics.

end The end value for tics.

Description
The range for a series of tics on an axis can be explicitly specified with this function. Any previously specified labeled tic-marks are overridden. The implicit `start`, `incr`, `end` form specifies that a series of tics will be plotted on the axis between the values `start` and `end` with an increment of `incr`. If `end` is not given, it is assumed to be infinity. The increment may be negative. If neither `start` nor `end` is given, `start` is assumed to be negative infinity, `end` is assumed to be positive infinity, and the tics will be drawn at integral multiples of `incr`. If the axis is logarithmic specified by the member function **scaleType**(), the increment will be used as a multiplicative factor.

Example
See **CPlot**::**axisRange**().
See Also
CPlot::**axisRange**(), **CPlot**::**ticsPosition**(), **CPlot**::**ticsLabel**().

CPlot::title

Synopsis in Ch
#include <**chplot.h**>
void title(**string_t** *title*);

Synopsis in C++
#include <**chplot.h**>
void title(**char** * *title*);

Purpose
Set the plot title.

Return Value
None.

Parameters
title The plot title.

Description

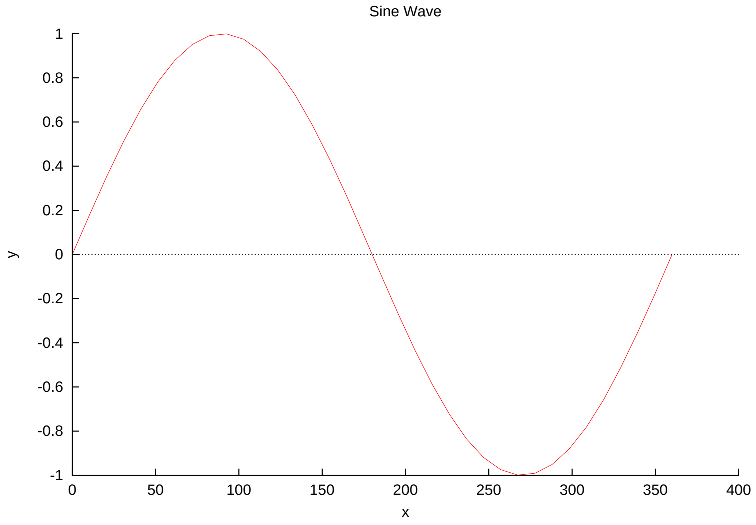
Add a title string to an existing plot variable. For no title, `NULL` can be specified. By default, no title is specified.

Example

Compare with the output for examples in **CPlot::data2D**() and **CPlot::data2DCurve**().

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    int i;
    double x[NUM], y[NUM];
    char *title="Sine Wave";                          // Define labels.
    class CPlot plot;

    for(i=0; i<NUM; i++) {
       x[i]= 0 + i*360.0/(NUM-1); // linspace(x, 0, 360)
       y[i] = sin(x[i]*M_PI/180); // Y-axis data
    }
    plot.title(title);
    plot.data2DCurve(x, y, NUM);
    plot.plotting();
    return 0;
}
```

Output



See Also

CPlot::label(), **CPlot::getLabel**(), **CPlot::getTitle**().

fplotxy

Synopsis in Ch
#include <chplot.h>
int fplotxy(**double** *(*func)(***double** *x)*, **double** *x0*, **double** *xf*, ...
/* [**int** *num*, [**char** * *title*, **char** * *xlabel*, **char** * *ylabel*], [**class CPlot** **pl*]] */);

Synopsis in C++
#include <chplot.h>
int fplotxy(**double** *(*func)(***double** *x)*, **double** *x0*, **double** *xf*, **int** *num*);
int fplotxy(**double** *(*func)(***double** *x)*, **double** *x0*, **double** *xf*, **int** *num*,
char * *title*, **char** * *xlabel*, **char** * *ylabel*);
int fplotxy(**double** *(*func)(***double** *x)*, **double** *x0*, **double** *xf*, **int** *num*,
char * *title*, **char** * *xlabel*, **char** * *ylabel*, **class CPlot** **pl*);

Syntax in Ch
fplotxy(*func*, *x0*, *xf*)
fplotxy(*func*, *x0*, *xf*, *num*, &*plot*)

Syntax in Ch and C++
fplotxy(*func*, *x0*, *xf*, *num*)
fplotxy(*func*, *x0*, *xf*, *num*, *title*, *xlabel*, *ylabel*)
fplotxy(*func*, *x0*, *xf*, *num*, *title*, *xlabel*, *ylabel*, &*plot*)

Purpose
Plot a 2D function of *x* in the range $x0 \leq x \leq xf$.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
func A pointer to a function that takes a `double` as an argument and returns a `double`.

x0 The lower bound of the range to be plotted.

xf The upper bound of the range to be plotted.

num The number of points to be plotted. The default is 100.

title The title of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

pl A pointer to an instance of the **CPlot** class.

Description
Plot a 2D function of *x* in the range $x0 \leq x \leq xf$. The function to be plotted, *func*, is specified as a pointer to a function that takes a *double* as an argument and returns a *double*. The arguments *x0* and *xf* are the

end-points of the range to be plotted. The optional argument *num* specifies how many points in the range are to be plotted. The number of points plotted are evenly spaced in the range. By default, 100 points are plotted. The *title*, *xlabel*, and *ylabel* for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting**() member function. If a previously initialized *plot* variable is passed, it will be re-initialized with the function parameters. If no pointer or a NULL pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function.

The following code segment

```
class CPlot plot;
fplotxy(func, x0, xf, n, "title", "xlabel", "ylabel", &plot);
```

is equivalent to

```
class CPlot plot;
plot.func2D(x0, xf, n, func);
plot.title("title");
plot.label(PLOT_AXIS_X, "xlabel");
plot.label(PLOT_AXIS_Y, "ylabel");
```

Example

```
#include<math.h>
#include<chplot.h>

#define N 100

double omega;
double func(double x) {
   double y;

   y = sin(omega*x);
   return y;
}

int main() {
   double x0, xf;
   CPlot plot;

   x0 = 0;
   xf = 2*M_PI;
   fplotxy(sin, x0, xf, N);
   fplotxy(sin, x0, xf, N, "sin(wx)", "x", "sin(x)");
   omega = 2;
   fplotxy(func, x0, xf, N, "sin(wx)", "x", "sin(wx)", &plot);
   plot.plotting();
}
```

See Also

CPlot, **fplotxyz**(), **plotxy**(), **plotxyf**(), **plotxyz**(), **plotxyzf**().

fplotxyz

Synopsis in Ch
#include <chplot.h>
int fplotxyz(**double** *(*func)(***double** *x*, **double** *y)*, **double** *x0*, **double** *xf*, **double** *y0*, **double** *yf*, ...
/* [**int** *x_num*, **int** *y_num*, /* [**char** * *title*, **char** * *xlabel*, **char** * *ylabel*], **char** * *zlabel*],
[**class CPlot** **pl*]]*/);

Synopsis in C++
#include <chplot.h>
double *x0*, **double** *xf*, **double** *y0*, **double** *yf*);
int fplotxyz(**double** *(*func)(***double** *x*, **double** *y)*, **double** *x0*, **double** *xf*, **double** *y0*, **double** *yf*,
int *x_num*, **int** *y_num*);
int fplotxyz(**double** *(*func)(***double** *x*, **double** *y)*, **double** *x0*, **double** *xf*, **double** *y0*, **double** *yf*,
int *x_num*, **int** *y_num*, **char** * *title*, **char** * *xlabel*, **char** * *ylabel*, **char** * *zlabel*);
int fplotxyz(**double** *(*func)(***double** *x*, **double** *y)*, **double** *x0*, **double** *xf*, **double** *y0*, **double** *yf*,
int *x_num*, **int** *y_num*, **char** * *title*, **char** * *xlabel*, **char** * *ylabel*, **char** * *zlabel*, **class CPlot** **pl*);

Syntax in Ch and C++
fplotxyz(*func*, *x0*, *xf*, *y0*, *yf*)
fplotxyz(*func*, *x0*, *xf*, *y0*, *yf*, *x_num*, *y_num*, &*plot*)

Syntax in Ch and C++
fplotxyz(*func*, *x0*, *xf*, *y0*, *yf*, *x_num*, *y_num*)
fplotxyz(*func*, *x0*, *xf*, *y0*, *yf*, *x_num*, *y_num*, *title*, *xlabel*, *ylabel*, *zlabel*)
fplotxyz(*func*, *x0*, *xf*, *y0*, *yf*, *x_num*, *y_num*, *title*, *xlabel*, *ylabel*, *zlabel*, &*plot*)

Purpose
Plot a 3D function of *x* and *y* in the range $x0 \le x \le xf$ and $y0 \le y \le yf$.

Return Value
This function returns 0 on success and -1 on failure.

Parameters
func A pointer to a function that takes two **double** arguments and returns a **double**.

x0 The lower bound of the x range to be plotted.

xf The upper bound of the x range to be plotted.

y0 The lower bound of the y range to be plotted.

yf The upper bound of the y range to be plotted.

x_num The number of points to be plotted. The default is 25.

y_num The number of points to be plotted. The default is 25.

title The title of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

zlabel The z-axis label.

pl A pointer to an instance of the **CPlot** class.

Description

Plot a 3D function of *x* and *y* in the range $x0 \leq x \leq xf$ and $y0 \leq y \leq yf$. The function to be plotted, *func*, is specified as a pointer to a function that takes two **double** arguments and returns a **double**. *x0* and *xf* are the end-points of the *x* range to be plotted. *y0* and *yf* are the end-points of the *y* range to be plotted. The optional arguments *x_num* and *y_num* specify how many points in the *x* and *y* ranges are to be plotted. The number of points plotted are evenly spaced in the ranges. By default, *x_num* and *y_num* are 25. The *title*, *xlabel*, *ylabel*, and *zlabel* for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting**() member function. If a previously initialized **CPlot** variable is passed, it will be re-initialized with the function parameters. If no pointer or a NULL pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function. This function can only be used to plot 3D grid or scatter data, it cannot be used to plot 3D paths.

The following code segment

```
class CPlot plot;
fplotxyz(func, x0, xf, y0, yf, nx, ny, "title", "xlabel", "ylabel",
         "zlabel", \&plot);
```

is equivalent to

```
class CPlot plot;
plot.func2D(x0, xf, y0, yf, nx, ny, func);
plot.title("title");
plot.label(PLOT_AXIS_X, "xlabel");
plot.label(PLOT_AXIS_Y, "ylabel");
plot.label(PLOT_AXIS_Z, "zlabel");
```

Example

```
#include <math.h>
#include <chplot.h>

double func(double x, double y) {               // function to be plotted
    return 3*(1-x)*(1-x)*exp(-(x*x) - (y+1)*(y+1) )
           - 10*(x/5 - x*x*x - pow(y,5))*exp(-x*x-y*y)
           - 1/3*exp(-(x+1)*(x+1) - y*y);
}
int main() {
    const char *title="fplotxyz()",                    // Define labels.
               *xlabel="X-axis",
               *ylabel="Y-axis",
               *zlabel="Z-axis";
    double x0 = -3, xf = 3, y0 = -4, yf = 4;
    int x_num = 20, y_num = 50;
    class CPlot plot;

    fplotxyz(func, x0, xf, y0, yf, x_num, y_num);
    fplotxyz(func, x0, xf, y0, yf, x_num, y_num, title, xlabel, ylabel, zlabel);
    fplotxyz(func, x0, xf, y0, yf, x_num, y_num, title, xlabel, ylabel, zlabel, &plot);
    plot.plotting();
}
```

See Also

CPlot, **fplotxy**(), **plotxy**(), **plotxyf**(), **plotxyz**(), **plotxyzf**().

plotxy

Synopsis in Ch
#include <chplot.h>
int plotxy(**double** *x*[&], **array double** &*y*, ...
/* [int n] [**char** * *title*, **char** **xlabel*, **char** **ylabel*],
[**class CPlot** **pl*] */);

Synopsis in C++
#include <chplot.h>
int plotxy(**double** *x*[], **double** *y*[], int n);
int plotxy(**double** *x*[], **double** *y*[], int n, **char** * *title*, **char** **xlabel*, **char** **ylabel*);
int plotxy(**double** *x*[], **double** *y*[], int n, **char** * *title*, **char** **xlabel*, **char** **ylabel*, **class CPlot** **pl*]);

Syntax in Ch
plotxy(*x*, *y*)
plotxy(*x*, *y*, *title*, *xlabel*, *ylabel*)
plotxy(*x*, *y*, *title*, *xlabel*, *ylabel*, &*plot*)

Syntax in Ch and C++
plotxy(*x*, *y*, n)
plotxy(*x*, *y*, n, *title*, *xlabel*, *ylabel*)
plotxy(*x*, *y*, n, *title*, *xlabel*, *ylabel*, &*plot*)

Purpose
Plot a 2D data set or initialize an instance of the **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

x A one-dimensional array of size n. The value of each element is used for the x-axis of the plot.

y A m x n dimensional array containing m curves, each of which is plotted against x.

n An integer for the number of elements of array x.

title The *title* of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

pl A pointer to an instance of the **CPlot** class.

Description
The arrays *x* and *y* can be of any supported data type of real numbers. Conversion of the data to **double** type is performed internally. The argument n is the number of elements for array x. The *title*, *xlabel*, and *ylabel* for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this

function. If a non-NULL pointer is passed, it will be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting**() member function. If a previously initialized **CPlot** variable is passed, it will be re-initialized with the function parameters. If no pointer or a NULL pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function.

The following code segment

```
class CPlot plot;
plotxy(x, y, n, "title", "xlabel", "ylabel", &plot);
```

is equivalent to

```
class CPlot plot;
plot.data2DCurve(x, y, n);
plot.title("title");
plot.label(PLOT_AXIS_X, "xlabel");
plot.label(PLOT_AXIS_Y, "ylabel");
```

Example

```
#include <math.h>
#include <chplot.h>

#define NUM 36
int main() {
    double x[NUM], x2[NUM], y[NUM];
    class CPlot plot;
    /*
    int i;
    for(i=0; i< NUM; i++) {
      x[i] = i*10;
      y[i] = sin(x[i]*M_PI/180);
    }
    */
    lindata(0, 360, x, NUM);
    lindata(0, 2*M_PI, x2, NUM);
    funcarray(sin, y, x2, NUM);
    plotxy(x, y, NUM);
    plotxy(x, y, NUM, "title", "xlablel", "ylabel");
    plotxy(x, y, NUM, "title", "xlablel", "ylabel", &plot);
    plot.plotting();
    return 0;
}
```

See Also

CPlot, **CPlot**::**data2D**(), **CPlot**::**data2DCurve**(), **fplotxy**(), **fplotxyz**(), **plotxyf**(), **plotxyz**(), **plotxyzf**().

plotxyf

Synopsis in Ch
#include <chplot.h>
int plotxyf(**string_t** *file*, ... /* [**char** * *title*, **char** * *xlabel*, **char** * *ylabel*], [**class CPlot** **pl*] */);

Synopsis in C++
#include <chplot.h>
int plotxyf(**string_t** *file*);
int plotxyf(**string_t** *file*, **char** * *title*, **char** * *xlabel*, **char** * *ylabel*);
int plotxyf(**string_t** *file*, **char** * *title*, **char** * *xlabel*, **char** * *ylabel*, **class CPlot** **pl*] */);

Syntax in Ch and C++
plotxyf(*file*)
plotxyf(*file*, *title*, *xlabel*, *ylabel*)
plotxyf(*file*, *title*, *xlabel*, *ylabel*, *&plot*)

Purpose
Plot 2D data from a file or initialize an instance of the **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

file The file containing the data to be plotted.

title The title of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

pl A pointer to an instance of the **CPlot** class.

Description
Plot 2D data from a file or initialize a **CPlot** variable. The data file should be formatted with each data point on a separate line. 2D data are specified by two values per point. The *title*, *xlabel*, and *ylabel*, for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting** member function. If a previously initialized **CPlot** variable is passed, it will be re-initialized with the function parameters. If no pointer or a `NULL` pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function. An empty line in the data file causes a break in the plot. Multiple curves can be plotted in this manner, however, the plot style will be the same for all curves.

The following code segment

```
class CPlot plot;
plotxyf("datafile", "title", "xlabel", "ylabel", &plot);
```

is equivalent to

```
class CPlot plot;
plot.dataFile("datafile");
plot.title("title");
plot.label(PLOT_AXIS_X, "xlabel");
plot.label(PLOT_AXIS_Y, "ylabel");
```

Example

```
#include <stdio.h>
#include <chplot.h>
#include <math.h>

int main() {
    char *filename;
    int i;
    class CPlot plot;
    FILE *out;

    filename = tmpnam(NULL);               //Create temporary file.
    out=fopen (filename ,"w");              //Write data to file.
    for(i=0;i<=359;i++) fprintf(out,"%i %f \n",i,sin(i*M_PI/180));
    fclose(out);
    plotxyf(filename);
    plotxyf(filename, "title", "xlabel", "ylabel");
    plotxyf(filename, "title", "xlabel", "ylabel", &plot);
    plot.plotting();
    remove(filename);
    return 0;
}
```

See Also

CPlot, **CPlot**::**dataFile**, **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyzf**().

plotxyz

Synopsis in Ch
#include <chplot.h>
int plotxyz(**double** *x*[&], **double** *y*[&], **array double** &*z*, ... /* [int n] [int nx, int ny]
/* [**char** * *title*, **char** **xlabel*, **char** **ylabel*], [**class CPlot** **pl*] */);

Synopsis in C++
#include <chplot.h>
int plotxy(**double** *x*[], **double** *y*[], **double** *z*[], **int** *n*);
int plotxy(**double** *x*[], **double** *y*[], **double** *z*[], **int** *n*, **char** * *title*, **char** **xlabel*, **char** **ylabel*, **char** **zlabel*);
int plotxy(**double** *x*[], **double** *y*[], **double** *z*[], **int** *n*, **char** * *title*, **char** **xlabel*, **char** **ylabel*, **char** **zlabel*,
class CPlot **pl*]);
int plotxy(**double** *x*[], **double** *y*[], **double** *z*[], **int** *nx*, **int** *ny*);
int plotxy(**double** *x*[], **double** *y*[], **double** *z*[], **int** *nx*, **int** *ny*, **char** * *title*, **char** **xlabel*, **char** **ylabel*,
char **zlabel*);
int plotxy(**double** *x*[], **double** *y*[], **double** *z*[], **int** *nx*, **int** *ny*, **char** * *title*, **char** **xlabel*, **char** **ylabel*,
char **zlabel*, **class CPlot** **pl*]);

Syntax in Ch
plotxyz(*x*, *y*, *z*)
plotxyz(*x*, *y*, *z*, *title*, *xlabel*, *ylabel*, *zlabel*)
plotxyz(*x*, *y*, *z*, *title*, *xlabel*, *ylabel*, *zlabel*, &*plot*)

Syntax in Ch and C++
plotxyz(*x*, *y*, *z*, n)
plotxyz(*x*, *y*, *z*, n, *title*, *xlabel*, *ylabel*, *zlabel*)
plotxyz(*x*, *y*, *z*, n, *title*, *xlabel*, *ylabel*, *zlabel*, &*plot*)
plotxyz(*x*, *y*, *z*, nx, ny)
plotxyz(*x*, *y*, *z*, nx, ny, *title*, *xlabel*, *ylabel*, *zlabel*)
plotxyz(*x*, *y*, *z*, nx, ny, *title*, *xlabel*, *ylabel*, *zlabel*, &*plot*)

Purpose
Plot a 3D data set or initialize an instance of the **CPlot** class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

x A one-dimensional array of size n_x. The value of each element is used for the x-axis of the plot.

y A one-dimensional array of size n_y. The value of each element is used for the y-axis of the plot.

z If the data are for a 3D curve, z is a m X n_z dimensional array, and $n_x = n_y = n_z$. If the data are for a 3D surface or grid, z is a m x n_z dimensional array, and $n_z = n_x \cdot n_y$.

n The number of data points for a 3D curve.

nx The number of data points in the x-coordinates for a 3D surface.

ny The number of data points in the y-coordinates for a 3D surface.

title The title of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

zlabel The y-axis label.

pl A pointer to an instance of the **CPlot** class.

Description

Plot a 3D data set or initialize a **CPlot** variable. For Cartesian data, *x* is a one-dimensional array of size n_x and *y* is a one-dimensional array of size n_y. *z* can be of two different dimensions depending on what type of data is to be plotted. If the data is for a 3D curve, *z* is a m x n_z dimensional array, and $n_x = n_y = n_z$ for the the optional argument `n`. If the data is for a 3D surface or grid, *z* is a m x n_z dimensional array, and $n_z = n_x \cdot n_y$ with optional arguments `nx` and `ny`. For cylindrical or spherical data *x* is a one dimensional array of size n_x (representing θ), *y* is a one dimensional array of size n_y (representing z or ϕ), and *z* is a m x n_z dimensional array (representing r). In all cases these data arrays can be of any supported data type. Conversion of the data to **double** type is performed internally. The *title*, *xlabel*, *ylabel*, and *zlabel* for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting** member function. If a previously initialized **CPlot** variable is passed, it will be re-initialized with the function parameters. If no pointer or a NULL pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function.

The following code segment for plotting a 3D curve

```
    class CPlot plot;
    plotxyz(x, y, z, n, "title", "xlabel", "ylabel", "zlabel", &plot);
```

is equivalent to

```
    class CPlot plot;
    plot.data3DCurve(x, y, z, n);
    plot.title("title");
    plot.label(PLOT_AXIS_X, "xlabel");
    plot.label(PLOT_AXIS_Y, "ylabel");
    plot.label(PLOT_AXIS_Z, "zlabel");
```

The following code segment for plotting a 3D surface

```
    class CPlot plot;
    plotxyz(x, y, z, nx, ny);
```

is equivalent to

```
    class CPlot plot;
    plot.data3DSurface(x, y, z, nx, ny);
```

Example

```
#include <math.h>
#include <chplot.h>

#define N 360
#define NUMX 20
#define NUMY 30
int main() {
    double x1[N], y1[N], z1[N];
    double x[NUMX], y[NUMY], z[NUMX*NUMY];
    int i,j;
    class CPlot plot, plot2;

    for(i=0; i<N; i++) {
        x1[i] = i;
        y1[i] = i;
        z1[i] = cos(x1[i]*M_PI/180);
    }
    plotxyz(x1, y1, z1, N);
    plotxyz(x1, y1, z1, N, "Title", "xlabel", "ylabel", "zlabel");
    plotxyz(x1, y1, z1, N, "Title", "xlabel", "ylabel", "zlabel", &plot);
    plot.plotting();

    for(i=0; i<NUMX; i++) {
      x[i] = -3 + i*6/19.0;  // linspace(x, -3, 3)
    }
    for(i=0; i<NUMY; i++) {
      y[i] = -4 + i*8/29.0;  // linspace(y, -4, 4)
    }
    for(i=0; i<NUMX; i++) {
        for(j=0; j<NUMY; j++) {
            z[NUMY*i+j] = 3*(1-x[i])*(1-x[i])*exp(-(x[i]*x[i])-(y[j]+1)*(y[j]+1))
            - 10*(x[i]/5 - x[i]*x[i]*x[i]-pow(y[j],5))*exp(-x[i]*x[i]-y[j]*y[j])
            - 1/3*exp(-(x[i]+1)*(x[i]+1)-y[j]*y[j]);
        }
    }
    plotxyz(x, y, z, NUMX, NUMY);
    plotxyz(x, y, z, NUMX, NUMY, "Title", "xlabel", "ylabel", "zlabel");
    plotxyz(x, y, z, NUMX, NUMY, "Title", "xlabel", "ylabel", "zlabel", &plot2);
    plot2.plotting();
}
```

See Also

CPlot, **CPlot**::**data3D**(), **CPlot**::**data3DCurve**(), **CPlot**::**data3DSurface**(), **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyf**(), **plotxyzf**().

plotxyzf

Synopsis
#include <chplot.h>
int plotxyzf(**string_t** *file*, ... /* [**char** * *title*, **char** * *xlabel*, **char** * *ylabel*], **char** * *zlabel*],
[**class CPlot** **pl*] */);

Synopsis in C++
#include <chplot.h>
int plotxyzf(**string_t** *file*);
int plotxyzf(**string_t** *file*, **char** * *title*, **char** * *xlabel*, **char** * *ylabel*, **char** * *zlabel*); **int plotxyzf**(**string_t** *file*, **char** * *title*, **char** * *xlabel*, **char** * *ylabel*, **char** * *zlabel*, **class CPlot** **pl*] */);

Syntax in Ch and C++
plotxyzf(*file*)
plotxyzf(*file*, *title*, *xlabel*, *ylabel*, *zlabel*)
plotxyzf(*file*, *title*, *xlabel*, *ylabel*, *zlabel*, &*plot*)

Purpose
Plot 3D data from a file or initialize an instance of the **CPlot**class.

Return Value
This function returns 0 on success and -1 on failure.

Parameters

file The file containing the data to be plotted.

title The title of the plot.

xlabel The x-axis label.

ylabel The y-axis label.

zlabel The z-axis label.

pl A pointer to an instance of the **CPlot** class.

Description
Plot 3D data from a file or initialize a **CPlot** variable. The data file should be formatted with each data point on a separate line. 3D data is specified by three values per data point. For a 3D grid or surface data, each row is separated in the data file by a blank line. For example, a 3 x 3 grid would be represented as follows:

```
x1  y1  z1
x1  y2  z2
x1  y3  z3

x2  y1  z4
x2  y2  z5
x2  y3  z6

x3  y1  z7
x3  y2  z8
x3  y3  z9
```

This function can only be used to plot Cartesian grid data. The *title*, *xlabel*, *ylabel*, and *zlabel* for the plot can also optionally be specified. A pointer to a plot structure can also be passed to this function. If a non-NULL pointer is passed, it will be be initialized with the function parameters. The plot can then be displayed using the **CPlot**::**plotting** member function. If a previously initialized **CPlot** variable is passed, it will be re-initialized with the function parameters. If no pointer or a NULL pointer is passed, an internal **CPlot** variable will be used and the plot will be displayed without calling the **CPlot**::**plotting**() member function. Two empty lines in the data file will cause a break in the plot. Multiple curves or surfaces can be plotted in this manner however, the plot style will be the same for all curves or surfaces.

The following code segment

```
class CPlot plot;
plotxyzf("datafile", "title", "xlabel", "ylabel", "zlabel", &plot);
```

is equivalent to

```
class CPlot plot;
plot.dimension(3);
plot.dataFile("datafile");
plot.title("title");
plot.label(PLOT_AXIS_X, "xlabel");
plot.label(PLOT_AXIS_Y, "ylabel");
plot.label(PLOT_AXIS_Z, "zlabel");
```

Example

```
#include <stdio.h>
#include <chplot.h>
#include <math.h>

int main() {
    char *filename;
    int i;
    class CPlot plot;
    FILE *out;

    filename = tmpnam(NULL);               //Create temporary file.
    out=fopen(filename ,"w");              //Write data to file.
    for(i=0;i<=359;i++) fprintf(out,"%i %f %f\n",i,sin(i*M_PI/180),cos(i*M_PI/180));
    fclose(out);
    plotxyzf(filename);
    plotxyzf(filename, "title", "xlabel", "ylabel", "zlabel");
    plotxyzf(filename, "title", "xlabel", "ylabel", "zlabel", &plot);
    plot.plotting();
    remove(filename);
    return 0;
}
```

See Also

CPlot, **CPlot**::**dataFile**(), **fplotxy**(), **fplotxyz**(), **plotxy**(), **plotxyz**(), **plotxyz**().

Chapter 5

Appendix A: Differences between Ch and C++ for Graphical Plotting

The **CPlot** class can be used to produce two dimensional (2D) and three dimensional (3D) plots in Ch. The SIGL also contains the **CPlot** class which can be compiled using a C++ compiler . Most member functions of the plotting class are the same for both Ch and C++. However, the plotting class in Ch takes advantage of salient features of computational array and other its related features. It is more convenient to use in Ch. For example, the actual array size in a function argument can be obtained. Therefore, the parameter for array sizes of many member functions are optional in Ch. Arrays of different data type can be passed to an array of reference in functions, which is not available in C++. Member functions of the **CPlot** class available only in Ch, not in SIGL are listed below.

Function	Description
CPlot::**data2D**()	Add one or more 2D data sets to an instance of the **CPlot** class.
CPlot::**data3D**()	Add one or more 3D data sets to an instance of the **CPlot** class.

Member functions **CPlot**::**data2D**() can be replaced by **CPlot**::**data2DCurve**(), whereas member function **CPlot**::**data3D**() can be replaced by **CPlot**::**data3DCurve**() and **CPlot**::**data3DSurface**().

Chapter 6

Appendix B: Source Code for the Figure on the Cover Page

```
#include <math.h>
#include <chplot.h>

#define POINTS 50            // number of data points for each curve
int main() {
    int i;
    double t[POINTS], b0[POINTS], b1[POINTS], b2[POINTS], b3[POINTS];
    class CPlot plot;

    for(i=0; i< POINTS; i++) {
      t[i] = 1+i*(10.0-1)/(POINTS-1);
      b0[i] = j0(t[i]);
      b1[i] = j1(t[i]);
      b2[i] = jn(2, t[i]);
      b3[i] = jn(3, t[i]);
    }
    plot.label(PLOT_AXIS_X,"t");              // x-label
    plot.label(PLOT_AXIS_Y,"Bessel functions"); // y-label
    plot.data2DCurve(t, b0);                    // plotting data
    plot.data2DCurve(t, b1);                    // plotting data
    plot.data2DCurve(t, b2);                    // plotting data
    plot.data2DCurve(t, b3);                    // plotting data
    plot.legend("j0(t)", 0);                    // legend for j0
    plot.legend("j1(t)", 1);                    // legend for j1
    plot.legend("j2(t)", 2);                    // legend for j2
    plot.legend("j3(t)", 3);                    // legend for j3
    plot.plotting();
    return 0;
}
```

Chapter 7

Porting Code to the Latest Version

7.1 Porting Code to SIGL Version 2.5

1. Changed

   ```
   CPlot::func2D(double (*func)(double x, void *param), void *param,
                 double x0, double xf, int n);
   CPlot::func3D(double (*func)(double x, double y, void *param),
                 void *param, double x0, double xf, double y0,
                 double yf, int nx, int ny);
   ```

   to

   ```
   CPlot::funcp2D(double x0, double xf, int n,
                  double (*func)(double x, void *param), void *param);
   CPlot::funcp3D(double x0, double xf, double y0, double yf,
                  int nx, int ny,
                  double (*func)(double x, double y, void *param),
                  void *param);
   ```

2. Changed

   ```
   CPlot::origin(double x, double y);
   ```

   to

   ```
   CPlot::boundingBoxOrigin(double x, double y);
   ```

3. Changed

   ```
   CPlot::grid(int flag);
   CPlot::grid(int flag, int type);
   ```

   to

```
    CPlot::grid(int flag);
    CPlot::grid(int flag, char *option);
```

Removed

```
    PLOT_GRID_POLAR
    PLOT_GRID_RECTANGULAR
```

Change

```
    plot.grid(PLOT_ON, PLOT_GRID_POLAR);
    plot.polarPlot(PLOT_ANGLE_DEG);
```

to

```
    plot.grid(PLOT_ON);
 or plot.grid(PLOT_ON, "polar");
 or lot.grid(PLOT_ON, "polar 30"); // the interval of radials is 30 degrees
    plot.polarPlot(PLOT_ANGLE_DEG);
```

4. Changed

```
    CPlot::arrow(double x_head, double y_head, double z_head,
                 double x_tail, double y_tail, double z_tail,
                 int linetype, int linewidth);
```

to

```
    CPlot::arrow(double x_head, double y_head, double z_head,
                 double x_tail, double y_tail, double z_tail);
    CPlot::arrow(double x_head, double y_head, double z_head,
                 double x_tail, double y_tail, double z_tail, option);
```

Change

```
    plot.arrow(x1, y1, z1, x2, y2, z2, 1, 3);
```

to

```
    char option[64];
    sprintf(option, "linetype 1 linewidth 3");
    plot.arrow(x1, y1, z1, x2, y2, z2, option);
```

5. `CPlot::axisRange(int axis, double minx, double max, double incr);`

is obsolete. Use

```
    CPlot::axisRange(int axis, double minx, double max);
    CPlot::ticsRange(int axis, incr);
```

Index

arrow(), *see* CPlot, 49, *see* CPlot, *see* CPlot
autoScale(), *see* CPlot, 49, *see* CPlot
axes(), 50, *see* CPlot
axis(), *see* CPlot, 49, *see* CPlot
axisRange(), *see* CPlot, 50, *see* CPlot, *see* CPlot

barSizd(), 50
barSize(), *see* CPlot
border(), *see* CPlot, 50, *see* CPlot
borderOffsets(), *see* CPlot, 50, *see* CPlot
boundingBoxOrigin(), 50, *see* CPlot
boxBorder(), 50, *see* CPlot
boxFill(), 50, *see* CPlot
boxWidth(), 50, *see* CPlot

changeViewAngle(), *see* CPlot, 50, *see* CPlot
circle(), *see* CPlot, 50, *see* CPlot
colorBox(), 50, *see* CPlot
contourLabel(), *see* CPlot, 50, *see* CPlot
contourLevels(), *see* CPlot, 50, *see* CPlot
contourMode(), *see* CPlot, 50, *see* CPlot
coordSystem(), *see* CPlot, 50, *see* CPlot
copyright, i
CPlot, **49**
- ˜CPlot, 49
- arrow(), **16**, 49, **55**
- autoScale(), 49, **59**
- axes(), 50, **66**
- axis(), **15**, 49, **60**
- axisRange(), **15**, 50, **62**
- barSize(), 50, **68**
- border(), **15**, 50, **69**
- borderOffsets(), 50, **71**
- boundingBoxOrigin(), 50
- boxBorder(), 50, **72**
- boxFill(), 50, **73**
- boxWidth(), 50, **76**
- changeViewAngle(), 50, **77**
- circle(), 50, **79**
- colorBox(), 50, **81**
- contourLabel(), **38**, 50, **83**
- contourLevel(), **38**
- contourLevels(), 50, **85**
- contourMode(), **38**, 50, **88**
- coordSystem(), **39**, 50, **91**
- CPlot(), 49
- data(), 50, **96**
- data2D(), 50, 251
- data2DCurve(), **8**, 50, **100**
- data2DSurface(), **11**
- data3D(), 50, 251
- data3DCurve(), 50, **101**
- data3DSurface(), 50, **104**
- dataFile(), **11**, 50, **106**
- dataSetNum(), 50, **109**
- deleteData(), 50, **110**
- deletePlots(), 50, **111**
- dimension(), **13**, 50, **112**
- displayTime(), 50, **112**
- enhanceText(), 50, **113**
- func2D(), 50, **118**
- func3D(), 50, **119**
- funcp2D(), 50, **121**
- funcp3D(), 50, **122**
- getLabel(), 50, **124**
- getOutputType(), 50, **125**
- getSubplot(), 50, **126**
- getTitle(), 50, **130**
- grid(), 50, **131**
- isUsed(), 50, **134**
- label(), **13**, 50, **134**
- legend(), **16**, 50, **136**
- legendLocation(), **16**, 50, **138**
- legendOption(), 50, **139**
- line(), 50, **142**
- lineType(), **28**, 50, **145**
- margins(), 50, **149**
- origin(), 51, **150**
- outputType(), **22**, 51, **151**
- plotting(), **8**, 51, **189**

plotType(), **28**, 51, **160**
point(), 51, **189**
pointType(), **28**, 51, **191**
polarPlot(), **32**, 51, **194**
polygon(), 51, **196**
rectangle(), 51, **200**
removeHiddenLine(), 51, **202**
scaleType(), 51, **205**
showMesh(), 51, **206**
size(), 51, **209**
size3D(), 51, **210**
sizeOutput(), 51, **211**
sizeRatio(), **32**, 51, **212**
smooth(), 51, **214**
subplot(), **22**, 51, **215**
text(), **16**, 51, **216**
tics(), 51, **217**
ticsDay(), 51, **218**
ticsDirection(), 51, **220**
ticsFormat(), 51, **221**
ticsLabel(), 51, **223**
ticsLevel(), 51, **225**
ticsLocation(), 51, **227**
ticsMirror(), 51, **229**
ticsMonth(), 51, **230**
ticsPosition(), 51, **232**
ticsRange(), 51, **233**
title(), **13**, 51, **234**

data(), 50, *see* CPlot
data2D(), *see* CPlot, 50, 251
data2DCurve(), 50, *see* CPlot
data3D(), *see* CPlot, 50, 251
data3DCurve(), 50, *see* CPlot
data3DSurface(), 50, *see* CPlot
dataFile(), *see* CPlot, 50, *see* CPlot
dataSetNum(), 50, *see* CPlot
deleteData(), 50, *see* CPlot
deletePlots(), *see* CPlot, 50, *see* CPlot
dimension(), *see* CPlot, 50, *see* CPlot
displayTime(), *see* CPlot, 50, *see* CPlot
distribution, 47
DYLD_FORCE_FLAT_NAMESPACE, 7
DYLD_INSERT_LIBRARIES, 7
DYLD_LIBRARY_PATH, 7

enhanceText(), 50, *see* CPlot

fplotxy(), 53, **236**
fplotxyz(), 53, **238**
func2D(), 50, *see* CPlot
func3D(), 50, *see* CPlot
funcp2D(), 50, *see* CPlot
funcp3D(), 50, *see* CPlot

getLabel(), 50, *see* CPlot
getOutputType(), 50, *see* CPlot
getSubplot(), *see* CPlot, 50, *see* CPlot
getTitle(), 50, *see* CPlot
grid(), *see* CPlot, 50, *see* CPlot, *see* CPlot

HP-UX, 1

install SIGL, 2
install SIGL in Mac OS X, 6
install SIGL in Unix, 4
install SIGL in Windows, 2
install.sh, 4, 6
isUsed(), *see* CPlot, 50, *see* CPlot

label(), *see* CPlot, 50, *see* CPlot
LD_LIBRARY_PATH, 5
legend(), *see* CPlot, 50, *see* CPlot
legendLocation(), *see* CPlot, 50, *see* CPlot
legendOption(), 50, *see* CPlot
libchplot.lib, 3
line(), *see* CPlot, 50, *see* CPlot
lineType(), 50, *see* CPlot
Linux, 1

Mac OS X, 1, 7
margins(), *see* CPlot, 50, *see* CPlot

origin(), *see* CPlot, 51, *see* CPlot
outputType(), *see* CPlot, 51, *see* CPlot

plot, 8
PLOT_ANGLE_DEG, 32, 39, **51**, 92, 195
PLOT_ANGLE_RAD, 32, 39, **51**, 92, 195
PLOT_AXIS_X, 14, **51**, 59, 61, 62, 67, 124, 135, 205, 217, 219, 222, 224, 228, 229, 231, 232, 234
PLOT_AXIS_X2, **51**, 59, 61, 62, 67, 124, 135, 205, 217, 219, 222, 224, 228, 229, 231, 233, 234
PLOT_AXIS_XY, 14, **51**, 59, 61, 63, 67, 135, 205, 217, 219, 222, 224, 228, 229, 231, 233, 234

PLOT_AXIS_XYZ, 14, **51**, 59, 63, 67, 135, 205, 217, 219, 222, 224, 229, 231, 233, 234
PLOT_AXIS_Y, 14, **51**, 59, 61, 62, 67, 124, 135, 205, 217, 219, 222, 224, 228, 229, 231, 233, 234
PLOT_AXIS_Y2, **51**, 59, 61, 62, 67, 124, 135, 205, 217, 219, 222, 224, 228, 229, 231, 233, 234
PLOT_AXIS_Z, 14, **51**, 59, 63, 67, 124, 135, 205, 217, 219, 222, 224, 229, 231, 233, 234
PLOT_BORDER_ALL, 15, **52**
PLOT_BORDER_BOTTOM, 15, **52**
PLOT_BORDER_LEFT, 15, **52**
PLOT_BORDER_RIGHT, 15, **52**
PLOT_BORDER_TOP, 15, **52**
PLOT_BOXFILL_EMPTY, **52**, 74
PLOT_BOXFILL_PATTERN, **52**, 74
PLOT_BOXFILL_SOLID, **52**, 74
PLOT_CONTOUR_BASE, 38, **52**, 89
PLOT_CONTOUR_SURFACE, **52**, 89
PLOT_COORD_CARTESIAN, 39, **52**, 92
PLOT_COORD_CYLINDRICAL, 39, **52**, 92
PLOT_COORD_SPHERICAL, 39, **52**, 92
PLOT_OFF, 15, **52**, 60, 61, 69, 82, 83, 131, 202, 207, 217, 229
PLOT_ON, 15, **52**, 60, 61, 69, 82, 83, 131, 202, 207, 217, 229
PLOT_OUTPUTTYPE_DISPLAY, 22, **52**, 125, 151
PLOT_OUTPUTTYPE_FILE, 22, **52**, 125, 151
PLOT_OUTPUTTYPE_STREAM, 22, **52**, 125, 151
PLOT_PLOTTYPE_BOXERRORBARS, 29, **52**, 73, 74, 76, 77, 163
PLOT_PLOTTYPE_BOXES, 29, **52**, 73, 74, 76, 77, 163, 189
PLOT_PLOTTYPE_BOXXYERRORBARS, 29, **52**, 73, 74, 76, 77, 163
PLOT_PLOTTYPE_CANDLESTICKS, 29, **52**, 73, 74, 76, 163, 180
PLOT_PLOTTYPE_DOTS, 29, **52**, 162
PLOT_PLOTTYPE_FILLEDCURVES, 29, **52**, 73, 74, 162, 185–187
PLOT_PLOTTYPE_FINANCEBARS, 29, **52**, 68, 164, 181
PLOT_PLOTTYPE_FSTEPS, 29, **52**, 162
PLOT_PLOTTYPE_HISTEPS, 29, **52**, 162
PLOT_PLOTTYPE_IMPULSES, 29, 33, **52**, 162
PLOT_PLOTTYPE_LINES, 29, 33, 38, **52**, 89, 162
PLOT_PLOTTYPE_LINESPOINTS, 29, 33, **52**, 162
PLOT_PLOTTYPE_POINTS, 29, 33, **52**, 162
PLOT_PLOTTYPE_STEPS, 29, **52**, 162
PLOT_PLOTTYPE_SURFACES, 33, 38, **52**, 89, 162
PLOT_PLOTTYPE_VECTORS, 29, 33, **52**
PLOT_PLOTTYPE_Vectors, 163, 179
PLOT_PLOTTYPE_XERRORBARS, 29, **52**, 68, 164
PLOT_PLOTTYPE_XERRORLINES, 29, **52**, 165
PLOT_PLOTTYPE_XYERRORBARS, 29, **53**, 68, 69, 164
PLOT_PLOTTYPE_XYERRORLINES, 29, **53**, 165
PLOT_PLOTTYPE_YERRORBARS, 29, **53**, 69, 164, 184
PLOT_PLOTTYPE_YERRORLINES, 29, **53**, 165
PLOT_PLOTTYPES_VECTORS, 180
PLOT_SCALETYPE_LINEAR, **53**, 205
PLOT_SCALETYPE_LOG, **53**, 205
PLOT_TEXT_CENTER, 16, **53**, 217
PLOT_TEXT_LEFT, 16, **53**, 217
PLOT_TEXT_RIGHT, 16, **53**, 217
PLOT_TICS_IN, **53**, 220
PLOT_TICS_OUT, **53**, 220
plotting(), *see* CPlot, 51, *see* CPlot
plotType(), *see* CPlot, 51, *see* CPlot
plotxy(), 53, **241**
plotxyf(), 53, **243**
plotxyz(), 53, **245**
plotxyzf(), 53, **248**
point(), *see* CPlot, 51, *see* CPlot
pointType(), 51, *see* CPlot
polarPlot(), *see* CPlot, 51, *see* CPlot
polygon(), *see* CPlot, 51, *see* CPlot

rectangle(), *see* CPlot, 51, *see* CPlot
removeHiddenLine(), *see* CPlot, 51, *see* CPlot

scaleType(), *see* CPlot, 51, *see* CPlot
showMesh(), *see* CPlot, 51, *see* CPlot
SIGL, 1
SILIB_HOME, 3, 5, 6
size(), *see* CPlot, 51, *see* CPlot
size3D(), *see* CPlot, 51, *see* CPlot
sizeOutput(), 51, *see* CPlot
sizeRatio(), *see* CPlot, 51, *see* CPlot
smooth(), 51, *see* CPlot

SoftIntegration Graphical Library, 1
Solaris, 1
subplot(), *see* CPlot, 51, *see* CPlot
system requirements, 1

text(), *see* CPlot, 51, *see* CPlot
tics(), *see* CPlot, 51, *see* CPlot
ticsDay(), *see* CPlot, 51, *see* CPlot
ticsDirection(), *see* CPlot, 51, *see* CPlot
ticsFormat(), *see* CPlot, 51, *see* CPlot
ticsLabel(), *see* CPlot, 51, *see* CPlot
ticsLevel(), *see* CPlot, 51, *see* CPlot
ticsLocation(), *see* CPlot, 51, *see* CPlot
ticsMirror(), *see* CPlot, 51, *see* CPlot
ticsMonth(), *see* CPlot, 51, *see* CPlot
ticsPosition(), 51, *see* CPlot
ticsRange(), 51, *see* CPlot
title(), *see* CPlot, 51, *see* CPlot
typographical conventions, ii

uninstall, 4, 6
uninstall SIGL, 3
uninstall SIGL in Mac OS X, 6
uninstall SIGL in Unix, 4
uninstall SIGL in Windows, 3

Visual .NET, 3

Web plotting, 39
Windows 2000, 1
Windows 95, 1
Windows 98, 1
Windows Me, 1
Windows NT, 1
Windows XP, 1