

# The Figure Class

Basic plotting in C++

## Contents

<b>1</b>	<b>Installation</b>	<b>2</b>
1.1	Opening example . . . . .	2
<b>2</b>	<b>Commands</b>	<b>3</b>
2.1	grid . . . . .	3
2.2	xlabel . . . . .	3
2.3	ylabel . . . . .	4
2.4	legend . . . . .	4
2.5	setlog . . . . .	4
2.6	plot . . . . .	5
2.7	plot3 . . . . .	5
2.8	fplot . . . . .	6
2.9	ranges . . . . .	6
2.10	save . . . . .	7
2.11	title . . . . .	7
<b>3</b>	<b>Line characteristics</b>	<b>7</b>

# 1 Installation

First of all note that this library needs MathGL - that's a plotting library with a vast amount of possibilities, options and plot types. The Figure class is only a nice interface for easy use of that particular library.

You can get it at [http://mathgl.sourceforge.net/doc\\_en/Download.html#Download](http://mathgl.sourceforge.net/doc_en/Download.html#Download).

Follow the steps in the INSTALL file.

In the directory of the CMakeLists.txt do:

Under Linux/Mac OS:

```
$ mkdir build
$ cd build
$ cmake ..
$ sudo make install
```

Under Windows:

- open a terminal with administrator rights
- do the same as above but without the `sudo`

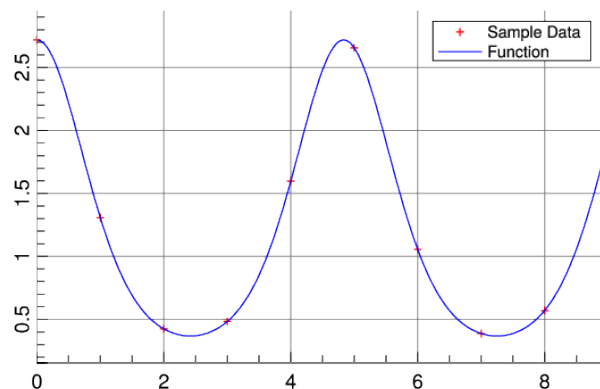
This installation requires CMake (<https://cmake.org/download/>). The “manual” way of installing it is described in INSTALL.

## 1.1 Opening example

This short example code will show, how the Figure class can be used.

```
int main()
{
    std::vector<double> x(10), y(10);
    for (int i = 1; i <= 10; ++i){
        x[i] = i; y[i] = std::exp(std::cos(0.2*i));
    }
    Eigen::VectorXd u = Eigen::VectorXd::LinSpaced(500, 1, 10),
        v = ( (0.2*u.array()).cos().exp() ).matrix();

    Figure fig;
    fig.plot(x, y, " +r", "Sample Data");
    fig.plot(u, v, "b", "Function");
    fig.legend();
    fig.save("plot.eps");
    return 0;
}
```



## 2 Commands

### 2.1 grid

Definition:

```
void grid( const bool on = true,
           const std::string gridType = "-",
           const std::string gridCol = "h" )
```

Restrictions: None.

Examples:

```
Figure fig;
fig.plot(x, y, "r");
fig.grid(false); // unset grid
fig.save("plot.eps");

Figure fig;
fig.plot(x, y, "r");
fig.grid(true, "!", "b"); // blue fine mesh
fig.save("plot.eps");
```

### 2.2 xlabel

Definition:

```
void xlabel( const char* label,
             const double pos = 0 )
```

Restrictions: If setlog is called xlabel should be called beforehand.

Examples:

```
Figure fig;
fig.plot(x, y, "+g"); // '+g' equals matlab/python '+-g'
fig.xlabel("Linear x axis");
fig.save("plot.eps");

Figure fig;
fig.setlog(true, true);
fig.plot(x, y, "+g");
fig.xlabel("Logarithmic x axis"); // Not good - should be called before 'setlog'
fig.save("plot.eps");

Figure fig;
fig.xlabel("Logarithmix x axis"); // Good
fig.setlog(true, true);
fig.plot(x, y, "+g");
fig.save("plot.eps");
```

## 2.3 ylabel

**Definition:**

```
void ylabel( const char* label,
             const double pos = 0 )
```

**Restrictions:** If `setlog` is called `ylabel` should be called beforehand.

**Examples:** See `xlabel`.

## 2.4 legend

**Definition:**

```
void legend( const double xPos = 1,
             const double yPos = 1 )
```

**Restrictions:** None.

## 2.5 setlog

**Definition:**

```
void setlog( const bool logx = false,
             const bool logy = false,
             const bool logz = false )
```

**Restrictions:** All plots will use the latest `setlog` options or default if none have been set.

**Examples:**

```
Figure fig;
fig.setlog(true, false); // -> semilogx
fig.plot(x0, y0, "b");
fig.setlog(false, true); // -> semilogy
fig.plot(x1, y1, "r");
fig.setlog(true, true); // -> loglog
fig.plot(x2, y2, "g");
fig.save("plot.eps"); // ATTENTION: all plots will have been plotted in loglog-scale

Figure fig;
fig.plot(x, y, "b");
fig.save("plot.eps"); // -> default (= linear) scaling
```

## 2.6 plot

**Definition:**

```
template <typename yVector>
void plot( const yVector& y,
           const std::string style,
           const char* legend = 0 )

template <typename xVector, typename yVector>
void plot( const xVector& x,
           const yVector& y,
           const std::string style,
           const char* legend = 0 )
```

**Restrictions:** `xVector` and `yVector` must have a `size()` method, which returns the size of the vector and a `data()` method, which returns a pointer to the first element in the vector. Furthermore `x` and `y` must have same length. Also note that the `style`-argument is required!

**Examples:**

```
Figure fig;
fig.plot(x, y, "b");
fig.save("data.eps");
```

```
Figure fig;
fig.plot(x, y); // Not OK - style missing
fig.save("data.eps");
```

```
Figure fig;
fig.plot(x, y, " *r", "Data w/ red dots"); // ' *r' equals matlab/python 'r*'
fig.save("data.eps");
```

## 2.7 plot3

**Definition:**

```
template <typename xVector, typename yVector, typename zVector>
void plot3( const xVector& x,
            const yVector& y,
            const zVector& z,
            const std::string style,
            const char* legend = 0 )
```

**Restrictions:** Same restrictions as in `plot` for two vectors, extended to `zVector`.

**Examples:**

```
Figure fig;
fig.plot3(x, y, z, "b");
fig.save("trajectories.eps");
```

## 2.8 fplot

**Definition:**

```
void fplot( const std::string function,
            const std::string style,
            const char* legend = 0 )
```

**Restrictions:** None.

**Examples:**

```
Figure fig;
fig.fplot("3*x^2 + 4.5/x + exp(x)", "b");
fig.fplot("exp(cos(pi*x))", "r", "some periodic function");
fig.ranges(0.5, 2, 0, 5); // be sure to set ranges for fplot!
fig.save("plot.eps");

Figure fig;
fig.plot(x, y, "b", "Benchmark");
fig.fplot("x^2", "k;", "\\ 0(x^2)");
// here we don't set the ranges as it uses the range given by the x,y data
// and we use fplot to draw a reference line (0(x^2))
fig.save("runtimes.eps");
```

## 2.9 ranges

**Definition:**

```
void ranges( const double xMin,
            const double xMax,
            const double yMin,
            const double yMax )
```

**Restrictions:**  $xMin < xMax$ ,  $yMin < yMax$  and ranges must be  $> 0$  for axis in logarithmic scale.

**Examples:**

```
Figure fig;
fig.ranges(-1,1,-1,1);
fig.plot(x, y, "b");

Figure fig;
fig.plot(x, y, "b");
fig.ranges(0, 2.3, 4, 5); // ranges can be called before or after 'plot'

Figure fig;
fig.ranges(-1, 1, 0, 5);
fig.setlog(true, true); // will run but MathGL will throw a warning
fig.plot(x, y, "b");
```

## 2.10 save

### Definition:

```
void save( const char* file )
```

**Restrictions:** The filename *must* end on .eps!

### Examples:

```
Figure fig;  
fig.save("plot.eps"); // OK
```

```
Figure fig;  
fig.save("plot.png"); // Not OK - Only eps-format supported!
```

## 2.11 title

### Definition:

```
void title( const char* text )
```

**Restrictions:** None.

## 3 Line characteristics

Linecolors<sup>a</sup>:

blue	b
green	g
red	r
cyan	c
magenta	m
yellow	y
gray	h
green-blue	l
sky-blue	n
orange	q
green-yellow	e
blue-violet	u
purple	p

<sup>a</sup> Upper-case letters will give a darker version of the lower-case version.

Linestyles:

none	
solid	-
dashed	;
small dashed	=
long dashed	
dotted	:
dash-dotted	j
small dash-dotted	i

None is used as follows:  
" r\*" gives red stars w/o  
any lines

Linemarkers:

+	+
o	o
◇	d
·	·
△	^
▽	v
◁	<
▷	>
⊙	#.
⊞	#+
⊠	#x