Document d'exemple d'ePiX

Version 1.0

Septembre, 2004

1 Vue d'ensemble

ePiX est un utilitaire poids-plume puissant et flexible pour la création de figures et graphes LaTeXien mathématiquement exacts à l'aide de commandes simples et faciles à retenir. Un manuel de l'utilisateur est disponible sous différent format à

`http://math.holycross.edu/~ahwang/current/ePiX.html`

Ce document d'exemple montre quelques unes des capacités d'ePiX en plaçant côte à côte le fichier d'entrée [1] et la sortie correspondante pour comparaison

- Les scènes sont décrites à l'aide des coordonnées cartésiennes mathématiquement canoniques, ce qui fait d'ePiX un véritable format vectoriel. Des figures bien réalisées ont une qualité photographique quels que soient leurs tailles et leurs formats. Il n'y a presque aucun choix par défaut : le format, la couleur, le point de vue (pour les images 3-D) et l'épaisseur des traits sont complétement controlables ;
- On peut facilement y incorporer une typographie de haute qualité ;
- On dispose de toute la puissance du C++ pour utiliser et étendre les capacités du programme ;
- La licence, la *GNU GPL*, est semblable aux conditions qui s'appliquent aux théorèmes : vous pouvez utilisez ePiX pour tout but, examiner le code, étudier le fonctionnement du programme, l'améliorer et publier vos améliorations tant que vous ne restreignez pas les droits des autres à faire pareil.

2 Exemples

ePiX fournit les capacités traditionnelles de représentation graphique comme les graphes en coordonnées cartésiennes ou polaire, la représentation de données issues d'un fichier, les courbes et surfaces paramétrées (mais sans effacement automatique des objets cachés). Comme signalé, l'implantation permet de considérer ePiX comme un langage de programmation, tout algorithme numérique écrit en C ou C++ peut être utilisé dans une figure ePiX. L'utilisation de concepts de programmation peut rendre la figure plus flexible – de sorte que l'apparence peut être changer précisément mais radicalement en ne faisant que quelques petites modifications du fichier d'entrée – et plus simple à maintenir.

Les figures ci-dessous ont été choisies pour insister sur des résultats qui sont relativement difficile à obtenir avec d'autres logiciels Libres (et quelques uns commerciaux) existants de représentation graphique. Les figures simples telles que les polygones, les ellipses et les splines sont presque autodescriptives, on n'en a donc donné aucun exemples.

Dans les fichiers ci-dessous, on place souvent plusieurs instructions courtes sur une même ligne pour faire tenir le fichier dans la page ; ce n'est pas une bonne habitude de programmation et on devrait l'éviter dans les vrais fichiers. Plusieurs fichiers d'exemple pourraient être raccourcis en fixant les constantes en dur.

1. Les fichiers d'exemple distribués avec ePiX ont des lignes d'`offset` pour que la figure se trouve près du code source.

```
#include "epix.h"
using namespace ePiX;

// la fonction à représenter
double f(double x) { return x*x; }

int main()
{
  bounding_box(P(-2,0),P(2,4));
  unitlength("1in");
  picture(3,3);

  begin();

  grid(32,32); // quadrillage fin

  bold();
  grid(4,4); // et grossier

  h_axis_labels(4, P(0,-2), b);
  v_axis_masklabels(P(0, 1), P(0, y_max), 3, P(0,0), c); // omet 0

  red();
  plot(f,x_min,x_max,20);

  end();
}
```

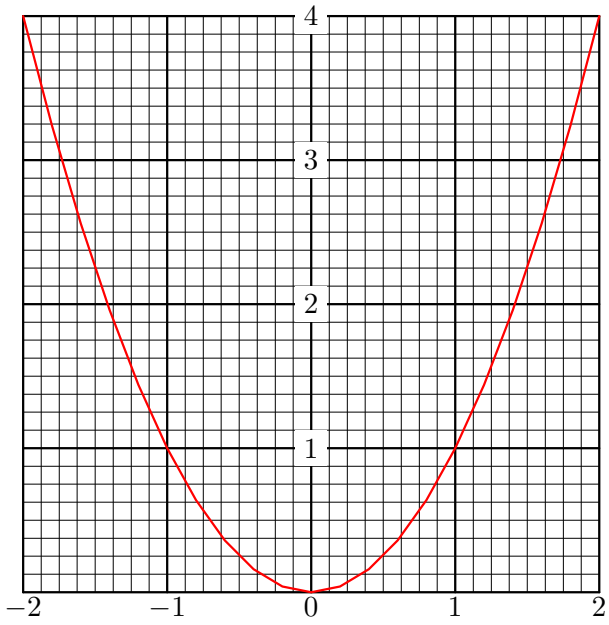


FIGURE 1 – Un graphe élémentaire.

```
#include "epix.h"
using namespace ePiX;

double f(double t) { return 2*t*(1-t)*(1-t); }
double g(double t) { return 1/(1-t*t); }

int main()
{
  bounding_box(P(-2,-4), P(2,4));
  picture(200,200);
  unitlength("1pt");

  begin();

  crop();
  // asymptotes verticales
  dashed();
  line(P(-1, y_min), P(-1, y_max));
  line(P( 1, y_min), P( 1, y_max));
  solid();

  // Axes
  h_axis(8);
  v_axis(8);

  h_axis_labels(4, P(-1, 2), tl); // aligne en haut à gauche
  v_axis_labels(4, P(-1, 2), tl);

  // Graphes
  plot(f, x_min, x_max, 40);
  bold();
  plot(g, x_min, x_max, 80);

  end();
}
```

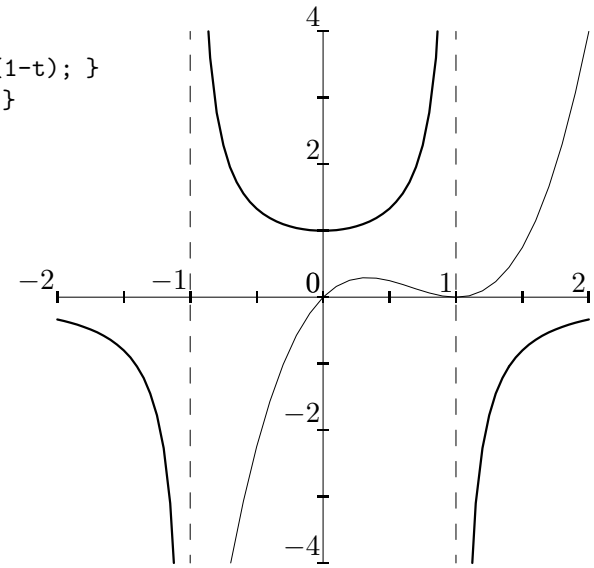


FIGURE 2 – Le « rognage » retire les éléments situés hors de la boite-cadre.

Il existe des « modes » angulaires pour les coordonnées polaires et sphériques. `ePiX` définit ses propres fonctions trigonométriques qui sont sensibles au mode courant.

```
#include "epix.h"
using namespace ePiX;

double f(double t)
{
  return 2*Cos(3*t);
}

int main()
{
  unitlength("1pt");
  bounding_box(P(-2,-2), P(2,2));
  picture(180, 180);

  begin();
  degrees();

  h_axis(P(x_min, y_min), P(x_max, y_min), 8);
  v_axis(P(x_min, y_min), P(x_min, y_max), 8);

  polar_grid(2, 4, 24); // rayon, anneaux, secteurs

  h_axis_labels(P(x_min,y_min), P(x_max,y_min), x_size, P(-12,-14));
  v_axis_labels(P(x_min,y_min), P(x_min,y_max), y_size, P(-4,0), 1);

  bold();
  polarplot(f, 0, 180, 120); // graphe sur [0, 180] avec 120 intervalles

  end();
}
```

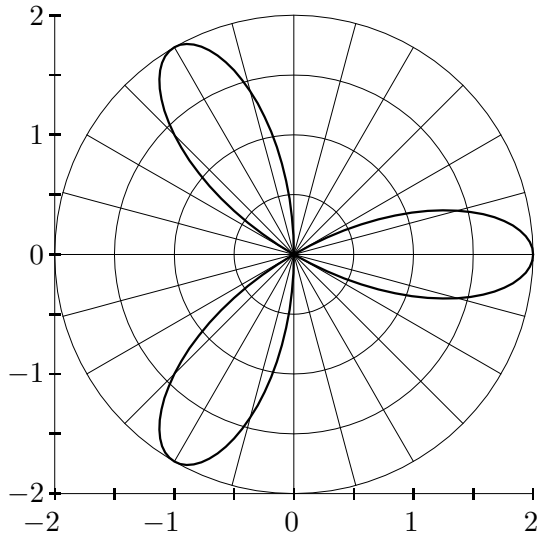


FIGURE 3 – Le graphe en polaire $r = 2\cos 3\theta$ pour $0 \leqslant \theta \leqslant \pi$.

Les régions entre les courbes peuvent être ombrées. La densité de gris varie entre 0 (blanc) et 1 (noir).

```
#include "epix.h"
using namespace ePiX;
double k=4;           // change la largeur de la bosse
double dx = 0.05;     // épaisseur de région ombrée fine
double x = 1/sqrt(k); // position de région ombrée fine
double dy = 0.25;     // pour distance verticale entre étiquettes

double f(double t) { return sqrt(fabs(k)/(2*M_PI))*exp(-k*t*t); }
P pt1 = P(x, f(x)+2*dy), pt2 = P(x+dx,f(x)+dy), pt3 = P(x+3*dx,f(x));

int main()
{
  unitlength("1pt");
  picture(150, 150);
  bounding_box(P(0,0), P(1,1));

  begin();
  fill();
  gray(0.1);     shadeplot(f, x_min, x, 90);
  gray(0.4);     rect(P(x,f(x)), P(x+dx, 0));
  gray(0.6);     shadeplot(f, x, x+dx, 10);
  fill(false);

  arrow_camber(0.25);  arrow_fill(1);
  arrow(pt1, P(0.5*(x_min+x), 0.5*f(0.5*(x_min+x))),
        P(0,0), "$F(x)=\\displaystyle\\int_a^x f(t)\\,dt$", tr);

  arrow(pt2, P(x+0.5*dx, f(x)),
        P(0,0), "Aire du rectangle = $f(x)\\,dx$", tr);

  arrow(pt3, P(x+dx, 0.5*f(x+dx)),
        P(0,0), "Aire = $F(x+dx)-F(x)$", tr);

  bold();
  plot(f, x_min, x_max, 120);

  plain(); h_axis(4); v_axis(4);

  label(P(x_min,0), P(0,-5), "$a$", b);
  label(P(x,0), P(0,-5), "$x$", b);
  label(P(x+dx,0), P(0,-2), "$x+dx$", br);

  end();

}
```

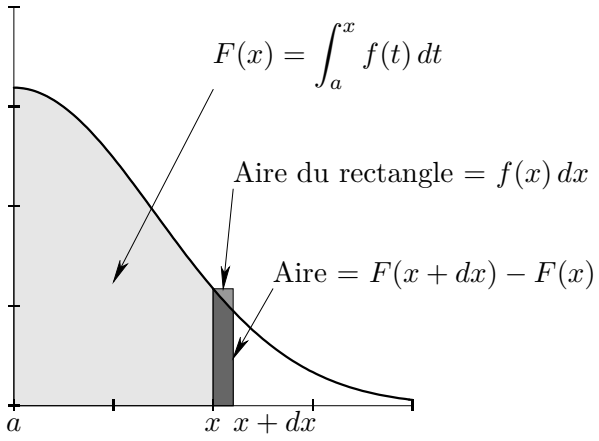
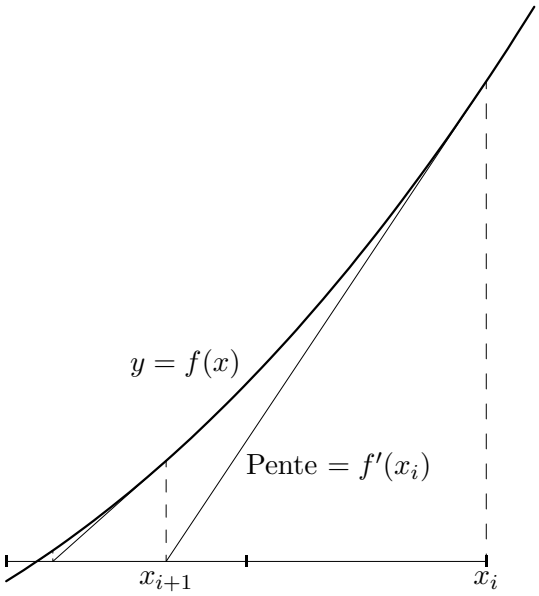


FIGURE 4 – Le théorème fondamental de l'analyse : $F'(x) = f(x) + o(1)$.

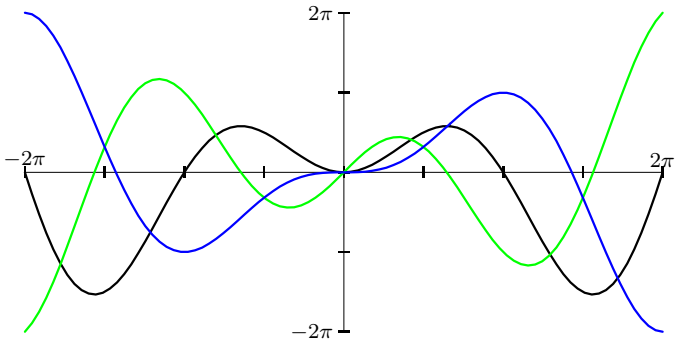
Les structures de controle – boucles et instructions conditionnelles – peuvent être utilisées pour créer des fichiers d'entrée dont la structure logique s'accorde à la structure mathématique de la figure.




```
#include "epix.h"
using namespace ePiX;

double f(double t) { return t*t*t-3*t+1; }
double df(double t) { return 3*t*t - 3; }

int main()
{
  bounding_box(P(1.5,0),P(2,3));
  unitlength("1in");
  picture(2.5,2.5);

  begin();
  h_axis(2);

  double x0 = 2;
  label(P(x0,0), P(0,-4), "$x_i$", b);

  for (int i=0; i < 3; ++i)
    {
      dashed();     line(P(x0,0), P(x0,f(x0)));
      solid();      line(P(x0,f(x0)), P(x0-f(x0)/df(x0),0));

      x0 -= f(x0)/df(x0); // méthode de Newton
    }

  bold();    plot(f, x_min, x_max+0.05, 60);

  double x1 = 2-f(2)/df(2);

  label(P(1.75, f(1.75)), P(-2,2), "$y=f(x)$", tl);
  label(P(x1,0), P(0,-4), "$x_{i+1}$", b);
  label(P(1.75, df(2)*(1.75-x1)), P(0,-4), "Pente $= f'(x_i)$", br);

  end();
}
```

FIGURE 5 – Méthode de Newton d'approximation de la racine.

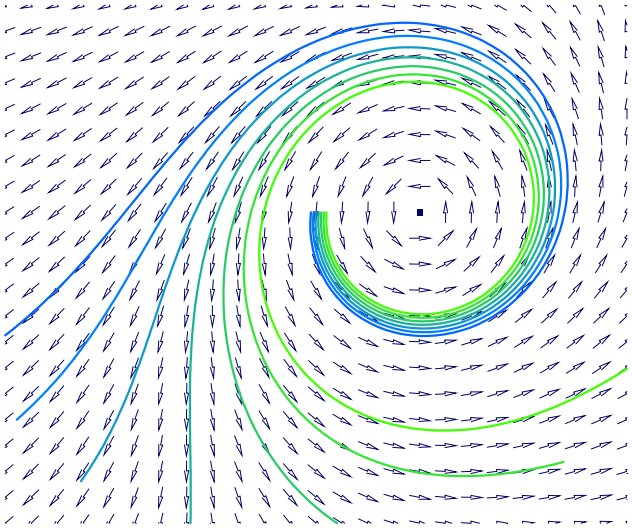
ePiX peut tracer les dérivées et les primitives :




```
#include "epix.h"
using namespace ePiX;

double MAX(2*M_PI);
double f(double t)
{
  return t*Sin(t);
}

int main()
{
  unitlength("1pt");
  picture(240, 120);
  bounding_box(P(-MAX,-MAX), P(MAX,MAX));

  begin();

  // Coordinate axes and labels
  h_axis(8);
  v_axis(4);
  font_size("scriptsize");

  label(P(0,y_max), P(-4,0), "$2\\pi$", l);
  label(P(0,y_min), P(-4,0), "$-2\\pi$", l);

  label(P(x_min,0), P(0,2), "$-2\\pi$", t);
  label(P(x_max,0), P(0,2), "$2\\pi$", t);

  bold();
  plot(f, x_min, x_max, 90);
  green();
  plot(Deriv(f), x_min, x_max, 90);

  blue();
  plot(Integral(f, 0), x_min, x_max, 90); // primitive nulle en 0

  end();
}
```

FIGURE 6 – $y = x \sin x$ (noir), sa dérivée (vert) et la primitive nulle en 0 (bleu).

Les solutions des équations différentielles ordinaires sont calculées par la méthode d'Euler. Les champs de vecteurs tangents peuvent être représentés avec une norme constante (figure) ou avec la vraie norme.



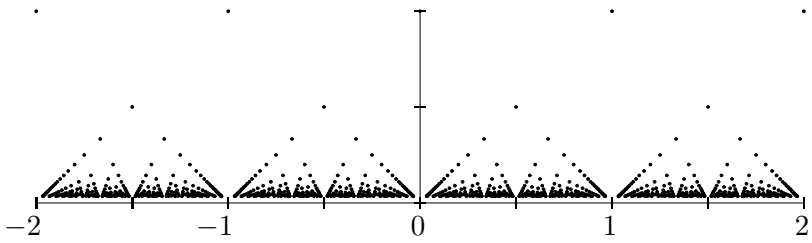
```
#include "epix.h"
using namespace ePiX;

P F(double s, double t)
{
  return (P(0.1,0.025)&P(s,t)) +
    (1/(0.01+s*s+t*t))*P(-t,s);
}

int main()
{
  unitlength("1pt");
  bounding_box(P(-4, -3), P(2,2));
  picture(234, 195);

  begin();
  crop();

  blue(0.4);
  dart_field(F, P(x_min, y_min), P(x_max, y_max), 4*x_size, 4*y_size);

  bold();
  for (int i=0; i<7; ++i)
    {
      rgb(0.25 - 0.05*i, 1 - 0.1*i, 0.2*i);
      ode_plot(F, P(-0.9-0.025*i,0), 20, 200);
    }
  end();
}
```

FIGURE 7 – Un champ de tangentes et six solutions d'une EDO.

La fonction f « dénominateur » définie par

$$f(x) = \begin{cases} \frac{1}{q} & \text{si } x = \frac{p}{q} \text{ est réduite} \\ 0 & \text{si } x \text{ est irrationnel} \end{cases}$$

peut être représentée avec des boucles `pour` imbriquées.




```
#include "epix.h"
using namespace ePiX;

int N = 30; // denominateur maximum représenté

int main()
{
  unitlength("0.01in");
  bounding_box(P(-2,0), P(2,1));
  picture(400,100);

  begin();

  h_axis(2*x_size);
  v_axis(2);

  h_axis_labels(x_size, P(-12, -12));

  dot_size(1); // trace des points de 1pt de diamètre
  for (int i=1; i< N; ++i)
    for (int j=i*x_min; j <= i*x_max; ++j)
      if (gcd(i, j) == 1)
        dot(P(j*1.0/i, 1.0/i));

  end();
}
```

FIGURE 8 – Une fonction « pathologique » en analyse réelle.

Les sommes finies sont définies par un algorithme et donc faciles à représenter. La fonction que l'on réduit et somme est la fonction cb « Charlie Brown » (en bleu). Le fait qu'elle soit dérivable nulle part est patent d'après la figure puisque le graphe est autosemblable (rouge) et n'est pas une droite.

```
#include "epix.h"
using namespace ePiX;

const int N=8; // Nombre de termes

double weierstrass(double t)
{
  double y=0;
  for(int i=0; i < N; ++i)
    y += pow(2,-i)*cb(pow(2,i)*t);

  return y;
}

int main()
{
  bounding_box(P(-2, 0), P(2, 1.5));
  picture(320, 120);
  unitlength("0.01in");

  begin();

  h_axis(2*x_size);
  v_axis(2*y_size);
  h_axis_labels(x_size, P(-12,-14));

  blue();
  plot(cb, x_min - 0.25, x_max+0.25, 4*x_size + 2);

  bold();
  black();
  plot(weierstrass, x_min, x_max, pow(2,N));

  red();
  plot(weierstrass, 0.5, 1.5, pow(2,N-2));

  end();
}
```

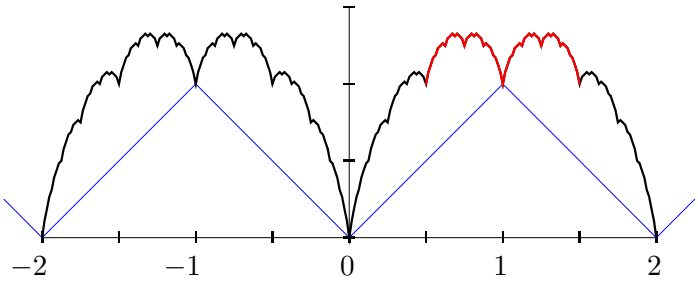


FIGURE 9 – Une fonction dérivable nulle part de Weierstrass.

`ePiX` peut approcher les extrémums d'une fonction sur un intervalle, ce qui est utile pour tracer des rectangles inscrits ou circonscrits dans un graphe. La fonction sinus est symbolisée par `f` dans tout le corps (sauf dans les étiquettes bien entendu) aussi pour redessiner la figure avec un autre intégrande il suffit de changer la définition de `f` et de retailler la `bounding_box`. Le nombre de rectangles est, de même, « paramétré ».

```
#include "epix.h"
using namespace ePiX;

const int N=12; // Nombre de rectangles
double f(double t) { return Sin(t); } // contient les références à l'intégrande

int main()
{
  bounding_box(P(0,0), P(3,1));
  unitlength("1in");
  picture(3, 1);

  begin();

  double dx = x_size/N;
  gray(0.1); // (dés)active le remplissage dans la boucle

  riemann_sum(f, x_min, x_max, N, UPPER);
  fill();
  riemann_sum(f, x_min, x_max, N, LOWER);
  fill(false);

  h_axis(x_size);
  v_axis(2*y_size);

  h_axis_labels(x_size, P(0,-4), b);
  label(P(x_max, f(x_max)), P(4,0), "$y=\\sin x$", r);

  bold();    blue();
  plot(f, x_min, x_max, 40);

  end();
}
```

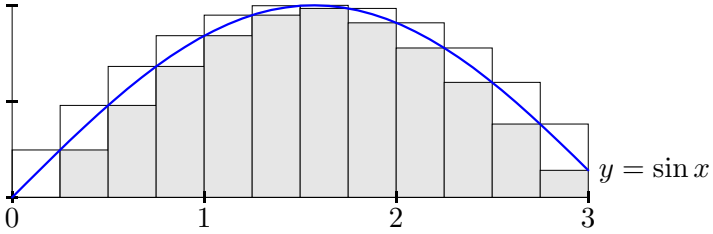


FIGURE 10 – Sommes supérieure et inférieure de $\int_0^3 \sin x \, dx$.

On peut utiliser une boucle pour controller toute une figure et créer ainsi une suite d'instantanés d'une figure variant avec le temps – comme une roue tournant – ou une courbe intégrale d'une EDO. La page du projet `ePiX` contient des liens vers des animations qui peuvent être « jouées » dans n'importe quel navigateur.

```
#include "epix.h"
using namespace ePiX;

P F(double t, double r)
{
  return P(t-r*Sin(t), 1-r*Cos(t));
}

int main()
{
  const double dt = 5*M_PI/11;
  double t = 0;

  picture(8, 1);
  bounding_box(P(-1,0), P(15,2));
  unitlength("0.625in");

  for(int i=0; i < 9; ++i)
    {
      t += dt;
      begin(); // L'image entière dans le corps de la boucle

      line(P(x_min, y_min), P(x_max, y_min)); // le sol

      circle wheel(P(t,1), 1);  wheel.draw();

      domain R = domain(P(0,0), P(t, 1), // [0,t] x [0,1]
                        mesh(10*i,5), mesh((int) ceil(1+4*t), 5));

      // le chemin
      bold();
      for (int j=0; j < 6; ++j)
        {
          rgb(1-0.125*j, 0.125*j, 0.5+0.25*j);
          plot(F, R.slice2(0.2*j));
        }

      // le rayon
      green();
      line(P(t,1), F(t,1));

      end();

      // séparateur de la figure et espace vertical
      if (i < 8)
        printf("\n\\vspace*{3ex}\n%%%%");
    }
}
```

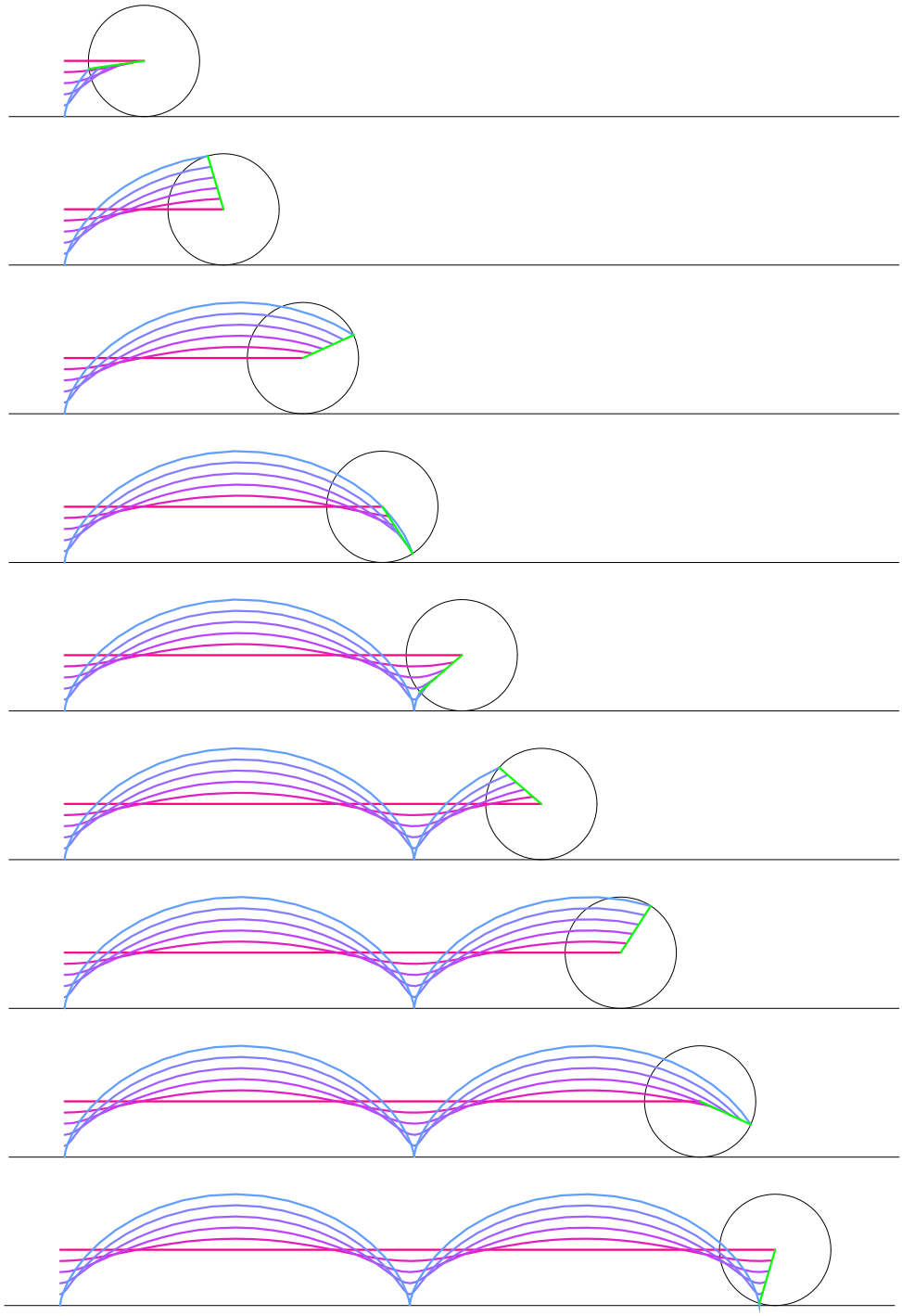


FIGURE 11 – Instantanés de cycloïdes.

La figure 12 – de Jacques L'helgoual – démontre quelques unes des capacités en géométrie sphérique : les courbes gauches peuvent être projetées radialement sur la sphère unité et les courbes planes projetées stéréographiquement depuis le pole nord ou sud avec ou sans effacement des traits cachés.

```
#include "epix.h"
using namespace ePiX;

const double k = 2*M_PI/(360*sqrt(3)); // suppose le mode « degrees »
double exp_cos(double t) { return exp(k*t)*Cos(t); }
double exp_sin(double t) { return exp(k*t)*Sin(t); }
double minus_exp_cos(double t) { return -exp_cos(t); }
double minus_exp_sin(double t) { return -exp_sin(t); }

int main()
{
  bounding_box(P(-1,-1), P(1,1));
  picture(160,160);
  unitlength("1pt");

  begin();
  degrees(); // fixe l'unité d'angle
  viewpoint(1, 2.5, 3);

  sphere S1=sphere(); // sphère unité
  S1.draw();

  pen(0.15); // parties cachées des loxodromies
  blue();
  backplot_N(exp_cos, exp_sin, -540, 540, 90);
  backplot_N(minus_exp_cos, minus_exp_sin, -540, 540, 90);
  red();
  backplot_N(exp_sin, minus_exp_cos, -540, 540, 90);
  backplot_N(minus_exp_sin, exp_cos, -540, 540, 90);

  black(); // coordinate grid
  for (int i=0; i<=12; ++i) { latitude(90-15*i,0,360); longitude(30*i,0,360); }

  bold(); // parties visibles des loxodromies
  blue();
  frontplot_N(exp_cos, exp_sin, -540, 540, 360);
  frontplot_N(minus_exp_cos, minus_exp_sin, -540, 540, 360);
  red();
  frontplot_N(exp_sin, minus_exp_cos, -540, 540, 360);
  frontplot_N(minus_exp_sin, exp_cos, -540, 540, 360);

  end();
}
```

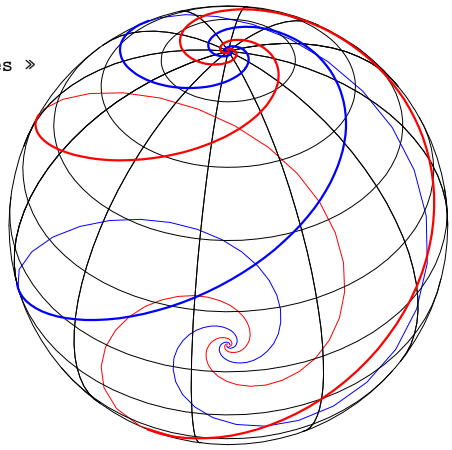
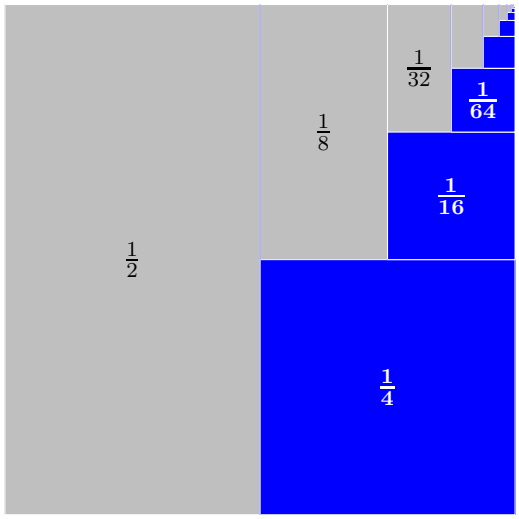


FIGURE 12 – Loxodromies sur la sphère unité.

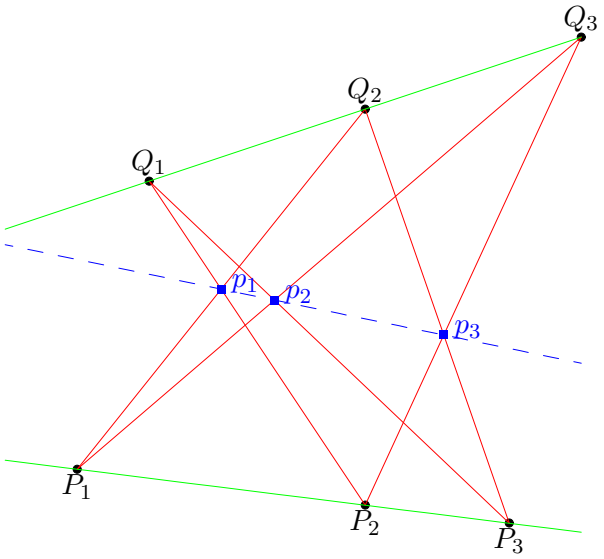
On peut inclure du code `PSTricks` dans un fichier `ePiX` pour obtenir un remplissage en couleur et d'autres effets. La commande `psset` place ses arguments dans le fichier de sortie.




```
#include "epix.h"
using namespace ePiX;

const int N=8;

int main()
{
  unitlength("1.5pt");
  bounding_box(P(0,0), P(1,1));
  picture(128,128);

  use_pstricks();
  begin();

  psset("fillcolor=lightgray, linecolor=white");
  rect(P(0,0), P(1,1));
  label(P(1.0/4,   1.0/2), "$\\frac{1}{2}$");
  label(P(5.0/8,   3.0/4), "$\\frac{1}{8}$");
  label(P(13.0/16, 7.0/8), "$\\frac{1}{32}$");

  psset("fillcolor=blue");
  double t=0.5;

  for(int i=0; i<N; ++i, t *= 0.5)
    {
      rect(P(1-t, 1-2*t), P(1, 1-t));
      line(P(1-t, 1-2*t), P(1-t, 1));
    }
  white();
  label(P(3.0/4,     1.0/4), "$\\mathbf{\\frac{1}{4}}$");
  label(P(7.0/8,     5.0/8), "$\\mathbf{\\frac{1}{16}}$");
  label(P(15.0/16, 13.0/16), "$\\mathbf{\\frac{1}{64}}$");

  end();
}
```

FIGURE 13 – Une suite géométrique de rapport 1/2.

ePiX calcule les intersections d'objets géométriques tels que les droites, les cercles et les plans.




```
#include "epix.h"
using namespace ePiX;

int main()
{
  bounding_box(P(-2,-2),P(2,2));
  unitlength("1in");
  picture(3,3);

  begin();
  crop();


  P V = P(2,-0.25),   W = P(3,1);
  P P1=P(-1.5,-1),    P2 = P1 + V,       P3 = P1 + 1.5*V;    // points
  P Q1 = P(-1,1),     Q2 = Q1 + 0.5*W,   Q3 = Q1 + W;

  segment L12(P1, Q2),   L13(P1, Q3);
  segment L21(P2, Q1),   L23(P2, Q3);
  segment L31(P3, Q1),   L32(P3, Q2);

  P R1 = L12*L21,    R2 = L13*L31,    R3 = L32*L23;   // points d'intersection

  dot(P1, P(0,-2), "$P_1$", b);   dot(Q1, P(0,2), "$Q_1$", t);
  dot(P2, P(0,-2), "$P_2$", b);   dot(Q2, P(0,2), "$Q_2$", t);
  dot(P3, P(0,-2), "$P_3$", b);   dot(Q3, P(0,2), "$Q_3$", t);

  red();  L12.draw();  L21.draw();  L13.draw();  L31.draw();  L32.draw();  L23.draw();

  green();    Line(P1,P3);    Line(Q1,Q3);

  blue();
  box(R1, P(4,2), "$p_1$", r);
  box(R2, P(4,2), "$p_2$", r);
  box(R3, P(4,2), "$p_3$", r);

  dashed();    Line(R1,R3);
  end();
}
```

FIGURE 14 – Théorème de Pascal sur les côtés de l'hexagone.

On peut construire des chemins par morceaux et les manipuler comme un objet unique.

```
#include "epix.h"
using namespace ePiX;

int main()
{
  bounding_box(P(-5, -5), P(5,5));
  picture(160, 160);
  unitlength("0.35mm");

  use_pstricks();
  begin();  degrees();
  double theta=60;
  double rad=0.25, Rad=4.5; // rayons des arcs
  double ht=1.0/16;         // demi-épaisseur d'une coupure

  // « débuts » des arcs de cercle
  double theta1 = Asin(ht/rad), theta2 = Asin(ht/Rad);

  // build keyhole in pieces; += concatenates, -= reverses orientation
  path contour(P(0,0), Rad*E_1, Rad*E_2, theta2, 360-theta2);    // arc extérieur
  contour -= path(polar(rad,-theta1), polar(Rad,-theta2));       // bord inférieur
  contour -= path(P(0,0), rad*E_1, rad*E_2, theta1, 360-theta1); // arc intérieur
  contour += path(polar(rad, theta1), polar(Rad, theta2));       // bord supérieur

  fill();
  std::cout << "\n\\newrgbcolor{orange}{1 0.7 0.2}";

  psset("fillcolor=orange,linecolor=red,linewidth=1.5pt");
  contour.set_fill();
  contour.draw();
  use_pstricks(false);

  dot(P(0,0));   // the origin
  gray(1); // blacken arrowheads
  arrow(P(Rad/4, 0.1*Rad), P(3*Rad/4, 0.1*Rad)); // pointes de flèche (en radians)
  arrow(P(3*Rad/4, -0.1*Rad), P(Rad/4, -0.1*Rad));
  arc_arrow(P(0,0), 0.9*Rad, 180-theta, 180+theta);

  label(P(Rad,ht), P(2,4), "$R\\to\\infty$", tr);
  label(P(0,rad), P(0,4), "$\\delta\\to0$", tl);
  label(polar(Rad, 45), P(0,0), "$\\gamma$", tr);

  end();
}
```

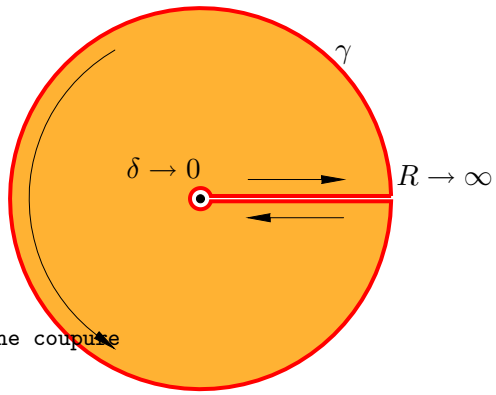
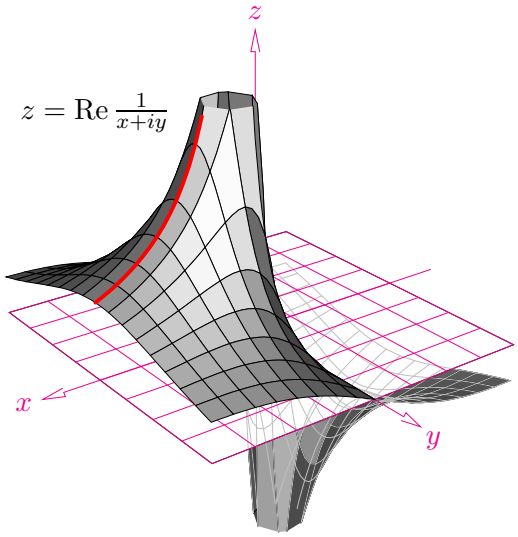


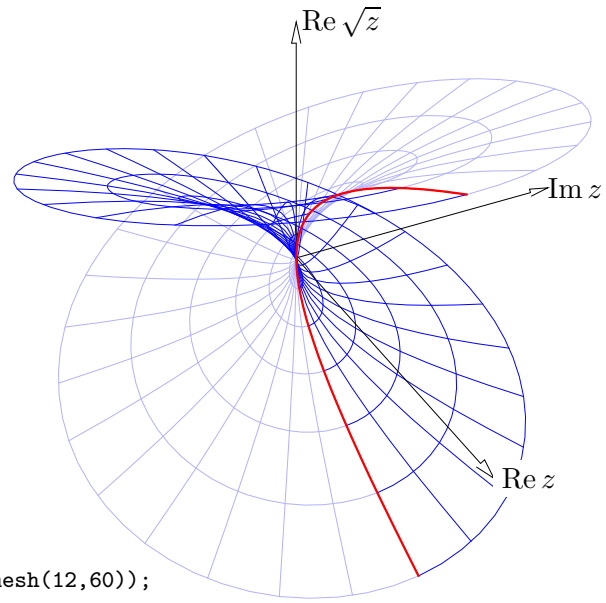
FIGURE 15 – Un contour en forme de serrure pour la coupure d'une branche intégrale.

ePiX fournit le « rognage », analogue tridimensionnel du cadrage ; les éléments de la figure hors de la boite de coordonnées sont effacés.




```
#include "epix.h"
using namespace ePiX;
const int M=8;
double MAX=1.5, Bd=2;
P F(double x, double y) // faux pole
{ return P(x, y, x/(0.01+x*x+y*y)); }
P pt1(-MAX,-MAX), pt2(MAX, MAX);

int main()
{
  bounding_box(P(-Bd,-Bd), P(Bd,Bd));
  unitlength("1in");
  picture(2.5,2.5);

  begin();
  domain R(pt1, pt2, mesh(2*M,2*M), mesh(6*M,6*M));
  label(P(x_min, y_max), P(2,-2), "$z=\\mathrm{Re}\\,\\frac{1}{x+iy}$", br);
  viewpoint(6,8,5);

  clip(); clip_box(P(-Bd,-Bd,-Bd), P(Bd,Bd,0)); fill(); surface(F, R);
  // moitié inférieure
  gray(0); rect(pt1, pt2);  fill(false);
  pen(0.15); rgb(0.75,0.75,0.75); plot(F, R); plain();
  // simule la transparence

  clip(false); magenta(); grid(pt1, pt2, M, M); // axes
  arrow(P(-Bd,0,0), P(Bd,0,0));  label(P(Bd,0), P(-4,-2), "$x$", l);
  arrow(P(0,-Bd,0), P(0,Bd,0));  label(P(0,Bd), P( 2,-2), "$y$", br);
  arrow(P(0, 0, 0), P(0,0,2.5)); label(P(0,0,2.5), P(0,4), "$z$", t);

  clip(); clip_box(P(-Bd,-Bd,0), P(Bd,Bd,Bd)); fill(); surface(F, R);
  // moitié supérieure
  fill(false); pen(1.5); red(); plot(F, R.resize1(0.5,MAX).slice2(0));
  end();
}
```

FIGURE 16 – La partie réelle de la fonction réciproque complexe.




```
#include "epix.h"
using namespace ePiX;

P f(double x, double y)
{
  pair z = pair(x,y);
  pair temp = z*z;
  return P(temp.x1(), temp.x2(), x);
}

inline P F(double r, double t)
{ return f(r*Cos(t), r*Sin(t)); }

P g(double t) { return t*P(t,0,1); }
const double MAX=1.5;

domain R(P(0,0), P(1.25,0.5), mesh(6,24), mesh(12,60));

int main()
{
  bounding_box(P(-MAX,-MAX),P(MAX,MAX));
  unitlength("1in");
  picture(2.5,2.5);

  begin();
  revolutions();    viewpoint(4,-2,3);

  pen(0.15);  rgb(0.7,0.7,1);
  plot(F, R);

  plain();  blue();
  plot(F, R.resize2(0.5,1));

  black();  pen(0.5);
  arrow(P(0,0,0), 2*E_1);
  arrow(P(0,0,0), 2*E_2);
  arrow(P(0,0,0), 1.5*E_3);

  masklabel(2*E_1, P(0,0), "$\\mathrm{Re}\\,z$", r);
  label(2*E_2, P(0,0), "$\\mathrm{Im}\\,z$", r);
  label(1.5*E_3, P(2,0), "$\\mathrm{Re}\\,\\sqrt{z}$", r);

  bold(); red();
  plot(g, -1.25, 1.25, 40);
  end();
}
```

FIGURE 17 – Deux feuillets de la surface de Riemann de $\sqrt{z}$.

Gallerie

Les fichiers d'entrée des figures ci-dessous sont dans le répertoire `samples` de l'arborescence des sources.