Extension to ePiX

Svend Daugaard Pedersen

25th February 2007

Contents

1 Introduction

ePiX_ext is a set of extensions to the ePiX library, offering a Cartesian and logarithmic coordinate systems (both single and double) with a lot of parameters that can be set by the user. Further more it offers drawing of a hatched polygon as well as hatched area between two function graphs.

You need some familiarity with ePiX to be able to use this extension.

2 Installation

To install ePiX_ext, use the option `--with-contrib` when you configure the main package. The library will be compiled separately (`libepixext.a`), and the library and header will be installed automatically when you do `make install`.

To use these extensions in a source file, the header must be included explicitly in each source file's preamble:

```
#include "epix.h"
#include "epix_ext.h"

using namespace ePiX;
using namespace ePiX_contrib;
```

3 Cartesian Coordinate System

The declaration statement

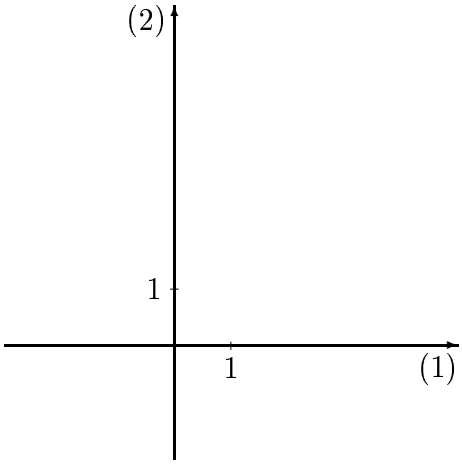
```
CartesianCoord cs(-3,5,-2,6);
```

creates a cartesian coordinate system with x-bounds $-3 \ldots 5$ and y-bounds $-2 \ldots 6$. To draw the coordinate system using the standard layout set by the constructor of *CartesianCoord*, execute the member function *draw* in *cs*:

```
cs.draw();
```

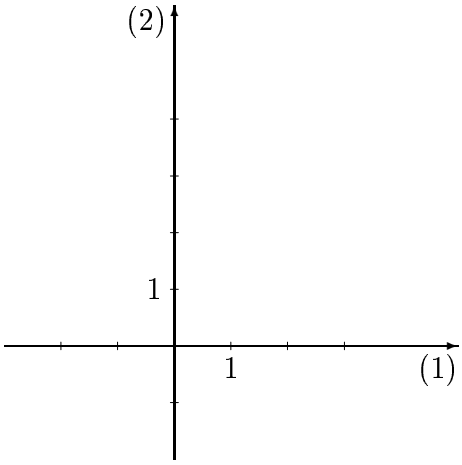
Here is an example of a complete C++ program making such a standard coordinate system:

```
#include "epix.h"

int main()
{
  CartesianCoord cs(-3,5,-2,6);

  picture(P(12, 12));
  unitlength("5mm");

  begin();
  cs.draw();
  end();

  return 0;
}
```



To make more division points you have to execute the member function *xMark()* and/or *yMark()* with the position of the first mark and the number of marks as parameters.

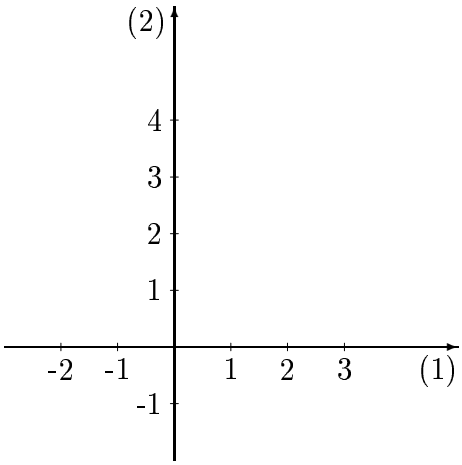
```
#include "epix.h"

int main()
{
  CartesianCoord cs(-3,5,-2,6);

  picture(P(12, 12));
  unitlength("5mm");

  begin();
  cs.xMarks(-2,6);
  cs.yMarks(-1,6);
  cs.draw();
  end();

  return 0;
}
```



The member functions *xMark()* and *yMarks()* accept a third parameter specifying the the distance between the marks. The default value of this is 1.

The member functions *xLabels()* and *yLabels()* are setting other labels on the axis than the default. These member functions have one, two or three parameters. The first (mandatory) parameter is a NULL terminated array of string pointers. The second parameter is used to specify the position af the first label (default is position of the first mark) and the third the distance between the labels (default is distance between marks).

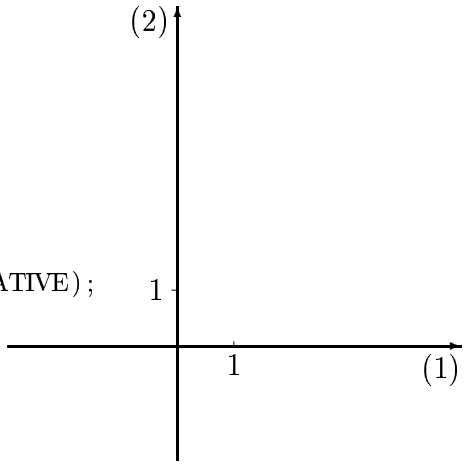
```
#include "epix.h"

static char* xlabels[] = {"-2","-1","","1","2","3",NULL};
static char* ylabels[] = {"-1","","1","2","3","4",NULL};

int main()
{
  CartesianCoord cs(-3,5,-2,6);

  picture(P(12,12));
  unitlength("5mm");

  begin();
  cs.xMarks(-2,6);
  cs.yMarks(-1,6);
  cs.xLabels(xlabels);
  cs.yLabels(ylabels);
  cs.draw();
  end();

  return 0;
}
```



Note the empty string at the 0-position to avoid collision between a label and the axis.

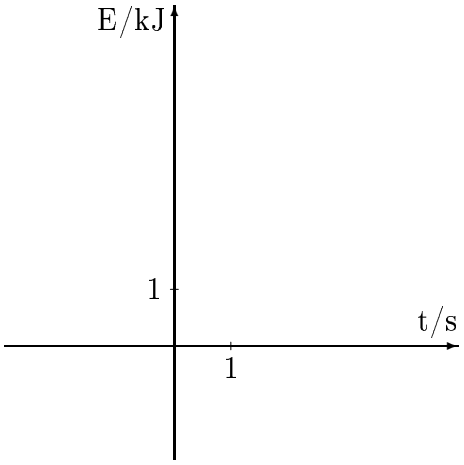
The length and position of the marks as well as the position of the labels can be specified. The position values are POSITIVE, NEGATIVE and (for the marks) CENTER. The marks are placed with the member function *markLayout()* with four parameters: the mark length (in pt) and the position for x-axis and the same for the y-axis. The labels position is set using the member function *labelPos()* with one or two parameters. The one parameter version sets the position to the same for both axis.

```
#include "epix.h"
int main()
{
  CartesianCoord cs(-3,5,-2,6);
  picture(P(12,12));
  unitlength("5mm");
  begin();
  cs.markLayout(2,POSITIVE,2,NEGATIVE);
  cs.labelPos(NEGATIVE);
  cs.draw();
  end();
  return 0;
}
```



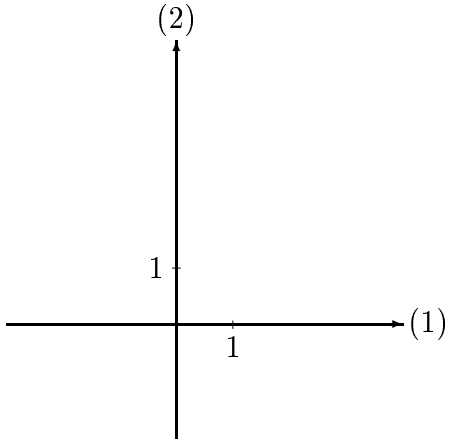
The axis name and the position of the name can be set with the member functions *xName()* and *yName()*. The parameters are the name and the position.

```
#include "epix.h"
int main()
{
  CartesianCoord cs(-3,5,-2,6);
  picture(P(12,12));
  unitlength("5mm");
  begin();

  cs.xName("t/s",POSITIVE);
  cs.yName("E/kJ",NEGATIVE);
  cs.draw();

  end();
  return 0;
}
```



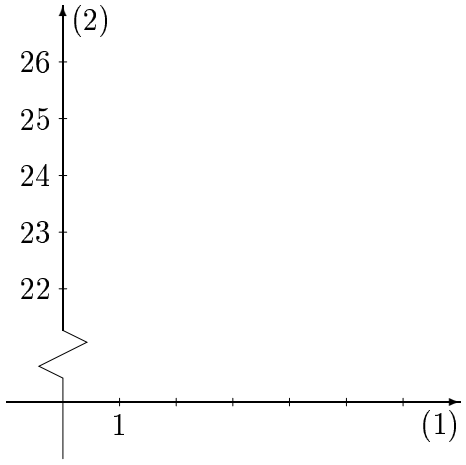
The CENTER position of the names places the names at the end of the axis. If this position is used you probably need to shorten the length of the axis (or the names will be written outside the picture bounds). This is done by calling the member function *axisBounds())*. This function accepts one or two parameters. The one-parameter version will set the upper right "corner" (most probably what you need), or both lower left and upper right corner.

```
#include "epix.h"
int main()
{
  CartesianCoord cs(-3,5,-2,6);
  picture(P(12,12));
  unitlength("5mm");
  begin();

  cs.axisBounds(P(4,5));
  cs.xName(CENTER);
  cs.yName(CENTER);
  cs.draw();

  end();
  return 0;
}
```



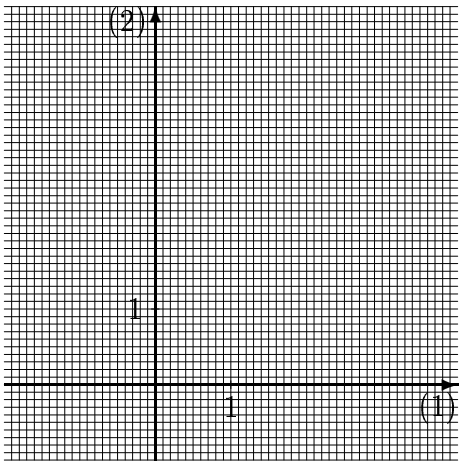
Also the crossing point for the axis can be specified (default is (0,0)) and you can ask for a "broken" axis:

```
#include "epix.h"
static char* ylabels[] = {"22","23","24","25","26",NULL};
int main()
{
  CartesianCoord cs(-1,7,19,27);
  picture(P(12, 12));
  unitlength("5mm");
  begin();

  cs.axisCross(P(0,20));
  cs.xMarks(1,6);
  cs.yMarks(22,5);
  cs.yLabels(ylabels);
  cs.yName(POSITIVE);
  cs.yBroken();
  cs.draw();

  end();
  return 0;
}
```



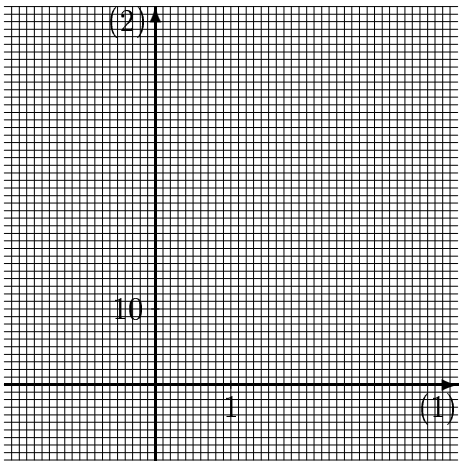
A graph paper can be added simply by calling the member function *grid()*.

```
#include <epix.h>
int main()
{
  CartesianCoord cs(-2,4,-1,5);
  picture(P(12,12));
  unitlength("5mm");
  begin();

  cs.grid();
  cs.draw();

  end();
  return 0;
}
```



The position of the paper is set by calling the member function *gridPosition()* with a bold line crossing point as parameter (default: axis crossing point). The density of the gridlines is set by calling the member function *gridDensity()* (see reference section for more details):

```
#include <epix.h>
static char* ylabels[] = {"10",NULL};
int main()
{
  CartesianCoord cs(-2,4,-10,50);
  picture(P(12,12));
  unitlength("5mm");
  begin();

  cs.yMarks(10,1);
  cs.yLabels(ylabels);
  cs.grid();
  cs.gridDensity(1,9,5,10,9,0);
  cs.draw();

  end();
  return 0;
}
```



4 Logarithmic Coordinate Systems

Logarithmic coordinate systems are made in much the same way as the cartesian coordinate system.

4.1 SingleLog Cordinate System

A single logarithmic coordinate system has a horizontal linear axis and a vertical logarithmic axis.

The declaration statement

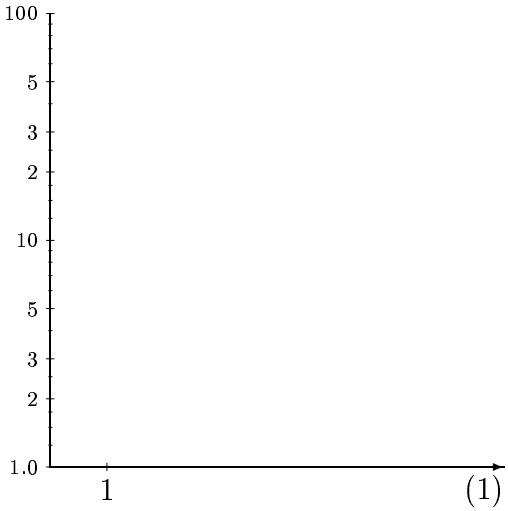
```
SingleLogCoord cs(0,10,1,100);
```

creates a logarithmic coordinate system with x-bounds $0 \ldots 10$ and y-bounds $1 \ldots 10$. To draw the coordinate system using the standard layout set by the constructor of *SingleLogCoord*, execute the member function *draw* in *cs*:

```
cs.draw();
```

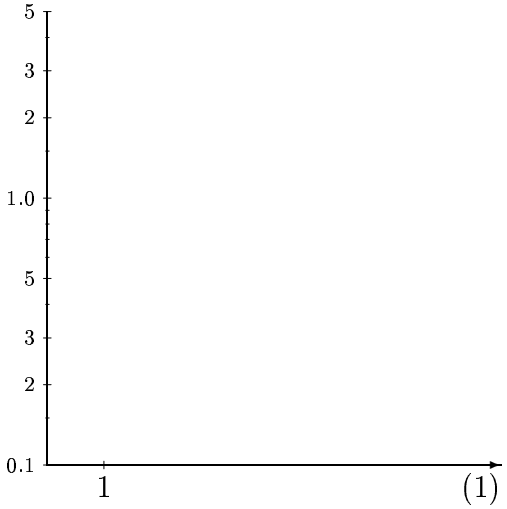
Here is an example of a complete C++ program making such a standard coordinate system:

```
#include "epix.h"

int main()
{
  SingleLogCoord sc(0,8,1,100);

  picture(P(12,12));
  unitlength("5mm");
  begin();

  sc.draw();

  end();
  return 0;
}
```



The logaritmic axis need not contain only full decades. The following C++ program is used to create a coordinate system with one and a "half" decades from 0.1 to 5:

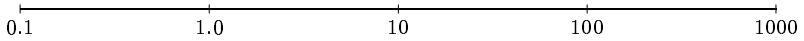
```
#include "epix.h"

int main()
{
  SingleLogCoord sc(0,8,0.1,5);
  picture(P(12,12));
  unitlength("5mm");
  begin();

  sc.logStyle(3);
  sc.draw();

  end();
  return 0;
}
```



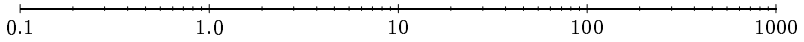
In this case the standard logaritmic style number 3 is used to set the division of the logaritmic axis.

There are six standard styles giving an increasing number of division points:

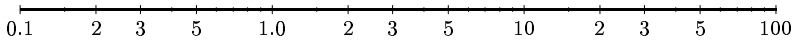
stdstyle = 1:



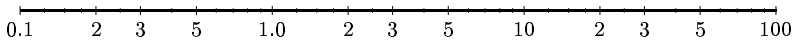
stdstyle = 2:



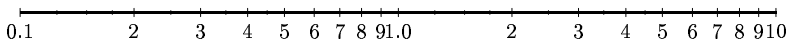
stdstyle = 3:



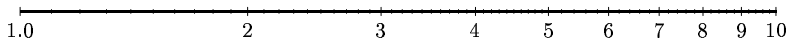
stdstyle = 4:



stdstyle = 5:



stdstyle = 6:

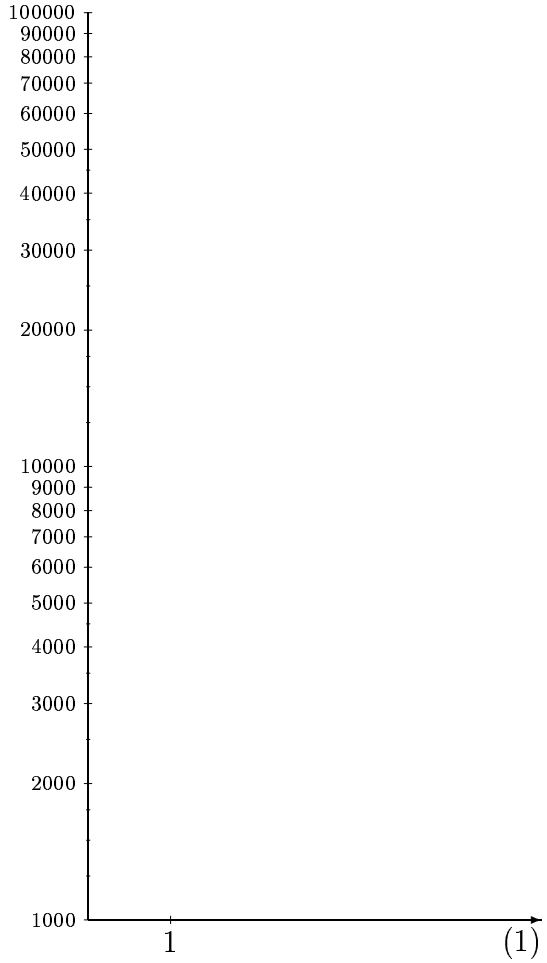


The default *stdstyle* is 4.

The start of each decade is labeled as shown inside the range 0.01 to 1000. Outside this range powers of ten are used.

To customize marking and labelling call the member function *logStyle()* as shown here:

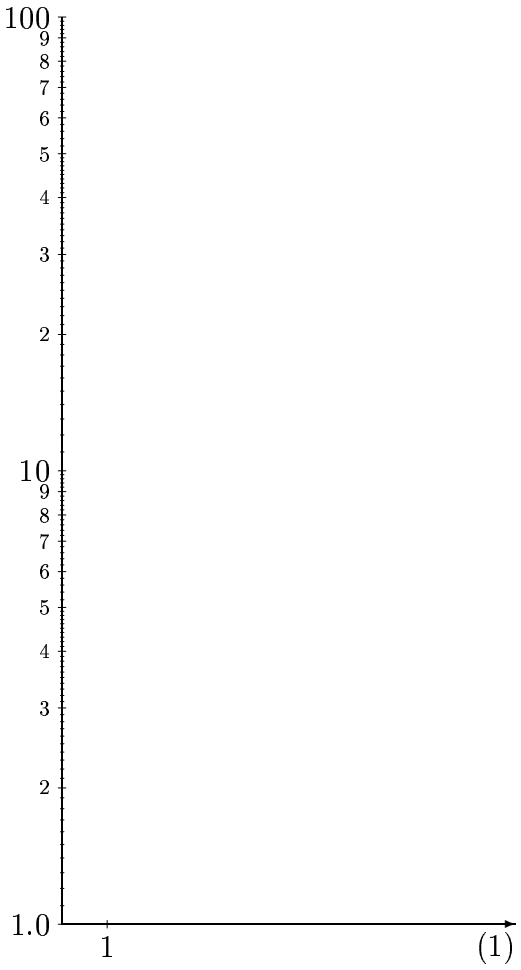
```
#include "epix.h"

Mark mrk[] =
 {
  {1,"1000",3},{2,"2000",1},
  {3,"3000",1},{4,"4000",1},
  {5,"5000",0},{6,"6000",0},
  {7,"7000",0},{8,"8000",0},
  {9,"9000",0},{1,"10000",3},
  {2,"20000",1},{3,"30000",1},
  {4,"40000",1},{5,"50000",0},
  {6,"60000",0},{7,"70000",0},
  {8,"80000",0},{9,"90000",0},
  {1,"100000",0},{10,NULL,0}
 };
int main()
{
  SingleLogCoord
    cs(-1,10,1000,100000);
  picture(P(12,24));
  unitlength("5mm");
  begin();

  cs.logStyle(mrk);
  cs.draw();

  end();
  return 0;
}
```



The variable *mrk* is a table of structures each containing three fields. The first field holds the position of the mark (the position within the actual decade). The next field is the label to be printed. And the third is the number of subdivisons following this mark. The table ends with position 10.

Another example:

```
#include "epix.h"

Mark mrk[] =
 {
  {1,NULL,9},
  {2,"\\scriptsize 2",9},
  {3,"\\scriptsize 3",9},
  {4,"\\scriptsize 4",9},
  {5,"\\scriptsize 5",4},
  {6,"\\scriptsize 6",4},
  {7,"\\scriptsize 7",4},
  {8,"\\scriptsize 8",4},
  {9,"\\scriptsize 9",4},
  {10,NULL,0}
 };

int main()
{
  SingleLogCoord cs(0,10,1,100);
  picture(P(12,24));
  unitlength("5mm");
  begin();

  cs.logStyle(mrk);
  cs.labelAttr("");
  cs.draw();

  end();
  return 0;
}
```

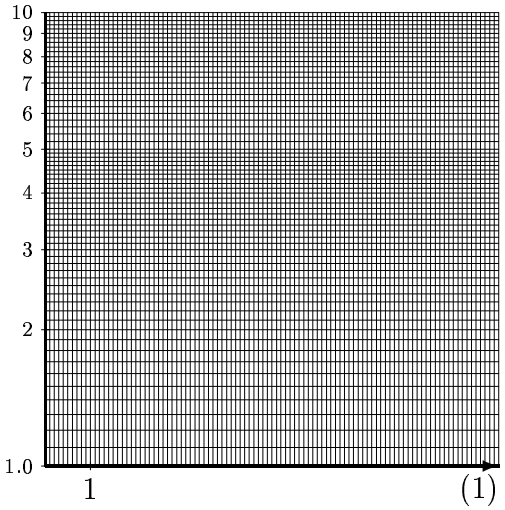


mrk is here a one-decade table. It is used for both decades on the axis. The NULL label at position 1 has the effect that the current decade value is printed (here 1.0, 10 and 100).

The member function *labelAttr()* can be used to set any attribute, for instance \it and/or \tiny. But note that you have to use a double backslash in a C++ string, because in C++ strings (like in TEX) a backslash is a command introducer. In the example the attribute is set to the empty string so that the numbers at the start of each decade are printed in normal size (default attribute string is "\\scriptsize"). All other numbers are printed using \scriptsize as specified in the Mark table.

A graph paper can be added in the same way as for the cartesian coordinate system:

```
#include "epix.h"
int main()
{
  SingleLogCoord cs(0,10,1,10);
  picture(P(12,12));
  unitlength("5mm");
  begin();

  cs.logStyle(6);
  cs.grid();
  cs.draw();

  end();
  return 0;
}
```

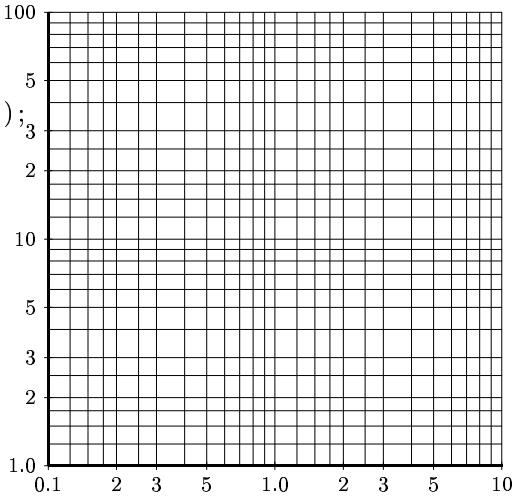


4.2 DoubleLog Cordinate System

A double logarithmic coordinate system has two logarithmic axisis. It is handled in much the same way as the Single Logarithmic Coordinate System. Just change *Single* to *Double*.

Here is an example:

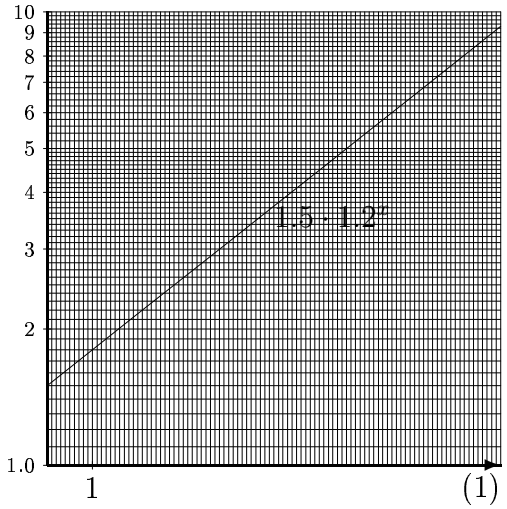
```
#include "epix.h"
int main()
{
  DoubleLogCoord cs(0.1,10,1,100);

  picture(P(12,12));
  unitlength("5mm");
  begin();

  cs.logStyle(4);
  cs.grid();
  cs.draw();

  end();
  return 0;
}
```



4.3 Graphs in Logarithmic Cordinate System

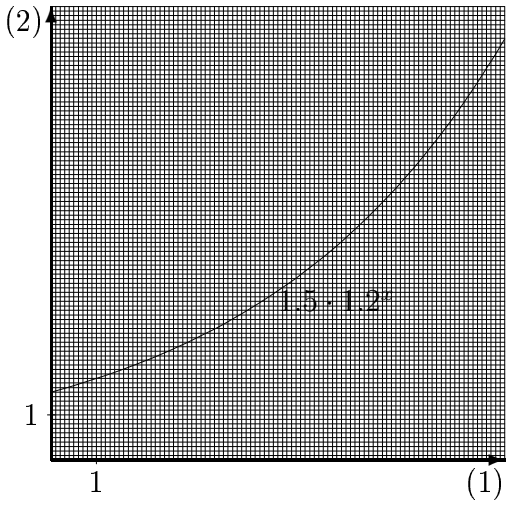
In the coordinate systems the member functions *plot()*, *adplot()*, *label()* and *marker()* are used in the same way as the corresponding ePiX functions. But they can be used in logarithmic coordinate systems, too.

```
#include "epix.h"
double f(double x)
{ return 1.5*pow(1.2,x); }
int main()
{
  SingleLogCoord cs(0,10,1,10);
  picture(P(12,12));
  unitlength("5mm");
  begin();
  cs.logStyle(6);
  cs.grid();
  cs.draw();
  cs.plot((*f),0,10,10);
  cs.label(P(5,f(5)),P(0,0),
     "$1.5\\cdot1.2^x$",br);
  end();
  return 0;
}
```



By changing *SingleLog* to *Cartesian* (and doing some other natural changes), you get:

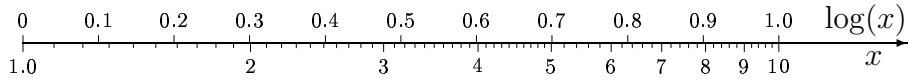
```
#include "epix.h"
double f(double x)
{ return 1.5*pow(1.2,x); }
int main()
{
  CartesianCoord cs(0,10,0,10);
  picture(P(12,12));
  unitlength("5mm");
  begin();
  cs.grid();
  cs.draw();
  cs.plot((*f),0,10,10);
  cs.label(P(5,f(5)),P(0,0),
     "$1.5\\cdot1.2^x$",br);
  end();
  return 0;
}
```



4.4 Printing Coordinate Axis

Coordinate axis can be printed alone. The output of the program below shows the theory behind logarithmic axis:

```
#include "epix.h"
static char* linLabels[] =
  {"\\scriptsize 0","\\scriptsize 0.1","\\scriptsize 0.2",
   "\\scriptsize 0.3","\\scriptsize 0.4","\\scriptsize 0.5",
   "\\scriptsize 0.6","\\scriptsize 0.7","\\scriptsize 0.8",
   "\\scriptsize 0.9","\\scriptsize 1.0",NULL
  };
int main()
{
  HorizLinAxis linAs;
  HorizLogAxis logAs;

  bounding_box(P(0,0),P(1,1.17));
  picture(P(10,1.5));
  unitlength("1cm");
  begin();

  linAs.nummrk = 11;
  linAs.distmrk = 0.1;
  linAs.mrk1 = 0; linAs.mrkpos = POSITIVE;
  linAs.labels = linLabels; linAs.labpos = POSITIVE;
  linAs.name = "$\\log(x)$"; linAs.nampos = POSITIVE;
  linAs.draw(P(0,0.75),1.17);

  logAs.drawaxis = false;
  logAs.arrow = true; logAs.arrowextra = 5;
  logAs.stdstyle = 6;
  logAs.mrkpos = NEGATIVE;
  logAs.name = "$x$";
  logAs.draw(P(0,0.75),1);

  end();
  return 0;
}
```



See *epix_ext.h* for a description of the fields in the axis structures.

5 Hatched figures

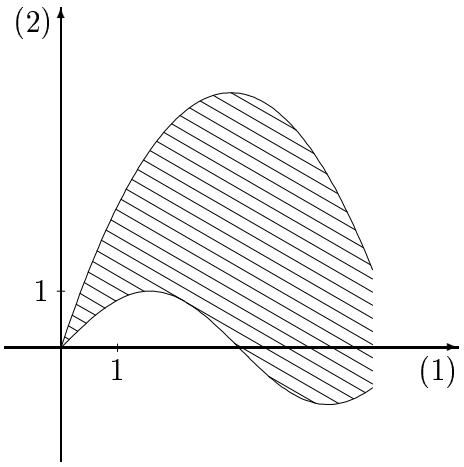
5.1 Hatched area

Commands to draw a hatched area between two graphs (or one graph and the x-axis) in a cartesian coordinate system are generated by calling the member functions:

```
void hatchArea(double angle,double dist,
               double f(double),double g(double),
               double a, double b, int n);

void hatchArea(double angle,double dist,
               double f(double),double a,double b, int n)
```

The first two parameters *angle* and *dist* specifies the angle of and the distance between the hatch lines. The next are the function(s), the start and the end point and the number of points used to draw the graphs.

```
#include "epix.h"

double f(double x)
{
  return 4.5-(x-3)*(x-3)/2;
}

int main()
{
  CartesianCoord cs(-1,7,-2,6);
  picture(P(12,12));
  unitlength("5mm");
  begin();

  cs.draw();
  cs.hatchArea(150,0.2,f,std::sin,0,5.5,100);

  end();
  return 0;
}
```



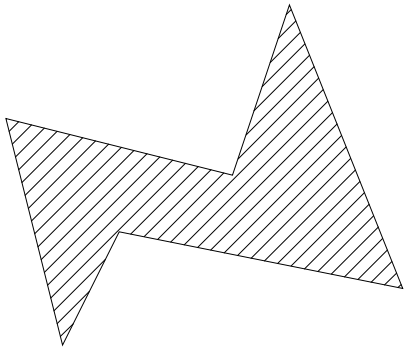
5.2 Hatched polygon

The member function *hatchArea()* is based on routines for drawing hatched polygons. They are generated by the procedures:

```
void hatch_polygon(double angle,double dist,
                   int corners,triple* points);
void hatch_polygon(double angle,double dist,
                   int corners,triple& points, ...);

void hatch(double angle,double dist,
           int corners,triple* points);
void hatch(double angle,double dist,
           int corners,triple& points, ...);
```

The difference between *hatch_polygon()* and *hatch()* is that *hatch()* will not draw the polygon. The meaning of the first two parameters are the same as for the *hatchArea()* member function. The next parameter is the number of corners in the polygon and then the corners.

```
#include "epix.h"

triple corners[] =
  {P(-2,-3),P(-3,1),P(1,0),P(2,3),P(4,-2),P(-1,-1)};

int main()
{
  bounding_box(P(-4,-3),P(4,2));
  picture(P(12,7.5));
  unitlength("5mm");

  begin();

  hatch_polygon(45,0.2,6,corners);

  end();

  return 0;
}
```



As is seen from the example, the polygon need not be convex. But the result is unpredictable if some of the edges cosses each other.

6 Setting Line Thickness

The line thickness used to draw axis (bold and normal), bold, medium and fine grid lines can be changed simply by setting variables to new values. These variables and their standard values are:

```
char* normalThickness      = "\\thinlines";
char* normalAxisThickness  = "\\thinlines";
char* boldAxisThickness    = "\\thicklines";
char* boldGridThickness    = "\\thinlines";
char* mediumGridThickness  = "\\allinethickness{0.075mm}";
char* fineGridThickness    = "\\allinethickness{0.03mm}";
char* crossHatchThickness  = "\\allinethickness{0.04mm}";
```

7 ePiX_ext reference

7.1 Member functions for all coordinate systems

void bounds(double xmin,double xmax,double ymin,double ymax)

Description:
Set bounds of coordinate system.

Parameters:
xmin: lower x-bound of picture.
xmax: upper x-bound of picture.
ymin: lower y-bound of picture.
ymax: upper y-bound of picture.

void draw()

Description:
Draw the coordinate system.

Parameters:
none.

void showAxis(bool b)
void showAxis(bool bx,bool by)

Description:
Draw axis or not.

Parameters:
b: if true draw both x- and y-axis.
bx: if true draw x-axis.
by: if true draw y-axis.

Constructor settings:
true for both axis.

void showArrow(bool b)
void showArrow(bool xb, bool yb)

Description:
Draw axis with or without an arrow.

Parameters:
b: if true draw arrow on both x- and y-axis.
bx: if true draw arrow on x-axis.
by: if true draw arrow on y-axis.

Constructor settings:
true for both axis.

void xName(Position pos)
void xName(char* name,Position pos)

Description:
Place name at the end of the x-axis.

Parameters:
name: name of the x-axis.
pos: position of name (POSITIVE, NEGATIVE or CENTER).

Constructor settings:
name: "(1)" for linear axis and **NULL** for logarithmic.
pos: NEGATIVE.

void yName(Position pos)
void yName(char* name,Position pos)

Description:
Place name at the end of the y-axis.

Parameters:
name: name of the y-axis.
pos: position of name (POSITIVE, NEGATIVE or CENTER).

Constructor settings:

name: "(2)" for linear axis and **NULL** for logarithmic.
pos: NEGATIVE.

void labelPos(Position pos)
void labelPos(Position xpos,Position ypos)

Description:
Set position of (mark) labels on the axis.

Parameters:
pos: position for labels on both x- and y-axis.
xpos: position for labels on x-axis.
ypos: position for labels on y-axis.

Constructor settings:
NEGATIVE for both axis.

void markLayout(double length,Position pos=CENTER)
void markLayout(double lx,Position px,double ly,Position py)

Description:
Set length and position of marks on the axis.

Parameters:
length: length (in pt) of marks on both x-axis and y-axis.
pos: position of marks on both x-axis and y-axis.
lx: length (in pt) of marks on x-axis.
px: position of marks on x-axis.
ly: length (in pt) of marks on y-axis.
py: position of marks on y-axis.

Constructor settings:
length: 3 pt.
pos: CENTER.

void boldAxis(bool b=true)
void boldAxis(bool bx,bool by)

Description:
draw axis with bold line.

Parameters:
b: if true draw both x- and y-axis with bold line.
bx: if true draw x-axis with bold line.
by: if true draw y-axis with bold line.

Constructor settings:
For both axis **true** if grid is selected, otherwise **false**.

void adplot(double f(double), double xmin, double xmax, int n)
void plot(double f(double), double xmin, double xmax, int n)

Description:
Works like ePiX adplot and plot but takes care of the the actual type of coordinate system (cartesian or logarithmic).

Parameters:
f: function to draw graph of.
xmin: lower x-bound of graph.
xmax: upper x-bound of graph.
n: number of points used to draw (straight lines between each point).

void label(triple base,triple offset,char* text)
void label(triple base,char *text)
void label(triple base,triple offset,char* text,epix_label_posn pos)

Description:
Works like ePiX label but takes care of the the actual type of coordinate system (cartesian or logarithmic).

Parameters:
base: base point of label.
offset: offset (in pt) of label.
text: text to write.
pos: position relative to base (c,r,tr,rt,t,tl,lt,l,bl,lb,b,br,rb).

7.2 Member functions for class CartesianCoord

CartesianCoord()
CartesianCoord(double xmin,double xmax,double ymin,double ymax)

Description:
Constructor for class CartesianCoord.

Parameters:

xmin: lower x-bound of picture (default -1).
xmax: upper x-bound of picture (default 10).
ymin: lower y-bound of picture (default -1).
ymax: upper y-bound of picture (default 10).

void axisCross(triple point)

Description:
Set the axis crossing point.

Parameters:
point: crossing point for the axis.

Constructor settings:
(0,0).

void axisBounds(triple upperRight)
void axisBounds(triple lowerLeft,triple upperRight)

Description:
Set bouds of axis in coordinate system.

Parameters:
lowerLeft: lower left corner of axis.
upperRight: upper right corner of axis.

Constructor settings:
Bounds of coordinate system.

void xMarks(double first,int num)
void xMarks(double first,int num,double dist)

Description:
Specify number and position of marks on x-axis.

Parameters:
first: position of first mark.
num: number of marks to set.
dist: distance between marks (default is 1.0).

Constructor settings:
first: 1.
num: 1.
dist: 1.

void yMarks(double first,int num).
void yMarks(double first,int num,double dist).

Description:
Specify number and position of marks on y-axis.

Parameters:
first: position of first mark.
num: number of marks to set.
dist: distance between marks (default is 1.0).

Constructor settings:
first: 1.
num: 1.
dist: 1.

void xLabels(char lab)**
void xLabels(char lab,double first)**
void xLabels(char lab,double first,double dist)**

Description:
Labels to place on x-axis.

Parameters:
lab: the labels in a **NULL**-terminated array of string pointers.
first: position of label (default is position of first mark).
dist: distance between labels (default distance between marks).

Constructor settings:
lab: {"1",**NULL**}.
first: first mark.
dist: distance between marks.

void yLabels(char lab)**
void yLabels(char lab,double first)**
void yLabels(char lab,double first,double dist)**

Description:
Labels to place on y-axis.

Parameters:
lab: the labels in a **NULL**-terminated array of string pointers.
first: position of label (default is position of first mark).
dist: distance between labels (default distance between marks).

Constructor settings:

lab: {"1",**NULL**}.
first: first mark.
dist: distance between marks.

void xBroken()

Description:
Make x-axis broken (to show that axis cross is not zero).

Parameters:
None.

void yBroken()

Description:
Make y-axis broken (to show that axis cross is not zero).

Parameters:
None.

void grid()
void grid(double xmin,double xmax,double ymin,double ymax)

Description:
Add a grid to the coordinate system. Draw coordinate axis in bold unless otherwise specified.

Parameters:
xmin: left edge of grid (default is left edge of coordinate system).
xmax: right edge of grid (default is right edge of coordinate system).
ymin: bottom edge of grid (default is bottom edge of coordinate system).
ymax: top edge of grid (default is top edge of coordinate system).

void gridPosition(triple boldCross)

Description:
Set position of grid in coordinate system.

Parameters:
boldCross: a crossing point for bold lines.

Constructor settings:
Axis crossing point.

void gridDensity(double dist)
void gridDensity(double dist,int num)
void gridDensity(double dist,int num,int mid)
void gridDensity(double xd,int xn,int xm,double yd,int yn,int ym)

Description:
Set density of grid lines.

Parameters:
dist: distance between grid lines.
num: number of fine lines (default is 9).
mid: the fine line to draw half bold (default is 0 meaning no half bold).
xd,xn,xm: dist, num and mid for x-axis.
yd,yn,ym: dist, num and mid for y-axis.

Constructor settings:
dist: 1.
num: 9.
mid: 5.

void hatchArea(double v,double d,double f(double),
double a,double b,int n)

Description:
Hatched area between graph and x-axis.

Parameters:
v: angle between x-axis and hatch line.
d: distance between hatch lines.
f: graph function.
a: left edge af area.
b: right edge of area.
n: number of points used to draw the graphs (see plot and adplot).

void hatchArea(double v,double d,double f(double),double g(double)
double a,double b,int n)

Description:
Hatched area between two graphs.

Parameters:

v: angle between x-axis and hatch line.
d: distance between hatch lines.
f: upper graph function.
g: lower graph function.
a: left edge af area.
b: right edge of area.
n: number of points used to draw the graphs (see plot and adplot).

7.3 Member functions for class SingleLogCoord

SingleLogCoord()
SingleLogCoord(double xmin,double xmax,double ymin,double ymax)

Description:
Constructor for class SingleLogCoord.

Parameters:
xmin: lower x-bound of picture (default -1).
xmax: upper x-bound of picture (default 10).
ymin: lower y-bound of picture (default 1).
ymax: upper y-bound of picture (default 100).

void axisCross(triple point)

Description:
Set crossing point for coordinate axis. Set to (xmin,ymin) by the constructor.

Parameters:
point: crossing point.

Constructor settings:
(xmin,ymin).

void axisBounds(triple upperRight)
void axisBounds(triple lowerLeft,triple upperRight)

Description:
Set bounds of axis in coordinate system. Set by constructor to bounds of picture.

Parameters:

lowerLeft: lower left corner of axis.
upperRight: upper right corner of axis.

Constructor settings:
lowerLift: (xmin,ymin).
upperRight: (xmax,ymax).

void xMarks(double first,int num)
void xMarks(double first,int num,double dist)

Description:
Specify number and position of marks on x-axis.

Parameters:
first: position of first mark.
num: number of marks to set.
dist: distance between marks (default is 1.0).

Constructor settings:
first: 1.
num: 1.
dist: 1.

void yMarks(double first,double last)

Description:
Set bounds of marks on logarithmic axis.

Parameters:
first: lower bound of marks.
last: upper bounds marks.

Constructor settings:
first: ymin.
last: ymax.

void xLabels(char lab)**
void xLabels(char lab,double first)**
void xLabels(char lab,double first,double dist)**

Description:
Labels to place on x-axis.

Parameters:

lab: the labels in a **NULL**-terminated array of string pointers.
first: position of label (default is position of first mark).
dist: distance between labels (default distance between marks).

Constructor settings:
lab: {"1",**NULL**}.
first: first mark.
dist: distance between marks.

void showArrow(bool b)
void showArrow(double extra)
void showArrow(bool b,double extra)

Description:
Draw axis with or without an arrow and add extra length to logarithmic axis.

void showArrow(bool b) will only affect x-axis.
void showArrow(double extra) will only affect y-axis.
void showArrow(bool b,double extra) affects both axis.

Parameters:
b: if true draw arrow on x-axis.
extra: add extra length (in pt) to y-axis and draw arrow.

Constructor settings:
Arrow on x-axis, no arrow on y-axis.

void xBroken()

Description:
Make x-axis broken (to show that axis cross is not zero).

Parameters:
None.

Constructor settings:
No broken axis.

void logStyle(int style)
void logStyle(Mark* mrk)

Description:

Set style of logarithmic axis. *Mark* is defined as:

```
class Mark
{
  public:
   double position; // position of main mark
   char*  label;    // label to print at main mark
   int    numsub;   // number of submarks until next main mark
};
```

Parameters:
style: standard style number (1 . . . 6).
mrk: pointer to an array of marks describing the logarithmic axis.

Constructor settings:
Standard style 4.

void labelAttr(char* attr)

Description:
Label attribute to use if label is not specified (**NULL**).

Parameters:
attr: attribute string.

Constructor settings:
attr = "\\scriptsize".

void grid()
void grid(double xmin,double xmax)

Description:
Add a grid to the coordinate system. Draw coordinate axis in bold unless otherwise specified.

Parameters:
xmin: left edge of grid (default is left edge of coordinate system).
xmax: right edge of grid (default is right edge of coordinate system).

Constructor settings:
xmin: left edge of coordinate system.
xmax: right edge of coordinate system.

void gridPosition(double crossX)

Description:
Set position of grid lines perpendicular to x-axis.

Parameters:

crossX: position for a bold line.

Constructor settings:
logarithmic axis position.

void gridDensity(double dist)
void gridDensity(double dist,int num)
void gridDensity(double dist,int num,int mid)

Description:
Density of grid lines perpendicular to x-axis.

Parameters:
dist: distance between grid lines.
num: number of fine lines (default is 9).
mid: the fine line to draw half bold (default 0 meaning no half bold).

Constructor settings:
dist: 1.
num: 9.
mid: 5.

7.4 Member functions for class DoubleLogCoord

DoubleLogCoord()
DoubleLogCoord(double xmin,double xmax,double ymin,double ymax)

Description:
Constructor for class DoubleLogCoord.

Parameters:
xmin: lower x-bound of picture (default 1).
xmax: upper x-bound of picture (default 100).
ymin: lower y-bound of picture (default 1).
ymax: upper y-bound of picture (default 100).

void axisCross(triple point)

Description:
Set crossing point for coordinate axis.

Parameters:
point: crossing point.

Constructor settings:
(xmin,ymin).

void axisBounds(triple upperRight)
void axisBounds(triple lowerLeft,triple upperRight)

Description:
Set bounds of axis in coordinate system. Set by constructor to bounds of picture.

Parameters:
lowerLeft: lower left corner of axis.
upperRight: upper right corner of axis.

Constructor settings:
lowerLeft: (xmin,ymin).
upperRight: (xmax,ymax).

void xMarks(double first,double last)

Description:
Set bounds of marks on horizontal logarithmic axis.

Parameters:
first: lower bound of marks.
last: upper bounds marks.

Constructor settings:
first: xmin.
last: xmax.

void yMarks(double first,double last)

Description:
Set bounds of marks on vertical logarithmic axis.

Parameters:
first: lower bound of marks.
last: upper bounds marks.

Constructor settings:
first: ymin.
last: ymax.

void showArrow(double extra)
void showArrow(bool bx,double extrax,bool by,double extray)

Description:
Draw axis with or without an arrow and add extra length to logarithmic axis.

Parameters:
extra: add extra length (in pt) and draw arrow (for both axis).
extrax: add extra length (in pt) to horizontal axis and draw arrow.
extray: add extra length (in pt) to vertical axis and draw arrow.

Constructor settings:
No arrows.

void logStyle(int style)
void logStyle(Mark* mrk)
void logStyle(int stylex,int styley)
void logStyle(Mark* mrkx,Mark* mrky)
void logStyle(int stylex,Mark* mrky)
void logStyle(Mark* mrkx,int styley)

Description:
Set style of axis. *Mark* is defined as:

```
class Mark
{
  public:
   double position; // position of main mark
   char*  label;    // label to print at main mark
   int    numsub;   // number of submarks until next main mark
};
```

Parameters:
style: standard style (1...6) to be used for both axis.
stylex: standard style (1...6) to be used for horizontal axis.
styley: standard style (1...6) to be used for vertical axis.
mrk: pointer to an array of marks describinging both axis.
mrkx: pointer to an array of marks describinging horizontal axis.
mrky: pointer to an array of marks describinging vertical axis.

Constructor settings:
Standard style 4 for both axis.

void labelAttr(char* attr)
void labelAttr(char* attrx,char* attry)

Description:
Label attribute to use if label is not specified (**NULL**).

Parameters:
attr: attribute string for both axis.
attrx: attribute string for horizontal axis.
attry: attribute string for vertical axis.

Constructor settings:
attr = "\\scriptsize".

void grid()

Description:
Add a grid to the coordinate system. Draw coordinate axis in bold unless otherwise specified.

Parameters:
None.