

Extension d'ePiX

Svend Daugaard Pedersen*

4 novembre 2006

*Traduction française par le T_EXnicien de surface, 2007-03-06

Table des matières

1	Introduction	3
2	Installation	3
3	Repère cartésien	3
4	Repères logarithmiques	9
4.1	Repère à une seule échelle logarithmique	9
4.2	Repère à double échelle logarithmique	13
4.3	Graphes dans les repères logarithmiques	13
4.4	Imprimer les axes de coordonnées	14
5	Figures hachurées	15
5.1	Aires hachurées	15
5.2	Polygone hachuré	16
6	Fixer l'épaisseur des traits	17
7	Références d'ePiX_ext	17
7.1	Fonctions membres pour tous les repères	17
7.2	Fonctions membres de la classe CartesianCoord	20
7.3	Fonctions membres de la classe SingleLogCoord	23
7.4	Fonctions membres de la classe DoubleLogCoord	26

1 Introduction

ePiX_ext est un ensemble d'extensions de la bibliothèque ePiX fournissant des repères cartésiens et logarithmiques (autant simple que double) avec un grand nombre de paramètres que l'utilisateur peut fixer. De plus il fournit les outils pour dessiner des polygones hachurés ainsi que des aires hachurées déterminées par deux courbes représentatives de fonctions.

Il faut connaître un peu ePiX pour pouvoir utiliser cette extension.

2 Installation

Pour installer ePiX_ext, utilisez l'option `--with-contrib` lors de la configuration du package principal. La bibliothèque sera compilée séparément (`libepixext.a`) et la bibliothèque et le fichier d'entête seront installés automatiquement quand vous lancerez le `make install`.

Pour utiliser cette extension dans un fichier source, il faut inclure explicitement l'entête dans le préambule du fichier source :

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

using namespace ePiX;
using namespace ePiX_contrib;
```

Depuis la version 1.0.7 d'ePiX, il faut lancer `epix -lepixext toto.cc` pour compiler un fichier `toto.cc` qui utilise `epix_ext`. (Le \TeX nicien de surface)

3 Repère cartésien

La déclaration

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

crée un repère cartésien avec x variant dans $[-3, 5]$ et y dans $[-2, 6]$. Pour tracer le repère utilisant la présentation normale déterminée par le constructeur de *CartesianCoord* il faut exécuter la fonction membre *draw* dans *cs* :

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

Voici un exemple d'un programme C++ complet traçant un tel repère cartésien :

```

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

using namespace ePiX;
using namespace ePiX_contrib;

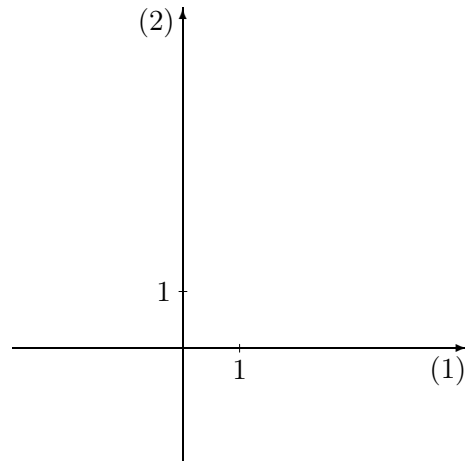
int main()
{
    CartesianCoord cs(-3,5,-2,6);

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

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

    return 0;
}

```



Pour obtenir plus de points de division vous devez exécuter la fonction membre *xMark()* ou *yMark()* en donnant en paramètres la position de la première coche et le nombre de coches voulues.

```

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

using namespace ePiX;
using namespace ePiX_contrib;

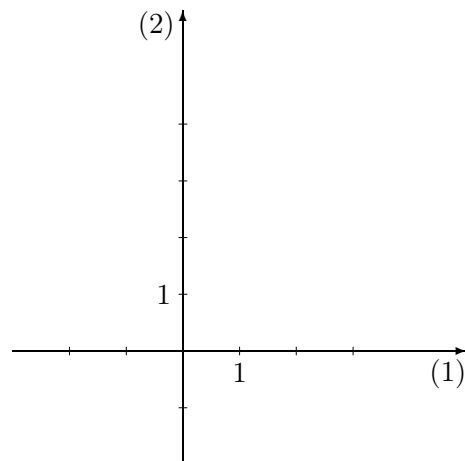
int main()
{
    CartesianCoord cs(-3,5,-2,6);

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

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

    return 0;
}

```



Les fonctions membres *xMark()* et *yMarks()* acceptent un troisième paramètre qui fixe la distance entre deux coches. La valeur par défaut est 1.

Les fonctions membres *xLabels()* et *yLabels()* positionnent sur les axes d'autres étiquettes que celles par défaut. Ces fonctions membres ont un, deux ou trois paramètres. Le premier (obligatoire) est une chaîne terminée par NULL de pointeurs sur chaîne. Le deuxième paramètre sert à fixer la position de la première étiquette (par défaut c'est la première coche) et le troisième fixe la distance entre deux étiquettes (par défaut la distance entre deux coches).

```

#include "epix.h"
#include "epix_ext.h"
using namespace ePiX;
using namespace ePiX_contrib;

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

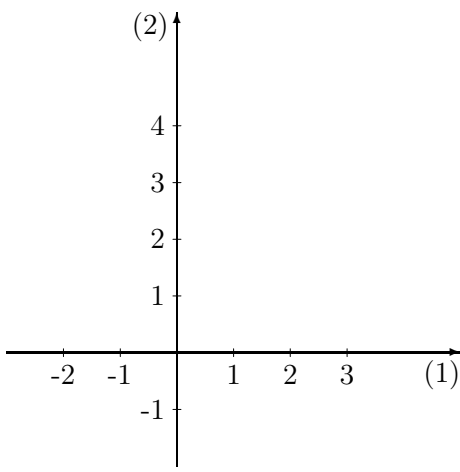
int main()
{
    CartesianCoord cs(-3,5,-2,6);

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

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

    return 0;
}

```



Notez la chaîne vide à la position 0 pour éviter la collision entre l'étiquette et l'axe.

La longueur et la position des coches ainsi que la position des étiquettes peuvent être précisées. Les valeurs de position sont POSITIVE, NEGATIVE et (pour les coches) CENTER. Les coches sont placées par la fonction membre *markLayout()* qui a quatre paramètres : la longueur (en pt) de la coche, la position pour l'axe des x et les mêmes informations pour l'axe des y . La position des étiquettes est fixée avec la fonction membre *labelPos()* qui admet un ou deux paramètres. La version à un paramètre fixe la position comme identique sur les deux axes.

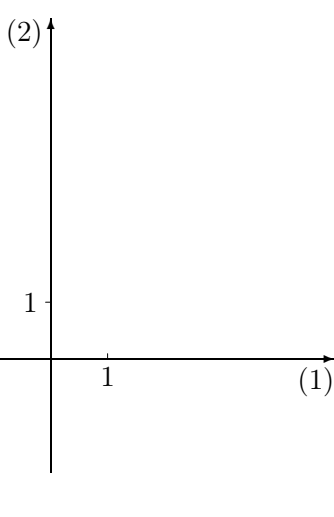
```

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

using namespace ePiX;
using namespace ePiX_contrib;

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;
}

```



Les noms des axes et les positions des noms peuvent être fixés avec les fonctions membres *xName()* et *yName()*. Les paramètres sont le nom et la position.

```

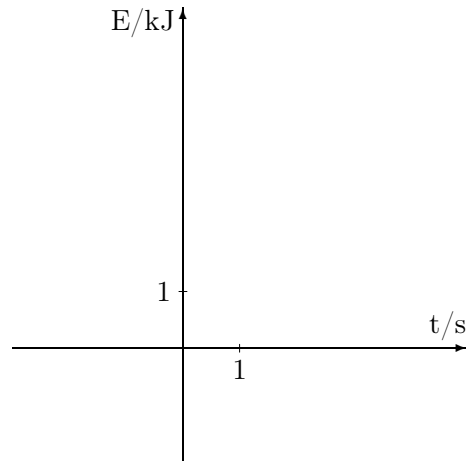
#include "epix.h"
#include "epix_ext.h"

using namespace ePiX;
using namespace ePiX_contrib;
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;
}

```



La position CENTER des noms place les noms au bout des axes. Si vous utilisez cette position vous aurez certainement besoin de raccourcir l'axe sinon le nom sera écrit en dehors des limites de l'image. On peut réaliser cela à l'aide de la fonction membre *axisBounds()*. Elle accepte un ou deux paramètres. La version à un paramètre fixera le « coin » supérieur droit (très probablement ce dont vous avez besoin), la version à deux paramètres fixe les coins inférieur gauche et supérieur droit.

```

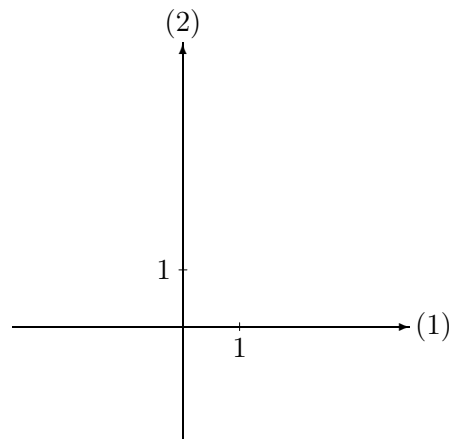
#include "epix.h"
#include "epix_ext.h"

using namespace ePiX;
using namespace ePiX_contrib;
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;
}

```



On peut préciser le point d'intersection des axes (par défaut c'est $(0,0)$) et on peut demander un axe « brisé » :

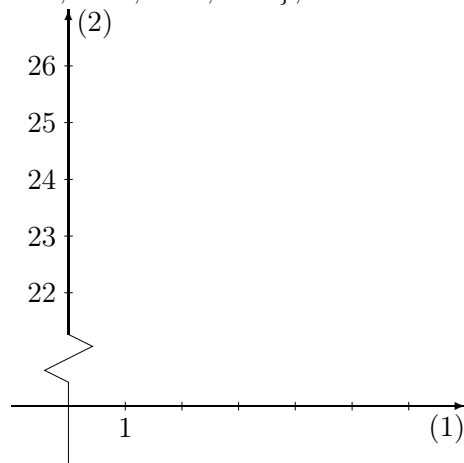
```

#include "epix.h"
#include "epix_ext.h"
using namespace ePiX;
using namespace ePiX_contrib;
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;
}

```



On peut ajouter un papier quadrillé en appelant la fonction membre *grid()*.

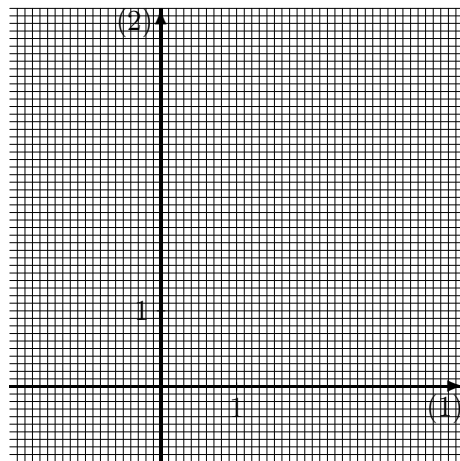
```

#include "epix.h"
#include "epix_ext.h"
using namespace ePiX;
using namespace ePiX_contrib;
int main()
{
    CartesianCoord cs(-2,4,-1,5);
    picture(P(12,12));
    unitlength("5mm");
    begin();

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

    end();
    return 0;
}

```

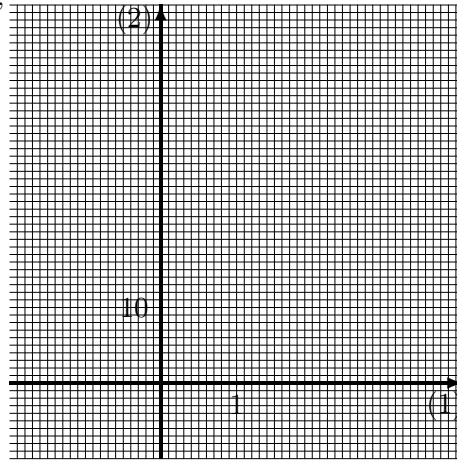


On fixe la position du papier quadrillé avec la fonction membre *gridPosition()* dont le paramètre est un point d'intersection des lignes grasses (par défaut c'est le point d'intersection des axes). La densité des lignes du quadrillage est déterminé par l'appel de la fonction membre *gridDensity()* (cf. la section Références d'ePiX_ext, page 17, pour plus de détails)

```
#include "epix.h"
#include "epix_ext.h"
using namespace ePiX;
using namespace ePiX_contrib;
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 Repères logarithmiques

Les repères logarithmiques sont construits de manière semblable aux repères cartésiens.

4.1 Repère à une seule échelle logarithmique

Un repère à une seule échelle logarithmique a sur l'axe horizontal une échelle linéaire et sur l'axe verticale une échelle logarithmique.

L'instruction

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

crée un repère dont l'axe des x , linéaire, s'étend de 0 à 10 et l'axe des y , logarithmique, s'étend de 1 à 100. Pour tracer le repère avec la présentation standard fixée par le constructeur *SingleLogCoord* exécuter la fonction membre *draw* dans *cs* :

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

Voici un exemple d'un programme C++ complet pour réaliser un tel repère standard :

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

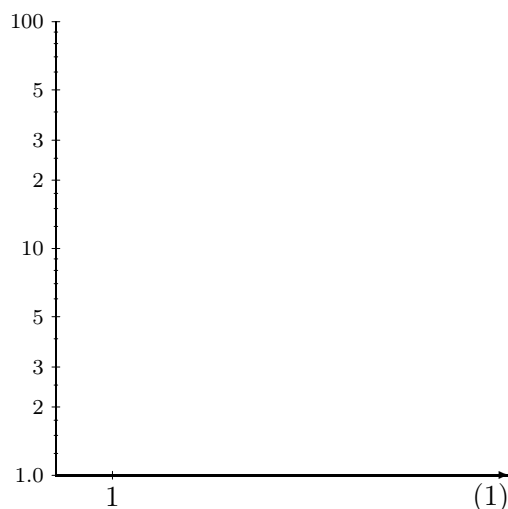
using namespace ePiX;
using namespace ePiX_contrib;

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

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

    sc.draw();

    end();
    return 0;
}
```



Il n'est pas nécessaire que l'échelle logarithmique ne contienne que des dizaines pleines. Le programme C++ suivant sert à créer un repère avec une dizaine « et demi » de 0,1 à 5.

```

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

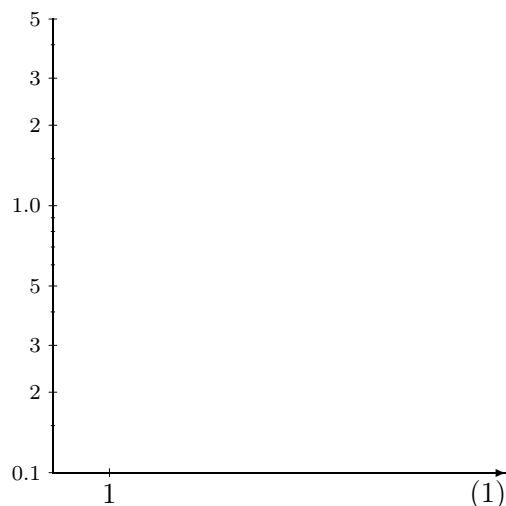
using namespace ePiX;
using namespace ePiX_contrib;

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

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

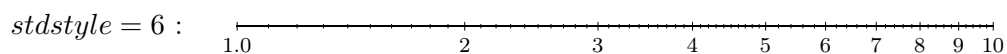
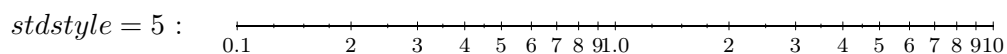
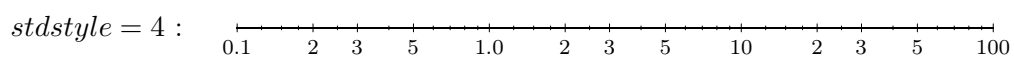
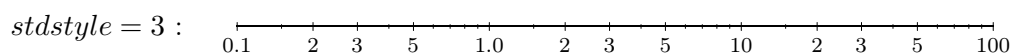
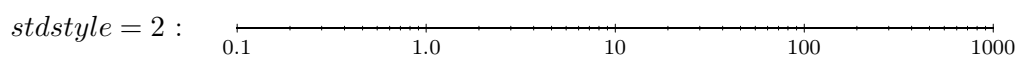
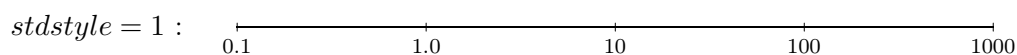
    end();
    return 0;
}

```



Dans ce cas on utilise le style logarithmique standard numéro 3 pour définir les division de l'échelle logarithmique.

Il y a six styles standards donnant un nombre croissant de points de division :



Le style *stdstyle* par défaut vaut 4.

Le début de chaque dizaine est étiqueté, comme on le voit, dans l'intervalle 0,01 à 1 000. En dehors de cet intervalle on utilise les puissances de 10.

Pour personnaliser les marques et les étiquettes appelez la fonction membre *logStyle()* comme dans ce qui suit :

```

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

using namespace ePiX;
using namespace ePiX_contrib;

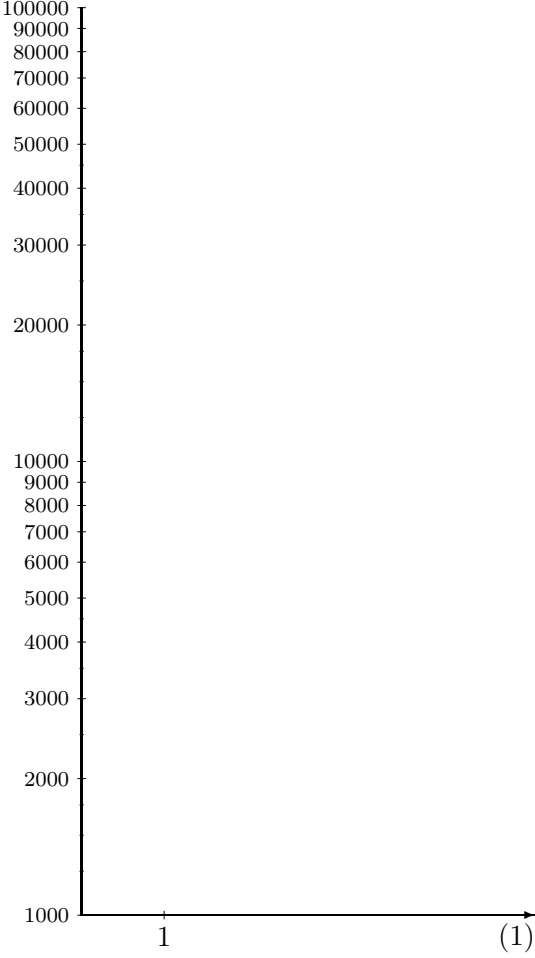
Mark mrk[] =
{
  {1, "1000", 3}, {2, "2000", 1},
  {3, "3000", 1}, {4, "4000", 1},
  {5, "5000", 0}, {6, "6000", 0},
  {7, "7000", 0}, {8, "8000", 0},
  {9, "9000", 0}, {1, "10000", 3},
  {2, "20000", 1}, {3, "30000", 1},
  {4, "40000", 1}, {5, "50000", 0},
  {6, "60000", 0}, {7, "70000", 0},
  {8, "80000", 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;
}

```



La variable *mrk* est une table de structures contenant chacune trois champs. Le premier contient l'emplacement de la coche (emplacement dans la dizaine véritable). Le suivant contient l'étiquette à imprimer. Le troisième est le nombre de subdivisions suivant la coche. La table s'achève avec la 10^e position.

Autre exemple

```

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

using namespace ePiX;
using namespace ePiX_contrib;

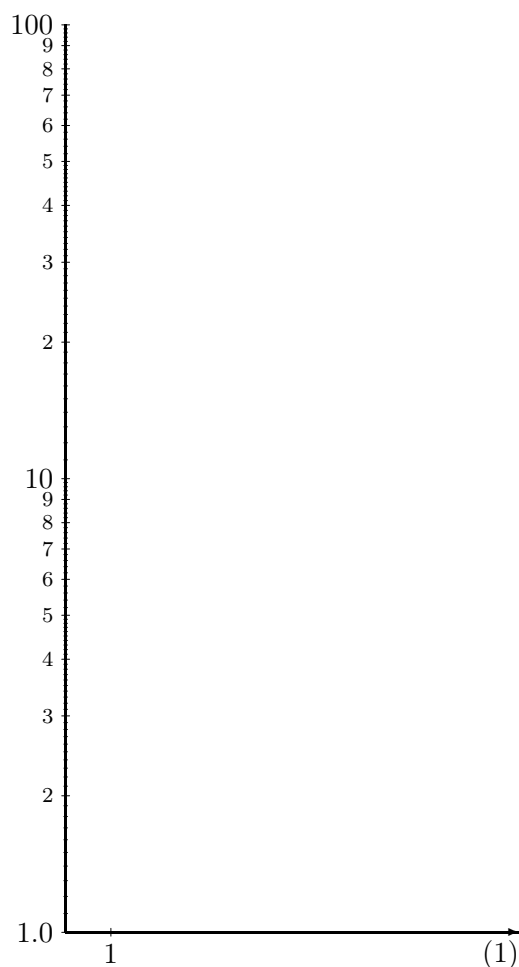
Mark mrk [] =
{
  {1, NULL, 9},
  {2, "\\scriptsize 2", 9},
  {3, "\\scriptsize 3", 9},
  {4, "\\scriptsize 4", 9},
  {5, "\\scriptsize 5", 4},
  {6, "\\scriptsize 6", 4},
  {7, "\\scriptsize 7", 4},
  {8, "\\scriptsize 8", 4},
  {9, "\\scriptsize 9", 4},
  {10, NULL, 0}
};

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;
}

```



Ici *mrk* est une table s'étendant sur une dizaine. Elle sert pour les deux dizaines de l'axe. L'étiquette NULL à la 1^{re} position a pour effet de faire imprimer la valeur courante de la dizaine (ici 1,0, 10 et 100).

La fonction membre *labelAttr()* peut servir à fixer n'importe quel attribut, par exemple `\it` ou `\tiny`. Mais notez que vous aurez besoin d'utiliser une double barre oblique inverse dans une chaîne C++ puisque, en C++, (comme dans `TeX`) une `boi` introduit une commande. Dans l'exemple, l'attribut prend pour valeur la chaîne vide afin que les nombres du début de chaque dizaine soit imprimé en taille normale (par défaut l'attribut est « `\\scriptsize` »).

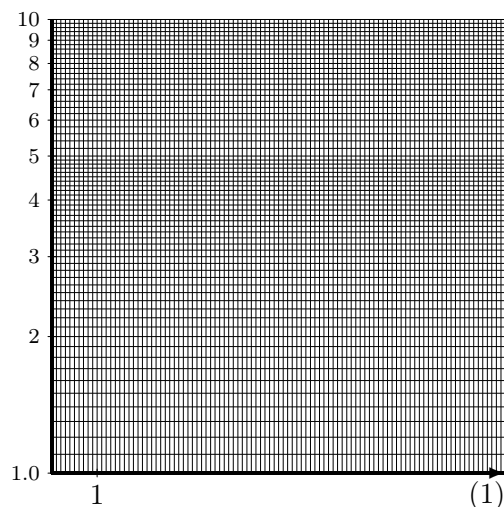
Comme dans le cas du repère cartésien, on peut ajouter un papier quadrillé :

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

using namespace ePiX;
using namespace ePiX_contrib;
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 Repère à double échelle logarithmique

Un repère à double échelle logarithmique est utilisé presque comme un repère à une seule échelle logarithmique. Il suffit de changer *Single* en *Double*.

Voici un exemple :

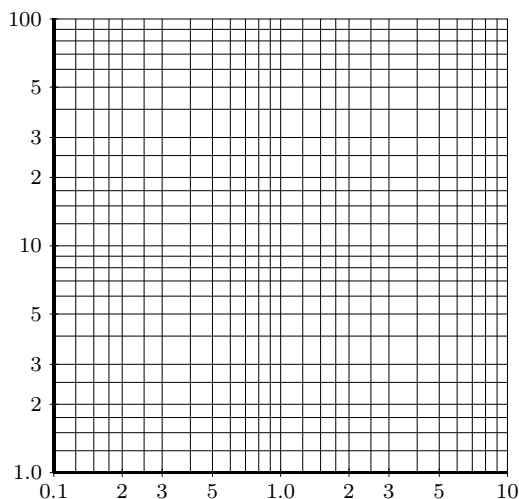
```
#include "epix.h"
#include "epix_ext.h"

using namespace ePiX;
using namespace ePiX_contrib;
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 Graphes dans les repères logarithmiques

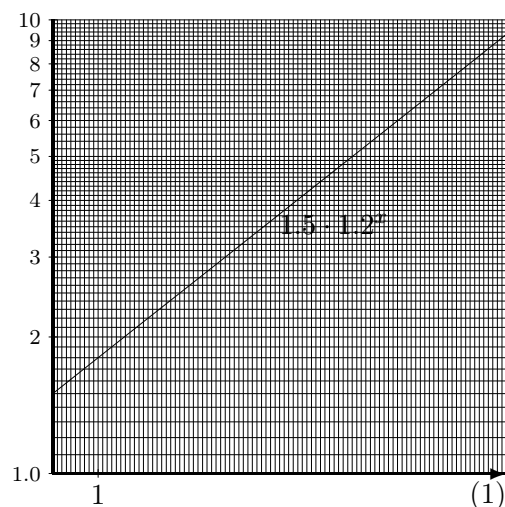
Dans les repères, on utilise les fonctions membres *plot()*, *adplot()*, *label()* et *marker()* de la même manière que les fonctions correspondantes d'ePiX. Mais on peut les utiliser aussi dans les repères logarithmiques.

```

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

using namespace ePiX;
using namespace ePiX_contrib;
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),
            "$1.5\\cdot 1.2^x$",br);
    end();
    return 0;
}

```



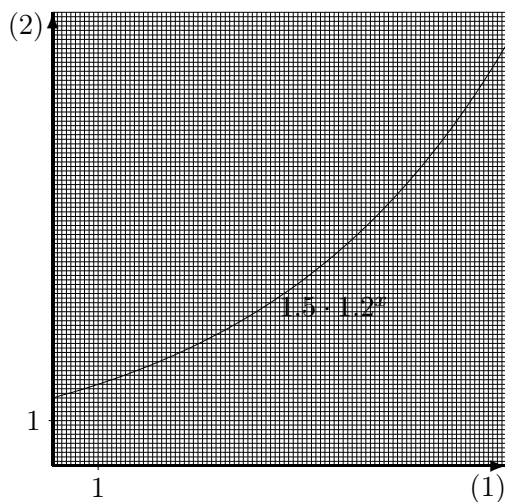
En changeant *SingleLog* en *Cartesian* (et avec quelques modifications naturelles), on obtient :

```

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

using namespace ePiX;
using namespace ePiX_contrib;
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),
            "$1.5\\cdot 1.2^x$",br);
    end();
    return 0;
}

```



4.4 Imprimer les axes de coordonnées

On peut imprimer les axes de coordonnées seuls. La sortie du programme ci-dessous montre la théorie qui se tient derrière les échelles logarithmiques :

```

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

using namespace ePiX;
using namespace ePiX_contrib;
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;

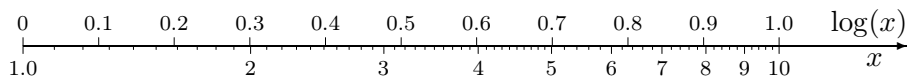
  bounding_box(P(0,0),P(1,1.17));
  picture(P(10,1.5));
  unitlength("1cm");
  begin();

  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;
}

```



Voyez *epix_ext.h* pour une description des champs de la structure des axes.

5 Figures hachurées

5.1 Aires hachurées

Les commandes pour hachurer une aire comprise entre deux courbes (ou une courbe et l'axe des x) sont créées par l'appel des fonctions membres :

```

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)

```

Les deux premiers paramètres *angle* et *dist* fixent l'angle des traits de hachure et la distance entre deux traits. Les paramètres suivants sont la ou les fonctions, les extrémités

et le nombre de points à utiliser pour tracer les courbes.

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

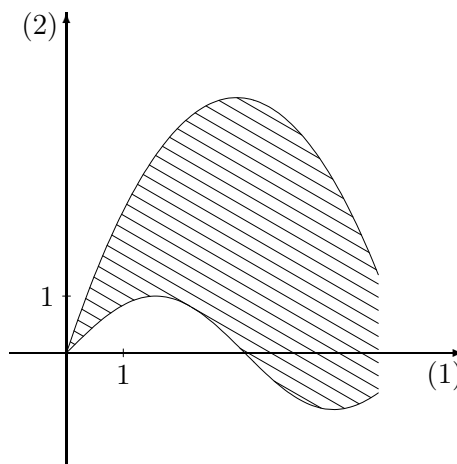
using namespace ePiX;
using namespace ePiX_contrib;

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 Polygone hachuré

La fonction membre *hatchArea()* s'appuie sur des routines de hachurage de polygones. Ces derniers sont produits par les procédures :

```
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, ...);
```

La différence entre *hatch_polygon()* et *hatch()* est que *hatch()* ne trace pas le polygone. Les deux premiers paramètres ont la même signification que dans la fonction membre *hatchArea()*. Le paramètre suivant est le nombre de sommets du polygone puis viennent les sommets.


```

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

using namespace ePiX;
using namespace ePiX_contrib;

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");

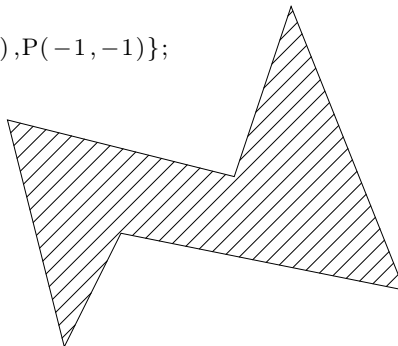
  begin();

  hatch_polygon(45,0.2,6, corners);

  end();

  return 0;
}

```



Comme on le voit dans l'exemple, il n'est pas nécessaire que le polygone soit convexe. Mais le résultat est imprévisible si quelques côtés se coupent.

6 Fixer l'épaisseur des traits

Les épaisseurs des traits servant à tracer un axe (gras et normal), les quadrillages gras, moyen et fin peuvent être changées en donnant de nouvelles valeurs à certaines variables. Ces variables et leurs valeurs par défaut sont :

```

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 Références d'ePiX_ext

7.1 Fonctions membres pour tous les repères

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

Description:

Fixe les bornes du repère.

Paramètres:

xmin : borne inférieure de l'image pour x
 xmax : borne supérieure de l'image pour x
 ymin : borne inférieure de l'image pour y
 ymax : borne supérieure de l'image pour y

void draw()

Description:

Trace le repère.

Paramètres:

aucun.

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

Description:

Trace un axe ou pas.

Paramètres:

b : si vrai trace les deux axes

bx : si vrai trace l'axe des x

by : si vrai trace l'axe des y

Valeurs fixées par le constructeur:

vrai pour les deux axes.

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

Description:

Trace les axes avec ou sans flèche.

Paramètres:

b : si vrai trace une flèche sur les deux axes

bx : si vrai trace une flèche sur l'axe des x

by : si vrai trace une flèche sur l'axe des y

Valeurs fixées par le constructeur:

vrai pour les deux axes.

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

Description:

Place le nom au bout de l'axe des x

Paramètres:

nom : nom de l'axe des x

pos : position du nom (POSITIVE, NEGATIVE ou CENTER).

Valeurs fixées par le constructeur:

nom : "(1)" pour une échelle linéaire et **NULL** pour une échelle logarithmique

pos : NEGATIVE.

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

Description:

Place le nom au bout de l'axe des y

Paramètres:

nom : nom de l'axe des y

pos : position du nom (POSITIVE, NEGATIVE ou CENTER).

Valeurs fixées par le constructeur:

nom : "(2)" pour une échelle linéaire et **NULL** pour une échelle logarithmique
 pos : **NEGATIVE**.

void labelPos(Position pos)

void labelPos(Position xpos,Position ypos)

Description:

Fixe la position des étiquettes des coches sur les axes.

Paramètres:

pos : position pour les étiquettes des deux axes.

xpos : position pour les étiquettes de l'axe des x .

ypos : position pour les étiquettes de l'axe des y .

Valeurs fixées par le constructeur:

NEGATIVE pour les deux axes.

void markLayout(double longueur,Position pos=CENTER**)**

void markLayout(double lx,Position px,double ly,Position py)

Description:

Fixe la longueur et la position des coches sur les axes

Paramètres:

longueur : longueur (en pt) des coches sur les deux axes.

pos : position des coches sur les deux axes.

lx : longueur (en pt) des coches sur l'axe des x

px : position des coches sur l'axe des x .

ly : longueur (en pt) des coches sur l'axe des y

py : position des coches sur l'axe des y .

Valeurs fixées par le constructeur:

longueur : 3 pt.

pos : **CENTER**.

void boldAxis(bool b=true**)**

void boldAxis(bool bx,bool by)

Description:

trace les axes avec un trait gras.

Paramètres:

b : si vrai trace les deux axes en gras.

bx : si vrai trace l'axe des x en gras.

by : si vrai trace l'axe des y en gras.

Valeurs fixées par le constructeur:

vrai pour les deux axes si on a sélectionné *grid* (quadrillage), **faux** sinon.

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

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

Description:

Fonctionne comme le adplot d'ePiX mais tient compte du type (cartésien ou logarithmique) du repère.

Paramètres:

- f : fonction dont il faut tracer le graphe.
- xmin : borne inférieure du graphique pour les x .
- xmax : borne supérieure du graphique pour les x .
- n : nombre de points utilisés pour dessiner (segments de droite entre deux points).

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:

Fonctionne comme le *label* d'ePiX mais tient compte du type (cartésien ou logarithmique) du repère.

Paramètres:

- base : point de base de l'étiquette.
- offset : déplacement (en pt) de l'étiquette.
- text : texte à écrire.
- pos : position relative au point de base (c, r, tr, rt, t, tl, lt, l, bl, lb, b, br, rb).

7.2 Fonctions membres de la classe CartesianCoord

CartesianCoord()

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

Description:

Constructeur de la classe CartesianCoord.

Paramètres:

- xmin : borne inférieure du graphique pour les x (défaut -1).
- xmax : borne supérieure du graphique pour les x (défaut 10).
- ymin : borne inférieure de l'image pour y (défaut -1).
- ymax : borne supérieure de l'image pour y (défaut 10).

void axisCross(triple point)

Description:

Fixe le point d'intersection des axes.

Paramètres:

- point : point d'intersection des axes.

Valeurs fixées par le constructeur:

(0,0).

void axisBounds(triple supDroit)

void axisBounds(triple infGauche, triple supDroit)

Description:

Fixe les bornes des axes dans le repère.

Paramètres:

- infGauche : coin inférieur gauche des axes.
- supDroit : coin supérieur droit des axes.

Valeurs fixées par le constructeur:

Bornes du repère.

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

Description:

Fixe le nombre et la position des coches sur l'axe des x .

Paramètres:

premier : position de la première coche.

num : nombre de coches.

dist : distance entre deux coches (défaut 1,0).

Valeurs fixées par le constructeur:

premier : 1.

num : 1.

dist : 1.

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

Description:

Fixe le nombre et la position des coches sur l'axe des y .

Paramètres:

premier : position de la première coche.

num : nombre de coches.

dist : distance entre deux coches (défaut 1,0).

Valeurs fixées par le constructeur:

premier : 1.

num : 1.

dist : 1.

void xLabels(char etiq)**
void xLabels(char etiq,double premier)**
void xLabels(char etiq,double premier,double dist)**

Description:

Étiquettes à placer sur l'axe des x .

Paramètres:

etiq : les étiquettes rangées dans un vecteur de pointeurs sur chaîne terminée par un **NULL**.

premier : position de l'étiquette (par défaut position de la première coche).

dist : distance entre étiquettes (par défaut distance entre deux coches).

Valeurs fixées par le constructeur:

etiq : {"1",**NULL**}.

premier : première coche.

dist : distance entre coches.

void yLabels(char etiq)**
void yLabels(char etiq,double premier)**
void yLabels(char etiq,double premier,double dist)**

Description:

Étiquettes à placer sur l'axe des y .

Paramètres:

eti_q : les étiquettes rangées dans un vecteur de pointeurs sur chaîne terminé par un **NULL**.

premier : position de l'étiquette (par défaut position de la première coche).

dist : distance entre étiquettes (par défaut distance entre deux coches).

Valeurs fixées par le constructeur:

eti_q : {"1",**NULL**}.

premier : première coche.

dist : distance entre coches.

void xBroken()

Description:

Produit un axe des x brisé (pour montrer que les axes ne se coupent pas en zéro).

Paramètres:

Aucun.

void yBroken()

Description:

Produit un axe des y brisé (pour montrer que les axes ne se coupent pas en zéro).

Paramètres:

Aucun.

void grid()

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

Description:

Ajoute un quadrillage au repère. Trace les axes de coordonnées en gras si le contraire n'est pas demandé.

Paramètres:

xmin : bord gauche du quadrillage (par défaut le bord gauche du repère).

xmax : bord droit du quadrillage (par défaut le bord droit du repère).

ymin : bord inférieur du quadrillage (par défaut le bord inférieur du repère).

ymax : bord supérieur du quadrillage (par défaut le bord supérieur du repère).

void gridPosition(triple CroixGras)

Description:

Fixe la position du quadrillage dans le repère.

Paramètres:

CroixGras : un point d'intersection des lignes grasses.

Valeurs fixées par le constructeur:

Point d'intersection des axes.

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:

Fixe la densité des lignes du quadrillage.

Paramètres:

dist : distance entre les lignes du quadrillage.
 num : nombre de lignes fines (9 par défaut).
 mid : la ligne fine à tracer en demi-gras (0 par défaut c.-à-d. pas de demi-gras).
 xd,xn,xm : dist, num et mid pour l'axe des x .
 yd,yn,ym : dist, num et mid pour l'axe des y .

Valeurs fixées par le constructeur:

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

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

Description:

Surface hachurée entre une courbe et l'axe des x .

Paramètres:

v : angle entre l'axe des x et les traits de hachure.
 d : distance entre traits de hachure.
 f : fonction à représenter.
 a : bord gauche de la surface.
 b : bord droit de la surface.
 n : nombre de points utilisés pour tracer la courbe (cf. plot et adplot).

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

Description:

Hachure la surface comprise entre deux courbes.

Paramètres:

v : angle entre l'axe des x et les traits de hachure.
 d : distance entre traits de hachure.
 f : fonction définissant la courbe supérieure.
 g : fonction définissant la courbe inférieure.
 a : bord gauche de la surface.
 b : bord droit de la surface.
 n : nombre de points utilisés pour tracer la courbe (cf. plot et adplot).

7.3 Fonctions membres de la classe SingleLogCoord

SingleLogCoord()

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

Description:

Constructeur de la classe SingleLogCoord.

Paramètres:

xmin : borne inférieure du graphique pour les x (défaut -1).
 xmax : borne supérieure du graphique pour les x (défaut 10).
 ymin : borne inférieure de l'image pour y (défaut 1).
 ymax : borne supérieure de l'image pour y (défaut 100).

void axisCross(triple point)

Description:

Fixe le point d'intersection des axes de coordonnées. Fixé à (xmin, ymin) par le constructeur.

Paramètres:

point : point d'intersection.

Valeurs fixées par le constructeur:

(xmin,ymin).

void axisBounds(triple supDroit)

void axisBounds(triple infGauche, triple supDroit)

Description:

Fixe les bornes des axes du repère. Le constructeur les rend égales aux bornes de la figure.

Paramètres:

infGauche : coin inférieur gauche des axes.

supDroit : coin supérieur droit des axes.

Valeurs fixées par le constructeur:

lowerLift : (xmin,ymin).

supDroit : (xmax,ymax).

void xMarks(double premier, int num)

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

Description:

Fixe le nombre et la position des coches sur l'axe des x .

Paramètres:

premier : position de la première coche.

num : nombre de coches.

dist : distance entre deux coches (défaut 1,0).

Valeurs fixées par le constructeur:

premier : 1.

num : 1.

dist : 1.

void yMarks(double premier, double dernier)

Description:

Fixe les bornes des coches sur l'axe portant l'échelle logarithmique.

Paramètres:

premier : valeur inférieure pour les coches.

dernier : valeur supérieure pour les coches.

Valeurs fixées par le constructeur:

premier : ymin.

dernier : ymax.

void xLabels(char etiq)**

void xLabels(char etiq, double premier)**

void xLabels(char etiq, double premier, double dist)**

Description:

Étiquettes à placer sur l'axe des x .

Paramètres:

etiq : les étiquettes rangées dans un vecteur de pointeurs sur chaîne terminé par un **NULL**.

premier : position de l'étiquette (par défaut position de la première coche).

dist : distance entre étiquettes (par défaut distance entre deux coches).

Valeurs fixées par le constructeur:

etiq : {"1",**NULL**}.

premier : première coche.

dist : distance entre coches.

void showArrow(bool b)

void showArrow(double extra)

void showArrow(bool b,double extra)

Description:

Trace les axes avec ou sans flèche et agrandit l'axe logarithmique.

void showArrow(bool b) n'affecte que l'axe des x .

void showArrow(double extra) n'affecte que l'axe des y .

void showArrow(bool b,double extra) affecte les deux axes.

Paramètres:

b : si vrai trace une flèche sur l'axe des x .

extra : ajoute une longueur (en pt) à l'axe des y et trace une flèche.

Valeurs fixées par le constructeur:

Flèche sur l'axe des x , pas de flèche sur l'axe des y .

void xBroken()

Description:

Produit un axe des x brisé (pour montrer que les axes ne se coupent pas en zéro).

Paramètres:

Aucun.

Valeurs fixées par le constructeur:

Pas d'axe brisé.

void logStyle(int style)

void logStyle(Mark* mrk)

Description:

Fixe le style de l'échelle logarithmique. *Mark* est défini comme suit :

```
class Mark
{
public:
    double position; // position de la coche principale
    char* label; // étiquette à imprimer à la coche principale
    int numsub; // nbr de sous-coches avant la coche principale suivante
};
```

Paramètres:

style : numéro de style standard (1...6).

mrk : pointeur sur un vecteur de coches décrivant l'échelle logarithmique.

Valeurs fixées par le constructeur:

Style standard numéro 4.

void labelAttr(char* attr)

Description:

Attribut d'étiquette à utiliser si *label* n'est pas spécifié (**NULL**).

Paramètres:

attr : chaîne d'attribut.

Valeurs fixées par le constructeur:

attr = "\\scriptsize".

void grid()

void grid(double xmin,double xmax)

Description:

Ajoute un quadrillage au repère. Trace les axes de coordonnées en gras si le contraire n'est pas demandé.

Paramètres:

xmin : bord gauche du quadrillage (par défaut le bord gauche du repère).

xmax : bord droit du quadrillage (par défaut le bord droit du repère).

Valeurs fixées par le constructeur:

xmin : bord gauche du repère.

xmax : bord droit du repère.

void gridPosition(double crossX)

Description:

Fixe la position des lignes du quadrillage perpendiculaires à l'axe des *x*.

Paramètres:

crossX : position d'une ligne grasse.

Valeurs fixées par le constructeur:

position de l'axe logarithmique.

void gridDensity(double dist)

void gridDensity(double dist,int num)

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

Description:

Densité des lignes du quadrillage perpendiculaires à l'axe des *x*.

Paramètres:

dist : distance entre deux lignes du quadrillage.

num : nombre de lignes fines (9 par défaut).

mid : la ligne fine à tracer en demi-gras (0 par défaut c.-à-d. pas de demi-gras).

Valeurs fixées par le constructeur:

dist : 1.

num : 9.

mid : 5.

7.4 Fonctions membres de la classe DoubleLogCoord

DoubleLogCoord()

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

Description:

Constructor for class DoubleLogCoord.

Paramètres:

xmin : borne inférieure du graphique pour les x (défaut 1).
 xmax : borne supérieure du graphique pour les x (défaut 100).
 ymin : borne inférieure de l'image pour y (défaut 1).
 ymax : borne supérieure de l'image pour y (défaut 100).

void axisCross(triple point)

Description:

Fixe le point d'intersection des axes de coordonnées.

Paramètres:

point : point d'intersection.

Valeurs fixées par le constructeur:

(xmin,ymin).

void axisBounds(triple supDroit)

void axisBounds(triple infGauche, triple supDroit)

Description:

Fixe les bornes des axes du repère. Le constructeur les rend égales aux bornes de la figure.

Paramètres:

infGauche : coin inférieur gauche des axes.

supDroit : coin supérieur droit des axes.

Valeurs fixées par le constructeur:

infGauche : (xmin,ymin).

supDroit : (xmax,ymax).

void xMarks(double premier, double dernier)

Description:

Fixe les bornes des coches sur l'axe logarithmique horizontal.

Paramètres:

premier : borne inférieure des coches.

dernier : borne supérieure des coches.

Valeurs fixées par le constructeur:

premier : xmin.

dernier : xmax.

void yMarks(double premier, double dernier)

Description:

Fixe les bornes des coches sur l'axe logarithmique horizontal.

Paramètres:

premier : borne inférieure des coches.

dernier : borne supérieure des coches.

Valeurs fixées par le constructeur:

premier : ymin.

dernier : ymax.

void showArrow(double extra)

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

Description:

Trace les axes avec ou sans flèche et agrandit les axes logarithmiques.

Paramètres:

extra : ajoute une longueur (en pt) et trace une flèche (pour les deux axes).

extrax : ajoute une longueur (en pt) à l'axe des x et trace une flèche.

extray : ajoute une longueur (en pt) à l'axe des y et trace une flèche.

Valeurs fixées par le constructeur:

Pas de flèche.

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:

Fixe le style des axes. *Mark* est défini comme suit :

```
class Mark
{
    public:
        double position; // position de la coche principale
        char* label; // étiquette à imprimer à la coche principale
        int numsub; // nombre de sous-coches avant la coche suivante
};
```

Paramètres:

style : numéro de style standard (1...6) pour les deux axes.

stylex : numéro de style standard (1...6) pour l'axe des x .

styley : numéro de style standard (1...6) pour l'axe des y .

mrk : pointeur sur un vecteur de coches décrivant les deux axes.

mrkx : pointeur sur un vecteur de coches décrivant l'axe des x .

mrky : pointeur sur un vecteur de coches décrivant l'axe des y .

Valeurs fixées par le constructeur:

Style standard numéro 4 pour les deux axes.

void labelAttr(char* attr)

void labelAttr(char* attrx,char* attry)

Description:

Attribut d'étiquette à utiliser si *label* n'est pas spécifié (**NULL**).

Paramètres:

attr : chaîne d'attribut pour les deux axes.

attrx : chaîne d'attribut pour l'axe des x .

attry : chaîne d'attribut pour l'axe des y .

Valeurs fixées par le constructeur:

attr = "\\scriptsize".

void grid()

Description:

Ajoute un quadrillage au repère. Trace les axes de coordonnées en gras si le contraire n'est pas demandé.

Paramètres:

Aucun.