Rencontres N°3
octobre 2004

Mathematica®

Éditorial

Jacqueline Zizi
jazi@club-internet.fr

Une rencontre attendue que celle du 6 octobre! La précédente, appelée "*Mathematica* Days" en France avait eu lieu en 1993, il y a donc 11 ans! Pour mémoire suivent les contenus des 2 premières brochures.

■ Rencontres n°1, 2000 - 2001

■ Rencontres n°2, 2002 - 2003

Remerciements

Cette brochure papier
réalisée
avec le traitement de textes *Mathematica*
pour la conférence parisienne
du 6 octobre 2004
ne serait pas là

sans l'enthousiasme, la compétence
et la bonne intelligence
de ses auteurs
qui modélisent, programment, explorent découvrent, et enseignent
avec *Mathematica*

sans la prise en charge
par Wolfram Research
des soucis et des frais
de production

Merci à tous!

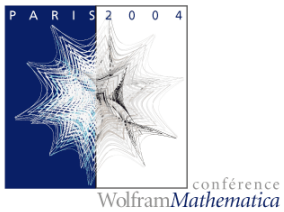
Les articles sont consultables,
et téléchargeables
sur le site
`http://wolfram.com`

Éditeur
Jacqueline Zizi
jazi@club-internet.fr
http://homepage.mac.com/jacquelinezizi

Rencontres autour de Mathematica

■ Et déjà...

Le 6 octobre, la Conférence de Paris au palais des Congrès, à qui nous devons la sortie de cette brochure "Rencontres" n°3.



■ Formations en France

Un groupe de plusieurs personnes assure, en France, des formations certifiées par Wolfram Research:

`https://www.wolfram.com/services/education/`

■ L'offre spéciale pour les lycées et prépas a été prolongée et améliorée

Un effort particulier sur les prix, met maintenant *Mathematica* à la portée de toutes les bourses dans ces classes. `http://www.wolfram.com`

Des accords avec les universités ont aussi été conclus dernièrement.

■ Wolfram Research à Apple Expo

Pour la première fois, Wolfram Research était présent à Apple Expo 2004 qui a eu lieu à Paris du 31 août au 4 septembre inclus.

■ Autriche: AISC 2004 au RISC (Research Institute for Symbolic Computation), Castle of Hagenberg, Autriche

Quand les spécialistes de l'IA (Intelligence Artificielle) rencontrent ceux du calcul symbolique, cela mène à la démonstration automatique, au design des programmes pour travailler les mathématiques abstraites, à la formalisation de nouvelles théories.... au futur des mathématiques....
`http://www.risc.uni-linz.ac.at/conferences/aisc2004/`

■ Wolfram Technology Conference 21-23 octobre à Champaign

Cette conférence s'inscrit dans la suite des "developer conference", qui avaient lieu environ tous les deux ans. Pour la première fois *plusieurs* français iront.. Il y a eu par le passé, une puis deux personnes, puis trois...

■ IMS 06 à Avignon? On y travaille....

Si vous voulez participer à son organisation, vous serez les bienvenus. S'adresser à Bruno Autin: `bautin1@club-internet.fr`

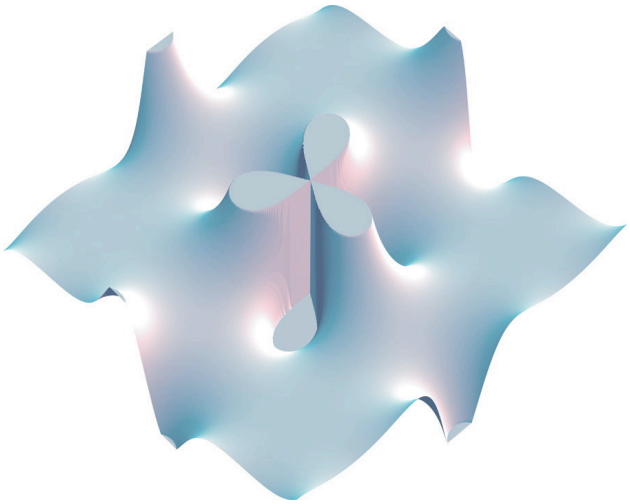
■ Et chez nos voisins hollandais

`http : // www.can.nl / english / index.htm`

■ *Mathematica* dans les écoles françaises d'architecture demain?

EZCT Architecture & Design Research organisera au printemps 2005 un workshop accompagné d'une rencontre autour deMathematica. Cet événement se tiendra au sein de la Villa Van Doesburg (maison construite entre 1927 et 1930) par l'architecte, sculpteur et artiste hollandais Théo van Doesburg, à Meudon.

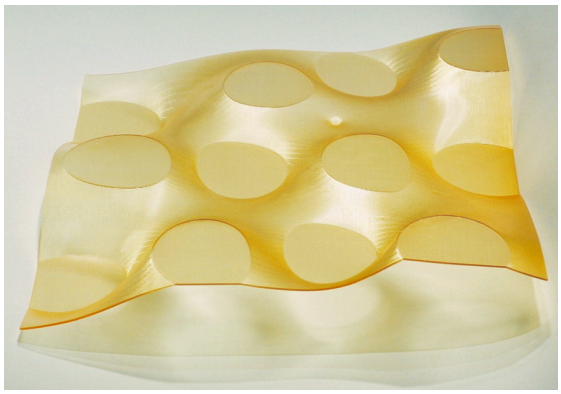
Les détails du workshop seront annoncés à l'adresse suivante : `www.ezct.net`
Pour tout renseignement s'adresser à :
`philippe.morel@ezct.net`



Plot3D Mathematica d'une surface ayant des surfaces d'arasement identiques
EZCT Architecture & Design Research avec Maryvonne Teissier, Paris 7

Ce workshop de printemps aura pour but d'étendre l'usage de Mathematica à l'enseignement de l'architecture au sein des différentes écoles et universités, cet outil répondant parfaitement aux nouvelles données liées à l'explosion de l'informatique et à l'exigence d'une maîtrise conceptuelle (bien sûr également pratique) des logiciels liés aux méthodes computationnelles de création architecturale.

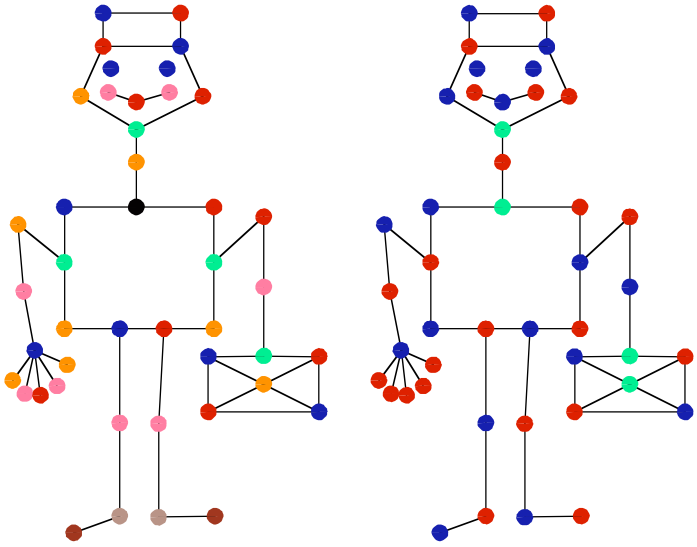
Participeront à cet événement des enseignants et étudiants en architecture, accompagnés d'enseignants dans les domaines des mathématiques et de l'informatique.



Modèle réalisé en stéréolithographie à partir d'un fichier STL Mathematica

Collège Doucet à Nanterre
2ème à
"Faites de le Science"

Jacqueline Zizi
jazi@club-internet.fr



■ Le concours "Faites de le Science"

En 2004 a eu lieu le concours "faites de la science". Organisé sur 4 départements parisiens, par plusieurs universités dont Orsay, il regroupe au départ de nombreux établissements. A la fin d'une première étape les meilleurs projets sont soutenus. Puis le concours dans sa phase finale prime les réalisations d'excellence.

2004
http : // www.u - psud.fr / orsay / index.nsf / Page / faitesdelascience

2005
http : // www.u - psud.fr / Orsay / default.nsf / Page / ConcFdS2005

■ L'équipe gagnante du collège André Doucet 4è 1

Ci-dessus le noyau dur de l'équipe, les "trouveurs", les bâtisseurs du CD, les meneurs du jeu. Mais les autres élèves aussi ont tous participé. Ils s'appellent: Samhane Abdallah, Loubna Adraoui, Emeline Arconte, Fatma Belkalem, Farah Benmerabet, Sami Brirmi, Jessica Dell, Arnaud Dherbometz, Tony Divet, Vetrivel Duvizert, Adrien Etienne, Johan Kitous, Elodie Lamothe, Amélie Lamoureux, Armelle M'Bouche, Fath-Allah Mabrouki, Dounia Medaci, Idir Moghraoui, Amal Mohamed, Emmanuelle Moreau, Nadia Mounissamy, Disnaid N'Guyen Van Ho, Mathieu Sampaio, Charlène Seguin. Leurs enseignantes sont Madame Chanudeaud, maths, et Madame Elzière, anglais.

■ Le sujet

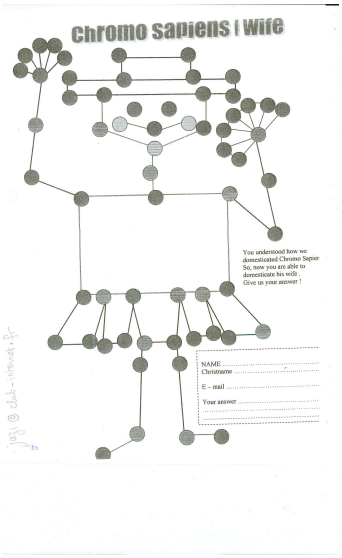
Le sujet doit être multidisciplinaire. A André Doucet, en 2003-2004, ils ont choisi "Anglais-Maths" et le "Chromo Sapiens". Il s'agit de trouver quelle formule magique permet de calculer le nombre de coloriages des sommets de figures simples, dont Chromo Sapiens, en fonction d'un nombre donné de couleurs.

La règle de coloration est que deux sommets rejoints ne peuvent être coloriés de la même façon. Trouver le nombre minimum de couleurs est aussi un objectif.

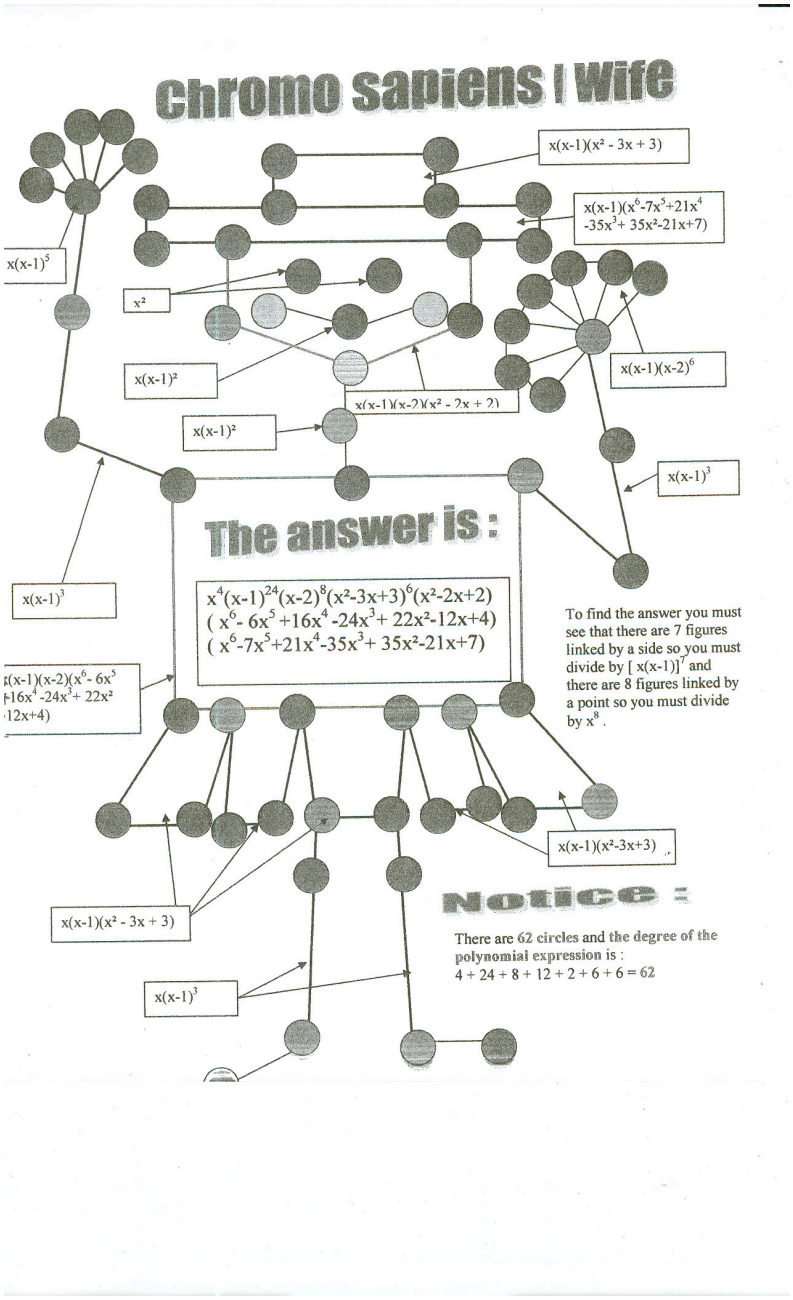
Le sujet a été créé de toutes pièces par Jacqueline Zizi, comme un produit dérivé de ses propres recherches sur les graphes et en particulier "Graph Explorer" développé en *Mathematica* principalement à Champaign lorsqu'elle y était Visiting Scholar chez Wolfram Research.. Donné il y a quelques années dans le cadre de l'association AMeJ (dites "Maths en Jeans"), il avait déjà passionné les élèves. Mais cette fois-ci, les résultats attendus ont été trouvés vite, et ils sont allés plus loin! Ils ont réalisés seul un CD graphique avec tous les exemples simples et toutes les démarches utiles. Très coloré, il a eu beaucoup de succès chez les visiteurs du stand où ils exposaient leur travaux, le jour du concours final. Mais ils sont allés encore plus loin...

■ L'avenir dans leurs questions

Chercher, pour des élèvs de 4ème sur un sujet non scolaire, c'est déjà bien, surtout s'ils commencent juste les polynômes. Mais un polynôme de degré 43... Trouver, c'est encore mieux. Prouver, cela devient fort. Mais se questionner, ouvrir d'autres problèmes, les résoudre, alors là, c'est vraiment remarquable! Et voici la question qu'ils ont posée à tous les présents le jour J...
"*You understood how we domesticated Chromo Sapiens, So, now you are able to domesticate his wife. Give us your answer!*"

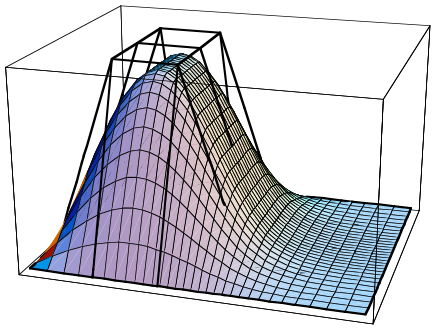


- **Et si vous ne trouvez pas la réponse, allez leur demander!**



Les B-Spline

Gérad Iglesias
gerard_iglesias@mac.com



Surface B-Spline avec son polyèdre descripteur

■ Introduction

Très utilisées dans le cadre des logiciels de création d'images, les splines constituent une primitive géométrique de base d'un nombre conséquent de logiciels de modélisation géométrique.

L'origine des splines remonte à l'utilisation, par les architectes et les dessinateurs industriels, de baguettes de bois ou de métal souple afin de tracer des courbes passant par un certain nombre de points caractéristiques.

Les courbes ainsi créées minimisent l'énergie de tension, et sont mathématiquement définies à l'aide de fonctions polynomiales par morceaux.

C'est ainsi que naturellement *Mathematica* offre des packages orientés calcul d'approximation et interpolation dont le résultat est une spline.

Dans le présent papier, je vais m'intéresser à une formulation plus visuelle et didactique qui vous permettra de mieux les appréhender et de jouer plus facilement avec leur représentation graphique 2D et 3D.

Les B-Splines permettent de définir des fonctions splines à l'aide d'une combinaison linéaire de fonctions de bases, la particularité étant que le lieu géométrique de la courbe ou surface ainsi obtenue est directement lié visuellement aux coefficients choisis, comme on peut le voir sur la figure .

Nous présenterons donc la défintion des fonctions de bases, ensuite de simples combinaisons linéaires nous permettrons de jouer avec ces intéressants objets...

■ Notation

Les notations sont inspirées de l'ouvrage de Les Piegl et Wayne Tiller 'The NURBS Book'.

Pour une couverture générale du domaine des courbes et surfaces vous pourrez consulter le livre de Gerald Farin 'Curves and Surfaces for CAGD'.

Pour faciliter les notations nous allons définir l'accès à l'élément d'une liste ou d'une matrice grâce à la notation par indice

$$\mathbf{U\_List}_{\mathbf{i\_Integer}} := \mathbf{U[[i+1]]}$$

$$\mathbf{P\_List}_{\mathbf{i\_Integer,j\_Integer}} := \mathbf{P[[i+1]][[j+1]]}$$

Le premier élément est par conséquent d'indice 0.

■ Fonctions de base

Nous définissons la i-eme base B-spline 'N' de degré 'p' définie sur une série de réels croissants 'U' de la manière suivante :

$$\mathbf{N_{i\_,0,U\_}[u\_?NumberQ] := 1 \;/;\; U_i \le u \wedge u < U_{i+1}}$$

$$\mathbf{N_{i\_,0,U\_}[u\_?NumberQ] := 0 \;/;\; u < U_i \vee U_{i+1} \le u}$$

$$\mathbf{N_{i\_,p\_,U\_}[u\_?NumberQ] :=} \frac{\mathbf{u - U_i}}{\mathbf{U_{i+p} - U_i}} \mathbf{N_{i,p-1,U}[u]} + \frac{\mathbf{U_{i+p+1} - u}}{\mathbf{U_{i+p+1} - U_{i+1}}} \mathbf{N_{i+1,p-1,U}[u]} \;/;\; \mathbf{U_{i+p} \neq U_i \wedge U_{i+p+1} \neq U_{i+1}}$$

$$\mathbf{N_{i\_,p\_,U\_}[u\_?NumberQ] :=} \frac{\mathbf{U_{i+p+1} - u}}{\mathbf{U_{i+p+1} - U_{i+1}}} \mathbf{N_{i+1,p-1,U}[u]} \;/;\; \mathbf{U_{i+p} == U_i \wedge U_{i+p+1} \neq U_{i+1}}$$

```
N_i_,p_,u_[u_?NumberQ] :=
  (u - U_i)/(U_(i+p) - U_i) N_(i,p-1,u)[u] /; U_(i+p) ≠ U_i ∧ U_(i+p+1) == U_(i+1)

N_i_,p_,u_[u_?NumberQ] := 0 /; U_(i+p) == U_i ∧ U_(i+p+1) == U_(i+1)
```

■ Exemple

Pour l'ensemble des fonctions splines de degré :

```
d = 3;
```

ayant pour support la série croissante de réels, que l'on appelle vecteur nœud' :

```
K = {0, 0, 0, 0, 1, 2, 3, 3, 3, 4};
```

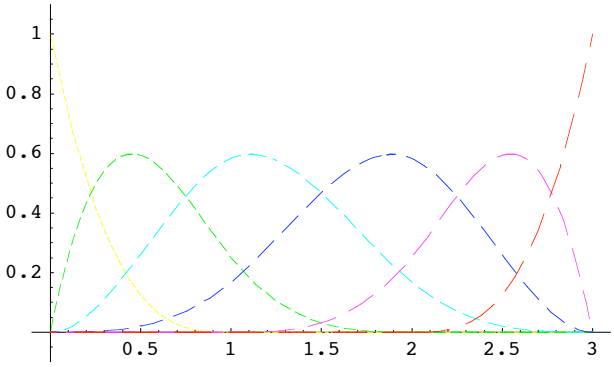
Le 'vecteur nœud' joue un rôle d'importance car il définit la longueur paramétrique des arcs polynomiaux de la courbe à définir.

Ici j'ai utilisé une astuce: {....,3,3,3,4}, qui permet d'éviter une discontinuité en $u = 3$, bien utile lors des évaluations numériques.

Les fonctions de base sont définies sur les supports successifs $U_i\ U_{i+d+1}$ avec i$\in$ {0,5}

Le tracé ne s'intéresse qu'à l'intervalle [0,3[car c'est l'intervalle pertinent pour les fonctions qui seront définies grâce à cette base.

```
baseGR =
 Plot[Evaluate[Table[N_{i,d,K}[u], {i, 0, 5}]], {u, 0, 3},
  PlotRange → {{-0.1, 3.1}, {-0.1, 1.1}},
  PlotStyle → Table[
    {Hue[i / 6], Dashing[{i / 100, i / 200}]}, {i, 1, 6}]]
```




```
- Graphics -
```

■ Fonction Spline

Exprimées sous forme de combinaison linéaire des fonctions de bases, elle fournit une fonction polynomiale par morceaux définie sur intervalle $[U_d,\ U_{\text{Length}[K]-d-1}[$:

$$\text{Bs}_{p\_,U\_,P\_}[u\_] := \sum_{i=0}^{\text{Length}[U]-p-2} N_{i,p,U}[u]\ P_i$$

■ Exemple

```
P = {2, 3, 4, 3.5, 3, 3.5}
```

```
{2, 3, 4, 3.5, 3, 3.5}
```

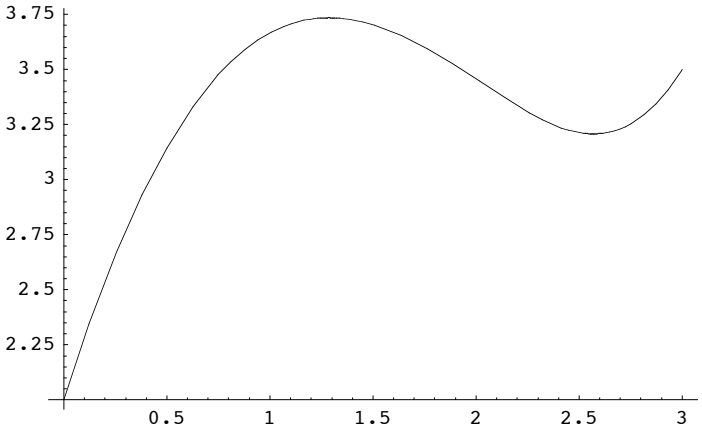
Les bases étant ainsi définies cela entraîne une propriété géométrique intéressante, les coefficients extrêmes sont interpolés

```
{Bs_{d,K,P}[0], Bs_{d,K,P}[3]}
```

```
{2, 3.5}
```

Le tracé de la fonction correspondante sur l'intervale [0,3]

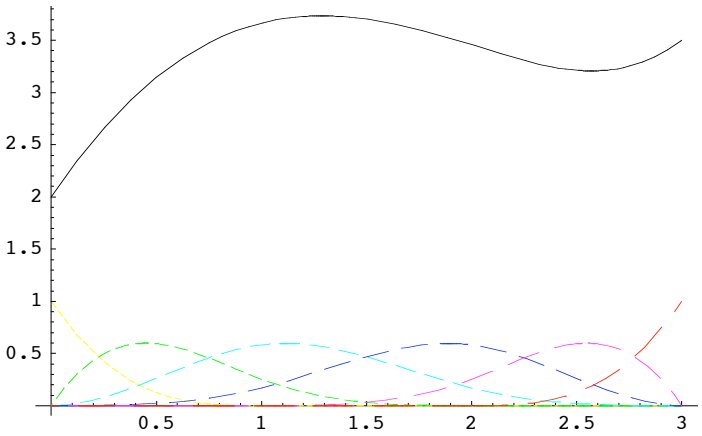
```
fonctionGR =
 Plot[Bs_{d,k,P}[u], {u, 0, 3}]
```



- Graphics -

Nous obtenons le tracé combiné avec les fonctions de base:

```
Show[{baseGR, fonctionGR},
 PlotRange → All]
```



- Graphics -

■ Courbe paramétrique

Les splines deviennent vraiment intéressantes lorsqu'elles sont utilisées pour décrire des courbes ou surface paramétriques, c'est à dire un ensemble de points du plan, ou de l'espace, dont chaque composante est une fonction B-Spline exprimée dans la même base de fonctions splines.

Pour une courbe paramétrique B-Spline dans l'espace R^n , dont les coefficients sont des points de l'espace R^n sous la forme :

$$\{\{x_0,\ y_0,\ z_0\},\ \{x_1,\ y_1,\ z_1\},\ \ ...,\ \{x_{\text{Length}[K]-d-2},\ y_{\text{Length}[K]-d-2},\ z_{\text{Length}[K]-d-2}\}\}$$

Nous avons simplement la généralisation de l'expression des fonctions réelles :

$$\mathbf{BsP_{d\_,U\_,P\_}[u\_] := \sum_{i=0}^{Length[U]-d-2} N_{i,d,U}[u]\ P_i}$$

■ Exemples

■ Une courbe dans le plan

Les coordonnées dans le plan, que l'on nomme 'polygone descripteur' :

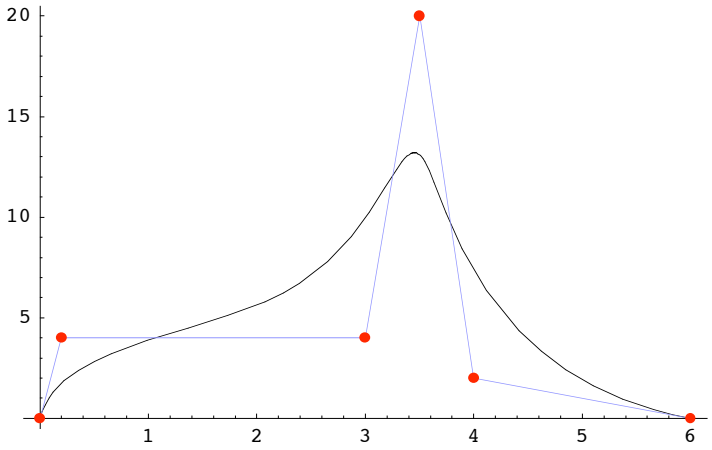
```
P2D =
  {{0, 0}, {0.2, 4}, {3, 4}, {3.5, 20}, {4, 2}, {6, 0}};
```

Un tracé silencieux

```
polyGR = ParametricPlot[BsP_{d,K,P2D}[u], {u, 0, 3},
   DisplayFunction → Identity];
```

La combinaison avec le polygone descripteur :

```
Show[{
  polyGR,
  Graphics[
   {RGBColor[0.5, 0.5, 1], Line[P2D], RGBColor[1, 0, 0],
    AbsolutePointSize[5], Map[Point[#] &, P2D]}]
 },
 DisplayFunction → $DisplayFunction
]
```




```
- Graphics -
```

■ Une courbe dans l'espace

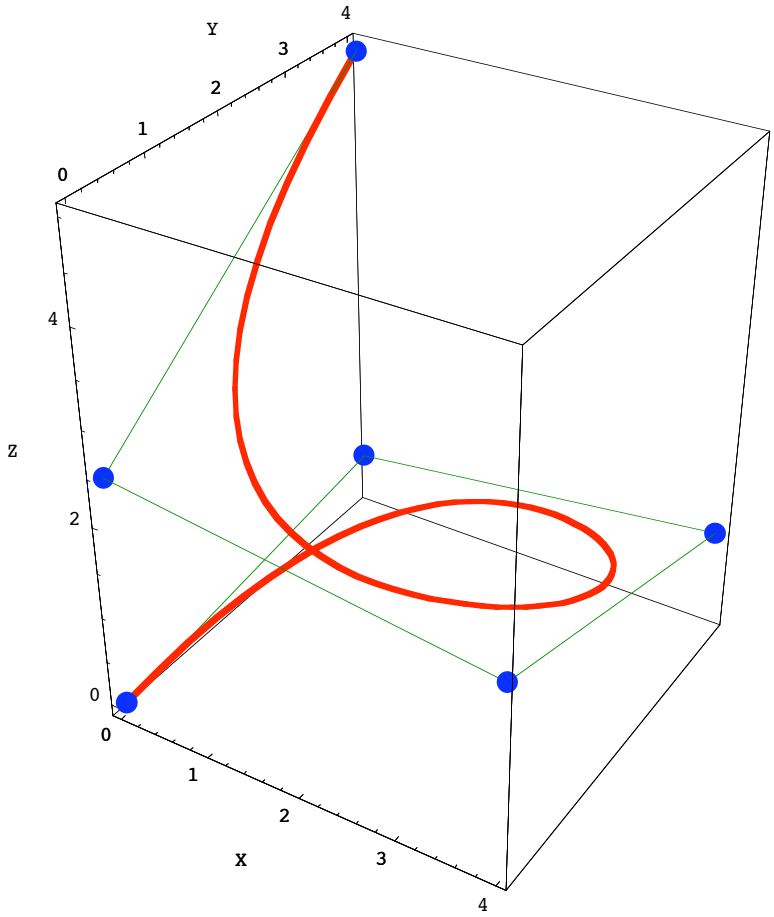
Simplement les coefficients sont définis dans l'espace:

```
P3D =
  {{0, 0, 0}, {0, 4, 0.5},
   {4, 4, 1}, {4, 0, 2}, {0, 0, 2.5}, {0, 4, 5}};

curve3DGR = ParametricPlot3D[BsP_{d,K,P3D}[u], {u, 0, 3},
  DisplayFunction → Identity]

- Graphics3D -
```

```
Show[
 Graphics3D[{
   RGBColor[0, 0.5, 0], Line[P3D],
   RGBColor[0, 0, 1],
   PointSize[0.03], Map[Point, P3D],
   RGBColor[1, 0, 0], Thickness[0.0075], curve3DGR[[1]]
  }],
 Axes → True,
 AxesLabel → {"X", "Y", "Z"},
 ViewPoint -> {1.502, -2.415, 1.833}]
```



- Graphics3D -

■ Surface paramétrique

Une surface paramétrique B-Spline est fonction de deux paramètres u et v, produit de chacune des 'directions' :

$$\mathrm{BsS}_{du\_,U\_,dv\_,V\_,P\_}[u\_,\ v\_] := \sum_{i=0}^{\mathrm{Length}[U]-du-2} \sum_{j=0}^{\mathrm{Length}[V]-dv-2} N_{i,du,U}[u]\, N_{j,dv,V}[v]\, P_{i,j}$$

Les deux directions sont indépendantes, elle peuvent avoir un degré et un vecteur nœud différent...

■ Exemple (la figure de la première page)

Inclut le package Graphics3D pour le tracé de matrice de points obtenus par l'évaluation de la surface en (u,v) :

```
<< Graphics`Graphics3D`
```

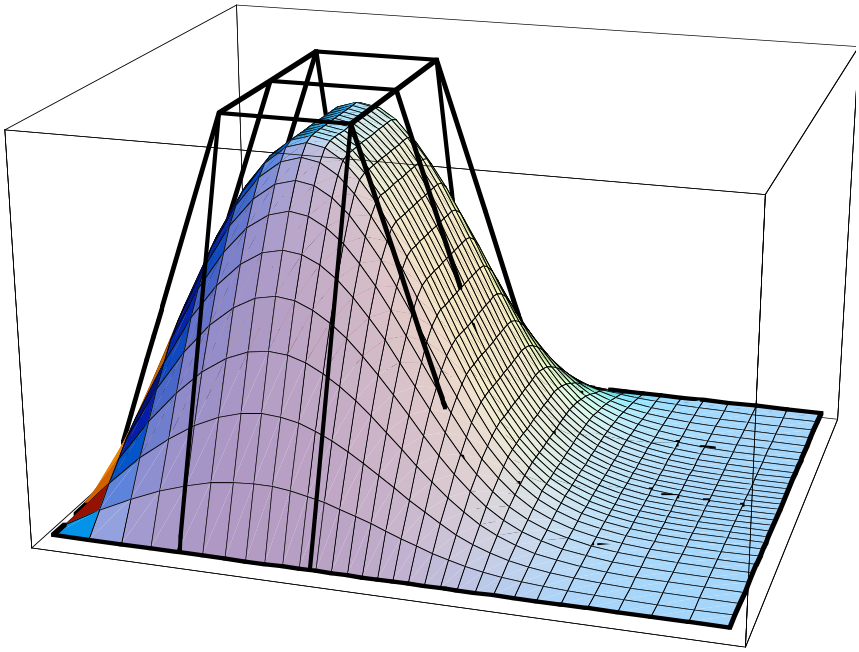
Ici le 'polyèdre descripteur' :

```
Poly3D = {
   {{0, 0, 0}, {0, 1, 0}, {0, 2, 0}, {0, 3, 0}, {0, 4, 0}},
   {{1, 0, 0}, {1, 1, 3}, {1, 2, 3}, {1, 3, 3}, {1, 4, 0}},
   {{2, 0, 0}, {2, 1, 3}, {2, 2, 3}, {2, 3, 3}, {2, 4, 0}},
   {{3, 0, 0}, {3, 1, 0}, {3, 2, 0}, {3, 3, 0}, {3, 4, 0}},
   {{4, 0, 0}, {4, 1, 0}, {4, 2, 0}, {4, 3, 0}, {4, 4, 0}},
   {{5, 0, 0}, {5, 1, 0}, {5, 2, 0}, {5, 3, 0}, {5, 4, 0}}
  };

surfaceGR =
 ListSurfacePlot3D[
  Table[BsS3,{0,0,0,0,1,2,3,3,3,4},2,{0,0,0,1,2,3,3,4},Poly3D[u, v],
   {u, 0, 3, 0.1}, {v, 0, 3, 0.1}],
  DisplayFunction → Identity]

- Graphics3D -
```

```
Show[{
  Graphics3D[{Thickness[0.005], Map[Line, Poly3D],
    Map[Line, Transpose[Poly3D]]}],
  surfaceGR
 },
 ViewPoint -> {0.934, -3.059, 1.104},
 DisplayFunction → $DisplayFunction]
```



- Graphics3D -

■ Conclusion

Il reste beaucoup d'expériences à réaliser avec ces quelques formules. En premier lieu il est possible de jouer sur l'espacement entre les éléments du vecteur nœud, ainsi que leur multiplicité.

On peut également définir assez simplement une extension de ce modèle qui autorise la modélisation exacte de courbes et surfaces conique, en passant au modèle des B-Spline rationnelles. On pourrait obtenir de la sorte la représentation exacte de surface de révolution, de cones, sphères, cercles et ellipse...

Ce modèle de repésentation de formes 'naturelles' est très populaire en image de synthèse pendant de nombreuses années, mais depuis peu un nouveau modèle est

utilisé, c'est celui des 'subdivision surfaces' qui autorise en particulier des topologies plus variées que les splines. Mais ceci est une autre histoire...

Références

1. Les Piegl,Wayne Tiller,The Nurbs Bool,Springer-Verlag. ISBN:3-540-55069-0

2. Gerald Farin, Curves And Surfaces for CAGD, Academic Press, Inc. ISBN : 0-12-249052-5

Autres références:

Pierre Bézier, How Renault uses numerical control for car body design and tooling,
Paper SAE 6800010 Society for Automotive Engineers, 1968.

P de Casteljau, Courbes et Surfaces\`a poles, Rapport technique,Citroen,Paris 1963.

Thèse de R.F. Riesenfeld, Application of B-splines Approximation to Geometric Problems of Computer Aided Design.
Syracuse U 1973. Available at University of Utah.

Brian A. Barsky, Computer Graphics and Geometric Lodeling Using Beta-splines, Springer-Verlag.
ISBN:3-540-70006-4

G. Iglesias, Modélisation surfacique et dessin de courbes sur surfaces. Thèse de doctorat, Paris XI, 1992.

Des preuves de programmes à la correspondance du Curry-Howard

Rémi Barrère
rbarrere@ensmm.fr

Partant du thème classique des preuves de programmes (spécifications logiques, domaines, préconditions…), nous aboutissons à la correspondance de Curry-Howard qui établit un lien entre logique, ensembles et algorithmes. Nous constatons que Mathematica concourt à traiter ces questions avec clarté et concision.

■ Introduction

On ne peut pas démontrer (au sens hypothético-déductif) qu'un programme réalise correctement un objectif énoncé en termes concrets, pas plus qu'on ne peut prouver qu'une théorie mathématique décrit correctement la réalité physique. On sait cela depuis l'affaire des géométries non-euclidiennes, bien que nombre de théoriciens argumentent comme s'ils l'ignoraient. En revanche, on peut démontrer des propriétés du programme qui amèneront la conviction que le programme fait bien ce pour quoi il a été conçu. On appelle spécification logique l'énoncé des propriétés attendues d'un algorithme ou d'une fonction.

Généralement, on présente cela de la façon suivante : des données d'entrée (paramètres) vérifiant la propriété P (précondition) produisent un résultat avec la propriété Q (postcondition) ; prouver (la correction d') un programme (algorithme, fonction), c'est démontrer qu'il est conforme à la spécification, à savoir que pour les paramètres vérifiant la précondition, il se termine et produit un résultat vérifiant la postcondition. En fait, on distingue la correction partielle (si le programme se termine, alors il vérifie la postcondition) de la correction totale (le programme se termine et il vérifie la postcondition).

Le propre d'un programme étant de s'appliquer à une classe de données, on appelle cette classe son domaine et sa description logique n'est autre qu'une précondition ; corrélativement, on appelle codomaine la classe des résultats, pendant ensembliste d'une postcondition. Domaine est ici employé au sens de domaine de définition ; on le rencontre souvent (ainsi que classe) au sens d' ensemble structuré, autrement dit de type. Or on peut tester au moyen d'un filtre l'appartenance d'un paramètre à une classe, ce qui nous donne opportunément le moyen d'exprimer directement les préconditions dans les programmes. Par exemple, f[k_Integer?Negative] désigne une fonction de domaine $\mathbb{Z}^-$, autrement

dit un algorithme de précondition $k \in \mathbb{Z}^-$. Toutes les constructions de filtres peuvent servir : tête, structure, prédicat, condition ainsi que leurs combinaisons.

```
abs[x:(_Integer|_Rational|_Real)]:= If[x>=0, x, -x]

norm[{x_,y_,z_}]:=Sqrt[x^2+y^2+z^2]

rectangle[x_/;-1<x<1]:= 1 ; rectangle[x_]:= 0
```

Une fois intégrées aux définitions, les préconditions sont testées au moment de l'évaluation. Dans une certaine mesure, cela confère à Mathematica la qualité de langage de spécification exécutable ; il se rapproche ainsi de langages comme B ou Z, tout en restant à distance respectueuse en raison de sa philosophie de typage dynamique. On n'explicite généralement pas la postcondition ; elle est vérifiée de fait si l'algorithme a été prouvé. Cela dit, surtout en phase de développement, rien n'interdit d'associer à une définition un test de postcondition, éventuellement assorti d'un message d'avertissement. On peut d'ailleurs mentionner ici le langage Eiffel qui incorpore un mécanisme de test de postcondition.

```
nsqrt[x_?NumberQ] :=
    With[{t = FixedPoint[(# + x / #) / 2 &, N[x]]},
        Switch[t,
            _Real?Positive, t,
            _, {$Warning, t}]
    ]

Map[nsqrt, {2, 2., -I}]

{1.41421, 1.41421, {$Warning, 0.707107 - 0.707107 i}}
```

Bien entendu, un filtre ne correspond pas forcément au domaine de la fonction ; il peut être plus restrictif (on parle alors de surspécification) ou plus laxiste. Nous ne devons d'ailleurs pas négliger pour autant la question incidente : que faire lorsque la précondition n'est pas vérifiée ? Une solution brutale consiste à tout arrêter en expédiant de préférence un message d'erreur. Cependant, en programmation symbolique, on choisit souvent de poursuivre le calcul avec un terme non évalué ; par exemple Abs est calculée pour un paramètre numérique mais reste en l'état si c'est un symbole indéfini. Ainsi nous pouvons manipuler des expressions avec indéterminées. De façon générale, le niveau de rigueur résulte d'un compromis : une spécification stricte augmente la fiabilité au détriment de la généricité ; pour certains usages prospectifs, on adopte une attitude plus laxiste.

■ Sémantique pré-post

La sémantique pré-post vise à établir des raisonnements logiques sur les programmes ; on la dit aussi axiomatique mais ce qualificatif est ambigu car la sémantique dénotationnelle est aussi de nature axiomatique. Bien qu'elle fût originellement développée pour décrire les programmes procéduraux, nous en donnons une version adaptée aux constructions fonctionnelles. Elle est fondée sur les formules dites pré-post ou triplets de Hoare, souvent notés {P} f {Q} mais que, le plus souvent, nous noterons simplement P f Q ; elles expriment que si les paramètres p_i de f vérifient la propriété P (précondition) alors son résultat f[…p_i…] vérifie la propriété Q (postcondition). Cela conduit à une approche contractuelle des programmes : la correction d'un programme est relative à une précondition et une postcondition ; le programmeur garantit que si P est vérifiée par les données, alors Q sera vérifiée par le résultat, ce que traduit la formule P f Q.

Le modus ponens des preuves de programmes

$$\frac{P,\ P\ f\ Q}{Q}$$

Dans les démonstrations, outre les règles usuelles de la logique, on recourt à des règles de déduction spécifiques au domaine. Un analogue du modus ponens stipule que si P est démontré ainsi que P f Q (le programme est prouvé), alors Q est un théorème. D'autres règles caractérisent les constructions algorithmiques fondamentales, par exemple de P f Q et Q g R on déduit P gof R.

Quelques règles pour les preuves de programmes

$$\frac{P\ f\ Q,\ Q\ g\ R}{P\ gof\ R} \qquad \frac{P \wedge C\ f\ Q,\ P \wedge \bar{C}\ g\ Q}{P\ If[C,\ f,\ g]\ Q} \qquad \frac{P\ f\ Q,\ P\ f\ R}{P\ f\ \{Q \wedge R\}} \qquad \frac{P\ f\ Q,\ R\ f\ Q}{\{P \vee R\}\ f\ Q}$$

Il y a également des règles pour les structures d'itération. L'une des règles, dite de conséquence, s'énonce ainsi : de P' ⇒P, P f Q et Q ⇒Q' on déduit P' f Q'.

$$\frac{P' \Rightarrow P,\ P\ f\ Q,\ Q \Rightarrow Q'}{P'\ f\ Q'}$$

De cette règle, il résulte qu'une multitude de couples (P, Q) caractérisent une même fonction mais la formule est d'autant plus intéressante que P est faible et Q forte. Le plus souvent, Q décrit l'objectif du programme et l'on cherche à établir la précondition P la moins restrictive possible de façon à délimiter le plus grand domaine possible ; on dit que l'on cherche la plus faible précondition.

```
abs[x_?NumberQ /; x ≥ 0] := x
abs[x_?NumberQ /; x < 0] := -x
```

Par exemple, il est clair que : {x⩾0} abs {abs[x]⩾0} ; mais la postcondition abs[x]⩾0 est aussi garantie par la précondition plus faible x:(_Integer|_Rational|_-Real). En fait, abs[x]⩾0 ne caractérise pas abs de façon pertinente ; la postcondition qui nous intéresse est que le résultat vaut bien | x | (que la définition informatique traduit bien la notion mathématique). Mais qu'est-ce qui caractérise | x | sinon que x≥0⇒abs[x]=x et x<0⇒abs[x]=-x, autrement dit sa définition même ? En construisant un programme comme on définit une fonction mathématique, on lui assure un même statut : sa preuve réside dans sa définition.

■ Correspondance de Curry-Howard

Cette idée que la preuve d'un programme se trouve dans sa construction même a donné lieu à une présentation appelée isomorphisme (ou plus prudemment correspondance) de Curry-Howard. Elle se fonde sur la concordance formelle entre les règles de déduction :

$$\frac{P,\ P\ f\ Q}{Q} \quad \text{et} \quad \frac{P,\ P \Rightarrow Q}{Q}$$

À la précondition correspond l'hypothèse, à la formule pré-post l'implication et à la postcondition la conclusion. On identifie alors le programme à la preuve, ce qui conduit au slogan : « programmer, c'est prouver ». D'une certaine façon, cela justifie l'attitude (pourtant critiquée) des programmeurs qui snobent les approches théoriques. De fait, on ne recourt aux preuves de programmes qu'en milieu professionnel lorsque des critères de fiabilité l'imposent ou dans le monde académique pour entraîner les étudiants à associer mentalement le raisonnement logique à la construction des programmes ; et accessoirement pour gagner quelques publications faciles. Il ne faut d'ailleurs pas perdre de vue que le problème n'est pas de déduire Q mais de trouver f (l'algorithme) connaissant P et Q, par exemple, exhiber un programme transformant une liste quelconque en liste triée.

Cette correspondance fait toucher du doigt l'idée sous-jacente qu'un système de calcul formel, comme un système formel, est un outil de concentration (synthèse au sens de la logique) de l'information, le rôle d'un calcul comme d'une déduction étant de reconstituer (synthèse au sens de l'ingéniérie) l'information. Encore reste-t-il à préciser que le présent exposé résulte d'une adaptation de la formulation originelle en terme de types, laquelle s'énonce ainsi : si la donnée p est de

type τ_1 et la fonction f de type $\tau_1 \longrightarrow \tau_2$, alors le résultat f[p] est de type τ_2 ; ce que l'on note :

$$\frac{p : \tau_1 \ , \ f : \tau_1 \longrightarrow \tau_2}{f[p] : \tau_2}$$

À la précondition correspond le type de la donnée, à la spécification (formule pré-post) le type de la fonction et à la postcondition le type du résultat. Type est ici synonyme de classe, si bien que cette formulation n'est autre qu'un pendant ensembliste des formules pré-post. Quelle que soit la formulation adoptée, il apparaît que la preuve d'un programme fait partie intégrante de son écriture ; de sorte que des réalisations différentes expriment des démonstrations différentes ; un programme correct n'est autre qu'une formule juste et une erreur de programme est de même nature qu'une erreur mathématique …

… des mathématiques enrichies de l'idée de processus d'évaluation. Car les choses se compliquent un peu dans le cas des définitions itératives, récursives ou paresseuses. Les premières requièrent des preuves de terminaison. À cause de la suspension (attributs de la famille Hold), les dernières induisent des propriétés inhabituelles des fonctions, qui ne sont plus évaluées à partir des niveaux les plus profonds. Les choses se compliquent encore davantage dans le cas des affectations caractéristiques du paradigme procédural, car pour déterminer les propriétés du résultat, il ne suffit plus de se laisser guider par le texte du programme, mais il faut suivre par la pensée les états successifs des variables affectées.

Pour les cas ordinaires, les programmes étant des compositions de fonctions, on les valide en les évaluant mentalement pas à pas comme on déroule une démonstration. Quant à la terminaison, elle est prouvée si l'on est assuré de la terminaison des fonctions invoquées dans le corps de la définition ; de sorte que les preuves de terminaison sont implicites. Considérons par exemple la fonction réelle f[x_,y_]:=Sqrt[x/y]. Sqrt est définie (et se termine) pour x y≥0 ; la fraction est définie (et se termine) pour y≠0 ; f est donc définie et se termine pour x y≥0∧ y≠0. Pour les définitions itératives ou récursives, la preuve découle du principe de récurrence.

$$\frac{P[n_0] \ , \ \forall \ n \geq n_0 \ , \ P[n] \Rightarrow P[n+1]}{\forall \ n \geq n_0 \ , \ P[n]}$$

Prenons l'exemple de la factorielle ; avec les quantifications sous-entendues, on peut écrire :

$$\frac{fact[0] = 1 \ , \ fact[n] = n! \Rightarrow fact[n+1] = (n+1)\ fact[n] = (n+1)!}{fact[n] = n!}$$

On prouve l'implication en posant fact[n+1]=(n+1) fact[n] ; le programme de la factorielle fact[0]=1 ; fact[n_]:= n fact[n-1] réside dans la preuve, ce qui donne raison au dicton : « C'est en prouvant la spécification que l'on trouve un programme qui la satisfait ». Celui-ci nous remémore la belle formule de Rignano : « Un raisonnement est une suite d'expériences simplement pensées, c'est-à-dire d' expériences que nous imaginons accomplir sur un ou plusieurs objets donnés … » ; il suffit de remplacer expérience par calcul.

À la quantification ∀ n ≥ 0 , correspond d'ailleurs le filtre n_?Positive (en fait, le filtre n_ correspond au n+1 de la formule). De façon analogue, l'expression mathématique de la factorielle comme produit d'entiers conduit aux opérateurs Product ou Fold et la version itérative indiquée ci-dessous forward[{n_,a_}]:={n+1,n a} découle de la décomposition de la preuve en {s[n,a]==n+1,f[n,a]==n a}.

```
forward[{n_, a_}] := {n + 1, n a}
factComp[n_] := Last[Nest[forward, {1, 1}, n]]

{factComp[0], factComp[5]}
```

```
{1, 120}
```

```
backward[0, a_] := a
backward[n_, a_] := backward[n - 1, n a]
factIter[n_] := backward[n, 1]

{factIter[0], factIter[5]}
```

```
{1, 120}
```

Partant d'une même idée, l'itération est ici réalisée d'abord au moyen d'une composition de fonction, puis d'une définition autoréférente (récursion terminale, autrement dit itération). La méthode consiste à ajouter un paramètre (nommé a dans les programmes ci-dessus) pour accumuler les résultats intermédiaires.

■ Logique, ensembles, calcul

D'aucuns pensent que le calcul est l'affaire de la machine alors que la déduction serait la chasse gardée de l'esprit ; les choses ne sont pas aussi simples car une machine peut déduire. La correspondance donne lieu à un dictionnaire mettant en relation formules logiques et constructions algorithmiques. À terme, on peut espérer en tirer avantage pour traduire des démonstrations en programmes et réciproquement ; un outil de démonstration automatique deviendrait alors un générateur de programmes. Toutefois, les recherches dans ce domaine prometteur n'en sont qu'à leur début.

logique	**algorithmique**
hypothèses (préconditions)	données
implication (spécification)	fonction
preuve	réalisation, programme, règle
conclusion (postcondition)	résultat
modus ponens	application
(méta-) règle de déduction	(méta-) règle d'évaluation
déduction	évaluation (réduction)
démonstration	suivi d'évaluation (Trace)
axiome	primitive ou donnée atomique
théorème	expression ou fonction
lemme	résultat intermédiaire

Aux connecteurs logiques correspondent différents constructeurs et à quelques formules on sait associer des constructions algorithmiques.

logique	**algorithmique**
conjonction	couple, n-uplet
disjonction	alternative, choix
$P \Rightarrow P$	Identity
$P \wedge Q \Rightarrow P$	First
$P \wedge Q \Rightarrow Q \wedge P$	{#[[2]], #[[1]]} &
transitivité de $\Rightarrow$	composition
récurrence	itération ou récursion

Il ne faudrait pas croire pour autant que l'on a découvert une panacée qui va générer automatiquement des programmes ; car il reste pour chaque programme à trouver l'axiomatique qui lui correspond. Prouver un programme peut être difficile, en particulier quand il met en jeu des nombres en représentation machine ou des fonctions polymorphes, ou encore quand il manipule des symboles indéfinis, classiquement hors du champ de l'informatique théorique. Cela requiert attention et rigueur ; malgré cela, on aboutit souvent à des conclusions démenties par la machine, témoignage cinglant de la médiocre fiabilité de l'esprit humain. Nous avons du reste rappelé en début d'article que la preuve mathéma-

tique ne garantit nullement l'adéquation à la réalité physique. C'est pourquoi on complète normalement ces preuves par de nombreux tests.

■ Discussion et conclusion

La correspondance de Curry-Howard, avec cette idée d'assimiler une preuve à une fonction, émane de la mouvance intuitionniste. L'affirmation que « les recherches dans ce domaine prometteur n'en sont qu'à leur début » est donc assortie de la précaution que le temps se mesure ici en décennies.

Cette correspondance entre logique et calcul est assez systématiquement présentée en termes de lambda-calcul, ce qui rend les documents quasiment illisibles, tant le lambda-calcul s'affiche comme un art d'exprimer de façon alambiquée ce qui autrement serait assez simple. On se heurte assez communément dans la littérature scientifique à cette propension à emberlificoter, à ajouter à la complexité naturelle des choses la confusion de l'esprit ; tandis que les authentiques savants, ceux qui s'imposent l'ascèse d'une pensée claire, s'évertuent au contraire à simplifier leur vue du monde en démêlant ce qui apparaît de prime abord intriqué. Certes, les pionniers sont excusables de n'avoir défriché que des chemins tortueux ; mais les suiveurs sont impardonnables de ne faire aucun effort de réfection.

Bref, il faut retirer la gangue stérile d'un formalisme abscons pour faire apparaître en toute netteté la correspondance de Curry-Howard, son articulation à la sémantique pré-post, et dans son sillage l'idée d'une équivalence ou du moins d'un recouvrement entre les domaines de la logique, de la calculabilité et de la théorie des types. Remarquons également que cette correspondance met en relation ce qu'en logique on qualifie du second ordre et ce qui en algorithmique concerne le processus d'évaluation. Ces réflexions laissent entrevoir une approche fondatrice unitaire, qui reste encore à mettre au point, par extraction de ce qui est commun à la logique, aux ensembles et au calcul. Est-il nécessaire de souligner l'aide considérable de Mathematica, lui-même résultat d'un effort d'épurer le fatras langagier et conceptuel accumulé durant les années de jeunesse de l'informatique ?

Enfin, d'une méditation sur cette correspondance dérivent inévitablement des questions fondamentales concernant l'usage du langage comme représentation scientifique de la réalité, les contraintes que cela induit et les conséquences que cela entraîne. Ces interrogations et d'autres apparentées sont discutées sur la liste électronique pim (pim@ml.free.fr).

■ Bibliographie

R. Barrère, *Mathematica*, Vuibert, 2002

J. Chazarain, *Programmer avec Scheme*, Vuibert, 1996

J-Y. Girard, Constructivité, vers une dualité moniste, 1996,
[en ligne] http://iml.univ-mrs.fr/~girard/Articles.html

J.-L. Krivine, Ensembles et preuves, 1997, [en ligne]
http://www.logique.jus-sieu.fr/www.krivine/index.html

J.-L. Krivine, Mathématique des programmes et programme des mathématiques, 1992, [en ligne] http://www.logique.jus-sieu.fr/www.krivine/index.html

B. Meyer, *Introduction à la théorie des langages de programmation*, InterÉditions, 1992

J.-F. Monin, *Comprendre les méthodes formelles*, Masson, 1996

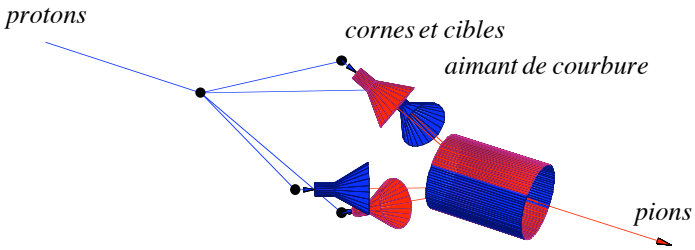
Dessin scientifique avec *Geometrica*

Bruno Autin
bautin1@club-internet.fr

Toute la puissance du traitement de listes, de la reconnaissance de formes et de la programmation fonctionnelle est systématiquement employée dans le logiciel *Geometrica* pour produire des dessins qui ont toute la rigueur de la géométrie et sont faciles à exécuter lorsqu'ils sont codés sous forme de fonctions comportant généralement un nombre limité d'arguments.

■ Introduction

Cet exemple est tiré d'une conférence sur un système d'accélérateurs dédié à la production de faisceaux intenses de neutrinos. Ces machines, appelées *usines à neutrinos* [1], utilisent un synchrotron à protons qui délivre un faisceau de plusieurs mégawatts à plusieurs cibles. Dans les cibles, une partie de l'énergie cinétique des protons est convertie en création de pions. Les pions ont une durée de vie brève et se désintègrent en muons et en neutrinos de muon dans un tunnel situé en aval des cibles. Le faisceau de neutrinos peut être utilisé tel quel mais a une large distribution en angle et énergie. La qualité du faisceau de neutrinos est améliorée si l'on exploite la désintégration des muons qui ont une durée de vie beaucoup plus longue que celle des pions, Les muons sont alors rapidement accélérés à plusieurs dizaines de GeV puis stockés dans un anneau où ils se désintègrent en électrons et en deux familles de neutrinos, les neutrinos d'*électron* et de *muon* qui ont une excellente définition en angle et énergie bien adaptée aux expériences d'oscillations neutrino. Une difficulté majeure des usines à neutrinos est la conception des cibles qui sont soumises à de violentes contraintes thermiques puisqu'environ un quart de l'énergie des protons est dissipée dans les cibles de faible volume. Une méthode prometteuse consiste à diviser l'énergie du faisceau de protons en le distribuant séquentiellement sur quatre cibles..



Les diverses étapes de la construction de la figure ci-dessus sont expliquées en détail en commençant par les objets 2D, puis en immergeant ces objets dans l'espace 3D, en traitant la totalité ou des parties de surface et en appliquant couleurs et légendes.

■ Trajectoires bi-dimensionnelles

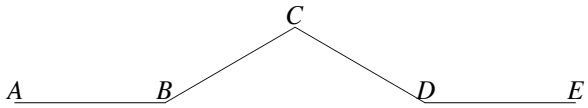
Geometrica est appelé avec la syntaxe employée pour toute application de *Mathematica*. *Geometrica02* est la version de *Geometrica* créée en 2002 et disponible commercialement.

```
<< Geometrica`Geometrica02`
```

La géométrie du système est déterminée par trois aimants de courbure. Les deux premiers sont supposés infiniment courts (approximation des *lentilles minces*) et le troisième a une longueur finie. La déflexion due aux premier et troisième aimants est ϕ et celle de l'aimant du mileu -2ϕ. De plus le rayon de sortie est colinéaire au rayon d'entrée. Une première esquisse de la trajectoire d'une particule émise en (0,0) est donnée par la ligne brisée *s*. Les sommets sont définis par la fonction "listable" *CPoint* dont les arguments sont la liste des abscisses et la liste des ordonnées. La listabilité est une propriété de la plupart des fonctions de *Geometrica*. La fonction *Segment* est générique pour toute ligne polygonale et non pas seulement pour le segment habituel défini par deux points. La légende

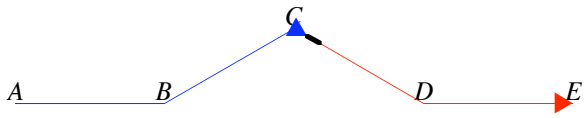
peut être automatique en spécifiant dans la fonction *Legend* une file de caractères correspondant au nom des sommets. L'option *Offset* ajuste la légende en coordonnées polaires autour du point de référence.

```
phi = Pi / 6;
l = 30;
{a, b, c, d, e} =
  CPoint[{0, l, l (1 + Cos[phi]),
    l (1 + 2 Cos[phi]), 2 l (1 + Cos[phi])},
   {0, 0, l Sin[phi], 0, 0}];
s = Segment[a, b, c, d, e];
te = Legend["ABCDE", s, Offset → {1, Pi / 2}];
Draw[te, s];
```



La cible est placée sur le segment *CD* aux points de paramètres 0.1 et 0.2 avec la fonction *Pointer* qui assure que les points sont liés au segment. Les paramètres 0 et 1 correspondent à l'origine et à l'extrémité respectivement. Dans la figure, les protons se déplacent sur la flèche bleue et les pions sur la flèche rouge. La cible qui est une simple barre de métal est représentée par un segment noir épais.

```
sCD = Segment[c, d];
{λ1, λ2} = {0.1, 0.2};
{t1, t2} = Pointer[sCD, {λ1, λ2}];
{s1, s2, s3} = {Segment[a, b, c, t1],
   Segment[t1, t2], Segment[t2, d, e]};
Draw[te, Blue, Arrow[s1], Red, Arrow[s3],
  Black, AbsoluteThickness[2], s2];
```

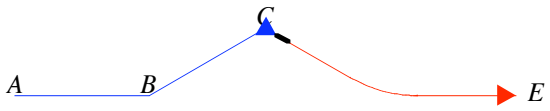


La trajectoire doit être affinée en remplaçant l'angle vif *D* par un arc de cercle. L'angle de déflexion est ϕ et le rayon de l'orbite ρ. Le centre de l'arc est donc le point K ($x_D + \rho$ tg $\phi/2$, ρ), l'extrémité de l'arc est D_2 ($x_D + \rho$ tan $\phi/2$, 0) et son origine D_1 est déduite de D_2 par la rotation de D_2 autour de K d'angle $-\phi$. La forme détaillée de l'arc est un point paramétré (*PPoint*) dont les coordonnées sont des fonctions pures du paramètre muet *#1* dans l'intervalle défini par la règle *PRange*. Cet usage des fonctions pures peut sembler étrange dans un programme de dessin mais il a l'avantage d'éviter le nom de la variable qui, de toute façon,

serait arbitraire. La fonction *Segment* accepte aussi bien des points cartésiens (*CPoint*) que des points paramétrés. Ainsi les lignes définies par *Segment* peuvent comprendre des segments rectilignes comme des segments courbes et subir globalement des opérations de transformations géométriques, de mesure ou de dessin. La légende est scindée en deux nouvelles légendes. Si certains sommets, comme ici le dernier sommet de s_1 ne doivent pas être nommés, il suffit d'introduire un blanc dans le texte à l'endroit du sommet omis. L*e* segment s_3 pourrait aussi être étiqueté mais l'arc est fait de nombreux segments élémentaires (40) et seul le dernier sommet est mentionné.

```
rho = 30;
x = rho Tan[phi / 2];
k = CPoint[x + d[[1]], rho];
d2 = Simplify@CPoint[x + d[[1]], 0];
d1 = Simplify@Rotate[d2, k, -phi];
ar = Arc[k, d1, phi]
s3 = Segment[t1, d1, ar, e];
te1 = Legend["ABC ", s1, Offset → {1, Pi / 2}];
te2 = Legend["E", e, Offset → {1, 0}];
Draw[te1, te2, Blue, Arrow[s1], Red,
  Arrow[s3], Black, AbsoluteThickness[2], s2];
```

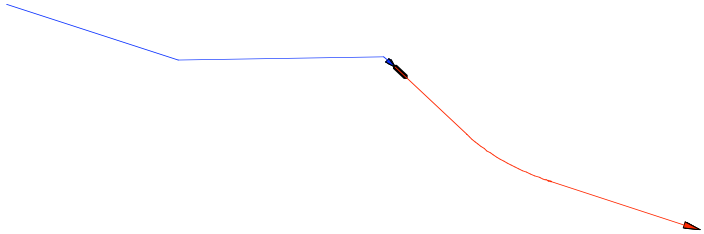
$$\texttt{PPoint}\left[30\,(2-\sqrt{3})+30\,(1+\sqrt{3})+30\,\texttt{Cos[\#1]\&},\ 30+30\,\texttt{Sin[\#1]\&},\ \texttt{PRange}\to\left\{-\frac{2\pi}{3},-\frac{\pi}{2}\right\}\right]$$



■ Immersion d'objets bi-dimensionnels dans l'espace

Le système complet de cibles est de révoluton autour de l'axe *AE*. Tous les éléments qui viennent d'être définis dans le plan doivent appartenir maintenant à l'espace. C'est le rôle de la fonction *To3D* qui ajoute aux points une troisième composante égale à zero. La tête de la première flèche est allongée avec le deuxième argument optionnel d'*Arrow*.

```
{aa, bb, cc, dd, ee, tt1, tt2, ss1, ss2, ss3} =
  To3D[{a, b, c, d, e, t1, t2, s1, s2, s3}];
Draw3D[Blue, Arrow[ss1, 6], Red, Arrow[ss3],
  Black, AbsoluteThickness[2], ss2];
```



■ Corne magnétique

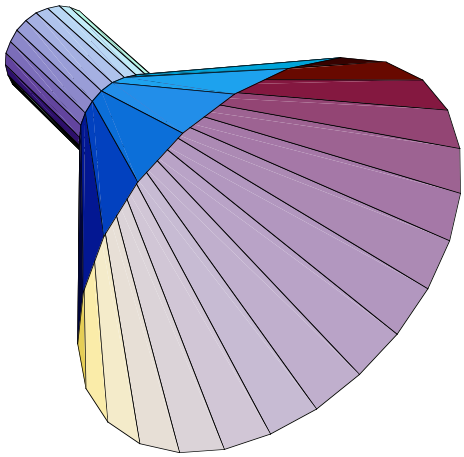
Une corne magnétique est une structure coaxiale qui focalise les particules, ici des pions, autour de son axe. Le champ est confiné à l'intérieur de la corne. Pour une raison de lisibilité, seule l'armature interne de la corne sera dessinée. La corne est une surface de révolution engendrée par la rotation d'une ligne polygonale autour de l'axe de la cible. Le côté intérieur est parallèle à l'axe. La ligne polygonale est d'abord construite dans le plan en utilisant la fonction *FoldList* de *Mathematica* et l'opérateur *Translate* de *Geometrica*. Le point de départ est h_1. Puis une liste *h* de vecteurs de translation est définie. Un vecteur est assimilé à un point cartésien pour préserver la listabilité de *Translate* qui ne fonctionnerait pas si, habituelle convention de *Mathematica*, le vecteur était spécifié par une liste.

```
λ3 = 1 / 5;
θ = Pi / 6;
h1 = Translate[t1, CPoint[Sin[phi] , Cos[phi]]];
h = CPoint[{l (λ2 - λ1) Cos[phi], λ3 l Cos[θ - phi] },
   {-l (λ2 - λ1) Sin[phi], λ3 l Sin[θ - phi]}];
sh = Segment @@ FoldList[Translate[#1, #2] &, h1, h];
Draw[sh, AbsoluteThickness[2], s2];
```

La corne est maintenant aisément créée. L'axe de rotation est *CD* et la surface de revolution définie par *SolidOfRevolution*. Noter que *SurfaceOfRevolution* existe

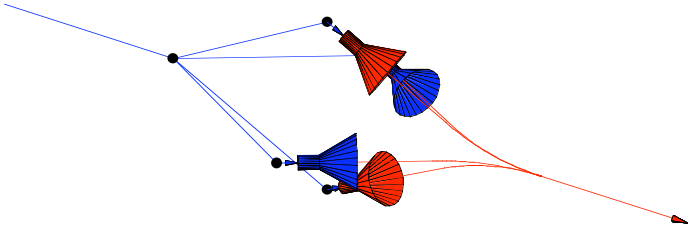
dans l'application *Graphics* de *Mathematica*. La surface peut être coloriée de bien ds façons.Ici c'est la représentation habituelle de *Mathematica* qui est adoptée en combinant *Paint* et l'option *Lighting*.

```
ax = ELine[cc, dd];
ssh = To3D[sh];
horn = SolidOfRevolution[ssh, ax];
Draw3D[Paint[horn], Lighting → True];
```



Les quatre trajectoires et les quatre cornes sont obtenues par rotation de la première ligne de faisceau *bl* autour de l'axe *AE* par pas de $\pi/2$. Le traitement de liste acquiert ici toute sa puissance: la liste de lignes de faisceau est obtenue par application répétée de *Rotate* dans *NestList*. Les éléments de même nature sont dans les colonnes de la matrice *bl4*. Cette matrice est donc transposée pour extraire la liste des éléments qui sont soumis aux mêmes directives graphiques.

```
bl = {cc, ss1, ss2, ss3, horn};
ax = ELine[aa, ee];
bl4 = NestList[Rotate[#, ax, Pi / 2] &, bl, 3];
{cc4, ss14, ss24, ss34,
   {horn41, horn42, horn43, horn44}} = Transpose[bl4];
Draw3D[bb, cc4, Blue, Arrow[ss14, 6],
  Paint[{horn41, horn43}, Blue], Black, ss24, Red,
  Arrow[ss34], Paint[{horn42, horn44}, Red]];
```



■ Aimant de courbure

L'aimant de courbure est fait de feuilles de courant qui créent soit un champ vertical (feuilles haut et bas) soit un champ horizontal (feuilles gauche et droite). Les feuilles sont des secteurs de cylindre (*Cylinder*) de révolution centré en *D* et d'axe *AE*. Le premier argument de la fonction *Cylinder* est le cercle de base, le second la direction des génératrices définie par un point pour la même raison de listabilité expliquée pour *Translate*. Des portions de surface peuvent être créées et manipulées en agissant sur les intervalles de variation des paramètres. L'option *PRange* dans la fonction *PPoint* renvoyée par *Cylinder* controle le domaine de définition des deux paramètres et la résolution des isocourbes.

La représentation paramétrique donnée ici directement par *PPoint* pourrait aussi être obtenue avec des paramètres explicites *u* et *v* en utilisant *Pointer*. Pour un cylindre de révolution, les deux familles d'isocourbes se composent de cercles d'une part et de génératrices d'autre part. La première liste au membre de droite

de *PRange* est l'intervalle de variation de l'angle qui paramétrise le cercle. La seconde liste est l'intervalle de variation du paramètre le long d'une génératrice; l'intervalle par défaut (0,1) signifie que la génératrice est limitée au cercle de base et à l'extrémité du vecteur de définition.

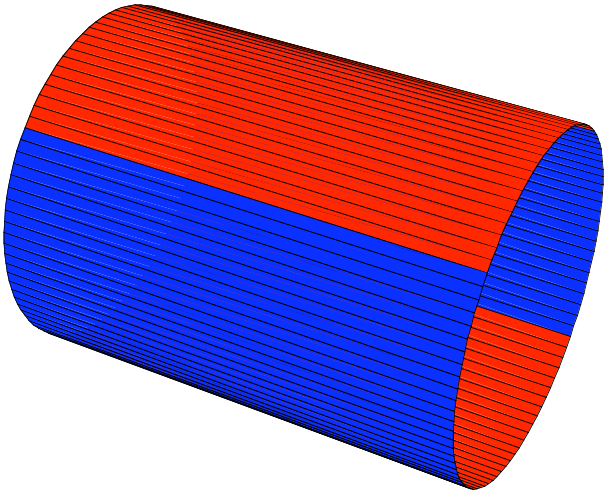
Le cercle de base est défini dans le plan par son centre *D* et son rayon. Après immersion dans l'espace il est tourné dans le plan vertical par rotation de $\pi/2$ autour de l'axe passant par *D* et perpendiculaire à *AE*. On notera que la définition euclidienne d'une droite par *ELine* est "polymorphique"; il existe en effet en géométrie euclidienne une grande variété de définitions de la droite. D'autre part, la conversion dans l'espace d'un cercle ou d'une droite du plan est plus compliquée que la conversion du point. Un cercle, ou plus généralement une conique, est formé de sa repésentation plane et du plan qui la contient. Une droite est l'intersection de deux plans. Dans le cas de *To3D*, un plan est $z = 0$ noté CPlane[0,0,1,0]. Pour la droite , le second plan est le plan vertical qui contient la droite.

```
ci = ECircle[d, λ2 l];
ax1 = ELine[d, ELine[a, e]];
{cci, aax1} = To3D[{ci, ax1}];
cci1 = Rotate[cci, aax1, Pi / 2];
cy = Cylinder[CPoint[1, 0, 0], cci1]
```

```
PPoint[30 (1 + √3) + #2 &, -6. Cos[#1] &,
 6. Sin[#1] &, PRange → {{-π, π}, {0, 1}, {25, 2}}]
```

Les secteurs sont limités par le domaine de variation des paramètres. Chaque secteur a une ouverture de $\pi/2$, le paramètre initial est $-\pi/4$. L'intervalle ($-x$,x), où x a été défini dans la première section (ρ tg $\phi/2$), signifie que le cylindre s'étend le long de l'axe *Ox* et est limité par les plans perpendiculaires à *Ox* d'abscisses (-x,x) puisque la direction des génératrices est donnée par CPoint[1,0,0]. La résolution des isocourbes est laissée à sa valeur par défaut. Chaque secteur contient 25 points par cercle et 2 points, les points d'extrémité, par génératrice. Après connexion des points, chaque secteur cylindrique est représenté par 25 rectangles.

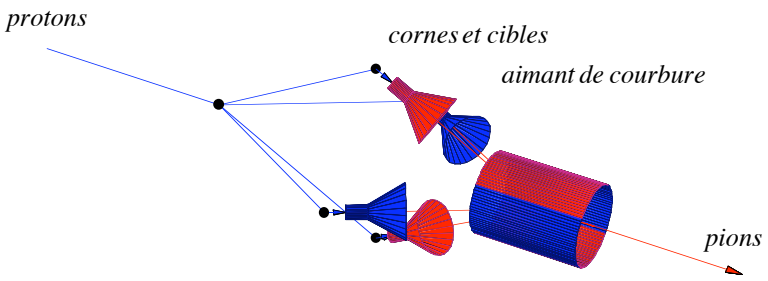
```
cy1 = cy /. (PRange → {x_, y_, z_}) →
     (PRange → {{-1, 1} Pi / 4, {-1, 1} x, z});
cy2 = cy /. (PRange → {x_, y_, z_}) →
     (PRange → {{1, 3} Pi / 4, {-1, 1} x, z});
cy3 = cy /. (PRange → {x_, y_, z_}) →
     (PRange → {{3, 5} Pi / 4, {-1, 1} x, z});
cy4 = cy /. (PRange → {x_, y_, z_}) →
     (PRange → {{5, 7} Pi / 4, {-1, 1} x, z});
Draw3D[Paint[{cy1, cy3}, Blue], Paint[{cy2, cy4}, Red]];
```



■ Assemblage du système entier

Le dessin final incorpore tous les éléments qui viennent d'être élaborés. Des légendes sont ajoutées pour décrire les diverses parties de la figure. Enfin, la couleur des rectangles est modifiée pour adoucir le rendu de la figure. Pour ceci, les cornes et l'aimant de courbure sont d'abord peints pour que l'intérieur des rectangles prenne sa couleur définitive. Puis le contour des rectangles qui est naturellement noir est colorié par superposition des cornes et aimant "bruts" affectés des variantes de bleu et de rouge choisies. Un objet brut dans *Geometrica* est une cage faite de polygones.

```
te = Legend[{"protons", "pions",
     "cornes et cibles", "aimant de courbure"},
   CPoint[{0, 110, 70, 90}, 0, {5, 5, 25, 25}]];
Draw3D[te, bb, cc4, Blue, Arrow[ss14, 6],
  Paint[{horn41, horn43, cy1, cy3}, Blue],
  Black, ss24, Red, Arrow[ss34],
  Paint[{horn42, horn44, cy2, cy4}, Red],
  NavyBlue, horn41, horn43, cy1, cy3,
  VioletRedMedium, horn42, horn44, cy2, cy4];
```



■ Conclusion

L'élaboration d'un dessin scientifique 3D a été décrit en détail. Le code est rendu concis par l'utilisation extensive du traitement de listes: la plupart des fonctions sont "listables", les opérateurs de *Geometrica* peuvent être intégrés dans des générateurs de liste de *Mathematica* tels que *NestList* ou *FoldList*. La souplesse des représentations tient aux trois formes de définition des objets géométriques: cartésienne, euclidienne et paramétrée. L'exemple traité dans cet article pourrait être aisément développé pour inclure une animation du passage des particules ou même des simulations puisque la technique de dessin de *Geometrica* a une précision rigoureuse. Enfin, la méthode acquiert toute sa puissance lorsqu'un dessin doit être répété pour diverses valeurs des paramètres: le code est alors encapsulé dans une fonction et l'appel de cette fonction rend l'exécution du dessin le plus complexe extrêmement simple. Autre avantage: un dessin défini par son code est de poids négligeable et peut être partagé sans difficulté si chaque utilisateur possède le programme de décodage, en l'occurence *Geometrica*. Des exemples de dessin fonctionnel figurent en référence [2].

■ References

[1] Proceedings of the ICFA/ECFA Workshop on Neutrino Factories based on Muon Storage Rings, B. Autin editor, Nuclear Instruments and Methods A, Vol. 451 (2000), No 1.
[2] B. Autin et al., *BeamOptics*, A Program of Analytical Beam Optics, CERN 98-06 (6 August 1998).

Du Moivre dans son assiette

Maryvonne Teissier
my.teissier@free.fr

■ Source

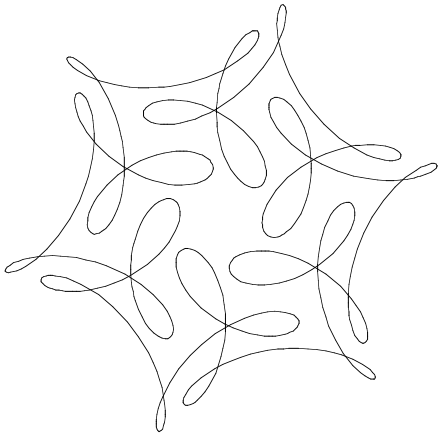
Exo 55 page 602 de la cinquième édition du livre Calculus écrit par Edwards & Penney et publié chez Prentice Hall. Les équations de la première courbe sont créditées dans ce livre à Frank A.Farris de Santa Clara University selon un article paru dans Mathematics Magazine en Juin 1996.

A la vue de cette courbe, obtenue à l'écran en une minute, je me suis rappelé avec émotion mon professeur de mathématiques de Spéciales et ses patientes classifications de courbes. De nos jours, devant la récolte profuse de telles courbes permise par l'ordinateur, ne conviendrait-il pas, jeunes, vieux, créateurs ou tâcherons, d'être plus que de simples consommateurs de ces images que fournirait le hasard, en exerçant notre regard et en jouant les anticipations?

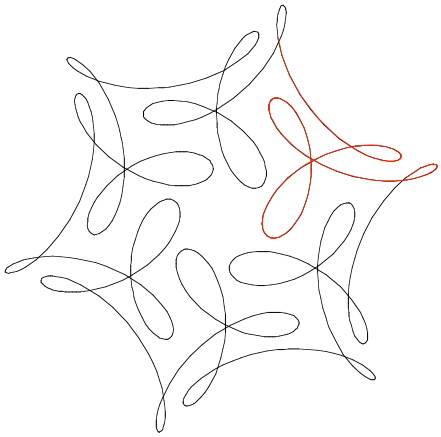
■ Une courbe paramétrée

```
SetOptions[ParametricPlot, Axes → False,
  AspectRatio → Automatic, ImageSize → 216,
  DisplayFunction → $DisplayFunction];
```

```
x[t_] := Cos[t] + Cos[7 t]/2 + Sin[17 t]/3
y[t_] := Sin[t] + Sin[7 t]/2 + Cos[17 t]/3
g1 = ParametricPlot[{x[t], y[t]}, {t, -π, π}];
```



```
g2 = ParametricPlot[{x[t], y[t]},
  {t, 0, Pi / 3}, PlotStyle → Hue[0],
  DisplayFunction → Identity]; Show[{g1, g2}];
```



■ Question 1

Le motif est conservé globalement par une rotation d'angle π /3. Pourquoi ?

■ Réponse : Formule de Moivre

Juste un peu d'histoire pour démêler les raisons de cette attribution.
La formule d' Euler (1748) dit que cos(t)+i sin(t) = $e^{i\,t}$, et cette formule, si simple et si élégante occulte le travail préalable de Moivre (1730) calculant cos(n t) et sin(n t) avec n entier en fonction des coefficients du binôme (Pascal 1654) et de puissances de cost et sint.

$$\cos(n\,t) = \cos^n(t) - \binom{n}{2}\sin^2(t)\cos^{n-2}(t) + \binom{n}{4}\sin^4(t)\cos^{n-4}(t) - \ldots$$
$$\sin(n\,t) = \binom{n}{1}\sin(t)\cos^{n-1}(t) - \binom{n}{3}\sin^3(t)\cos^{n-3}(t) + \ldots$$

Moivre n'utilise pas les imaginaires encore contreversés sinon l'on aurait eu:

$$\begin{aligned}\cos(\mathrm{nt}) + i\sin(\mathrm{nt}) &= \cos^n(t) + \binom{n}{1} i\sin(t)\cos^{n-1}(t) - \binom{n}{2}\sin^2(t)\cos^{n-2}(t) \\ &\quad - \binom{n}{3} i\sin^3(t)\cos^{n-3}(t) + \binom{n}{4}\sin^4(t)\cos^{n-4}(t) - \ldots \\ &= (\cos(t) + i\sin(t))^n\end{aligned}$$

soit, avec l' écriture d'Euler, $e^{i\,n\,t} = (e^{i\,t})^n$. Clef ou corollaire immédiat ?
Et pourquoi taire Ptolémé (150 ans avant Jésus Christ)? La formule de Moivre est l'itération de la formule d'addition des angles pour les sinus et les cosinus que Ptolémé employait déjà. Mon propos est seulement de mettre en perspective les ingrédients d'une démonstration pérenne. En ce qui concerne les relations entre les complexes et leur représentation dans $\mathbb{R}^2$, il faudra attendre Gauss (1799) et Argand (1806). Nous utilisons ici sans sourciller , cette représentation et la formule d'Euler.

Soit ϕ l'application de $\mathbb{R}^2$ dans $\mathbb{C}$ définie par ϕ(x,y) = x + i y.
La définition paramètrique de l'image par ϕ de la courbe précédente est:

$$t \to e^{i\,t} + \frac{1}{2}e^{7\,i\,t} + \frac{1}{3}e^{17\,i\left(\frac{\pi}{2}-t\right)}$$

qui donne

$$t \to e^{i\,t} + \frac{1}{2}e^{7\,i\,t} + \frac{i}{3}e^{-17\,i\,t}$$

et qui s'écrit sous la forme du produit

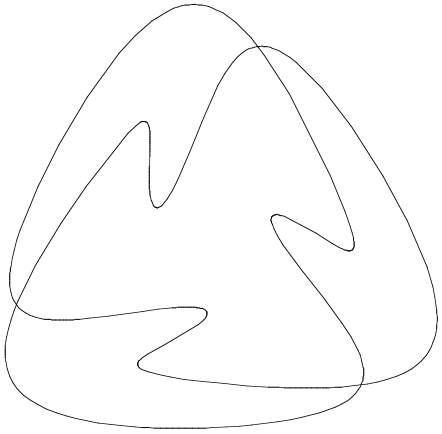
$$t \to e^{i\,t}\left(1+\frac{1}{2}\,e^{6\,i\,t}+\frac{i}{3}\,e^{-18\,i\,t}\right)$$

Le premier terme du produit a pour période 2π et le deuxième terme $\pi/3$. On obtient ainsi l'invariance du motif global par la rotation d'angle π / 3 . La question de l'intervalle minimal d'étude a perdu l'importance qu'il avait pour le tracé de courbe si laborieux dans le passé, mais il suffit de vouloir colorier la courbe pour qu'aussitôt ce problème ressurgisse.

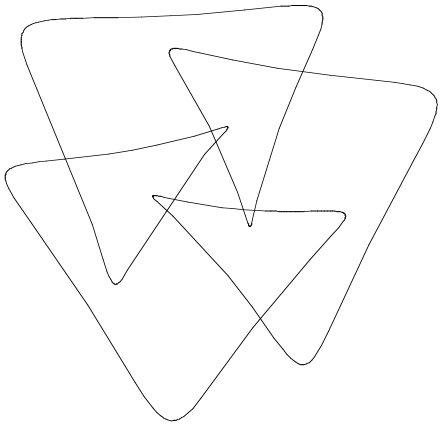
■ Exemples

En vous aidant des pages précédentes, et des exemples qui suivent, trouvez trois courbes paramétrées stables par une rotation d'angle respectivement
2π / 3 , 2π / 5 et 2π / 7.

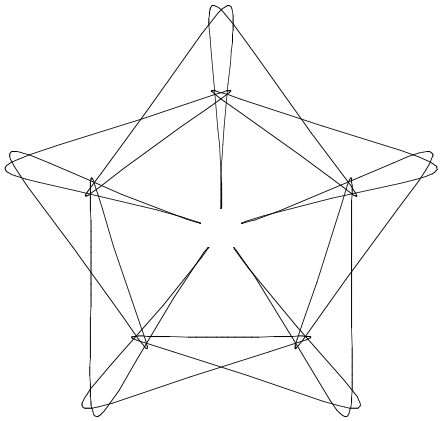
```
Clear[x]; x[t_] := 2 Cos[2 t] + Cos[5 t] - 2 Sin[7 t] / 5
Clear[y]; y[t_] := 2 Sin[2 t] + Sin[5 t] - 2 Cos[7 t] / 5
g3 = ParametricPlot[{x[t], y[t]}, {t, -Pi, Pi}];
```



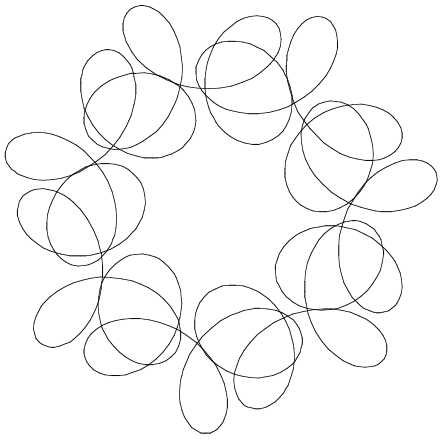
```
x[t_] := Cos[t] + Cos[4 t] + Sin[8 t] / 3
y[t_] := Sin[t] + Sin[4 t] + Cos[8 t] / 3
g4 = ParametricPlot[{x[t], y[t]}, {t, -Pi, Pi}];
```



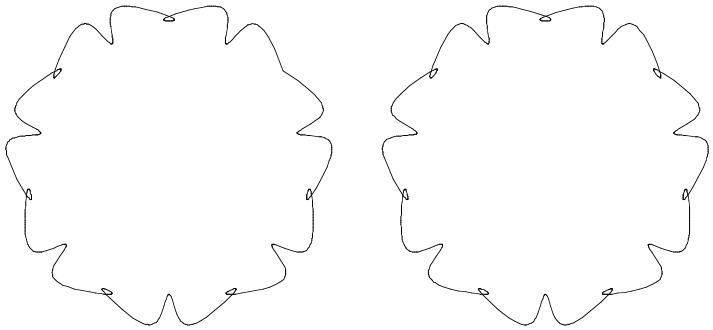
```
x[t_] := Cos[t] + Cos[11 t] / 2 + Sin[14 t] / 3
y[t_] := Sin[t] + Sin[11 t] / 2 + Cos[14 t] / 3
g5 = ParametricPlot[{x[t], y[t]}, {t, -Pi, Pi}];
```



```
x[t_] := Cos[t] + Cos[8 t] / 5 + Sin[20 t] / 3
y[t_] := Sin[t] + Sin[8 t] / 5 + Cos[20 t] / 3
g6 = ParametricPlot[{x[t], y[t]}, {t, -Pi, Pi}];
```

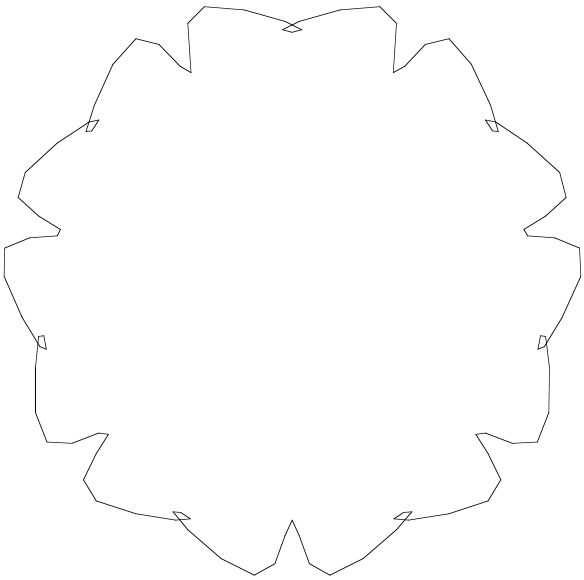


```
x[t_] := Cos[t] + Cos[15 t] / 12 + Sin[20 t] / 30
y[t_] := Sin[t] + Sin[15 t] / 12 + Cos[20 t] / 30
g = ParametricPlot[{x[t], y[t]},
    {t, -Pi, Pi}, DisplayFunction → Identity];
g7 = ParametricPlot[{x[t], y[t]}, {t, -Pi, Pi + 0.001},
  DisplayFunction → Identity];
Show[GraphicsArray[{g, g7}], ImageSize → 360];
```

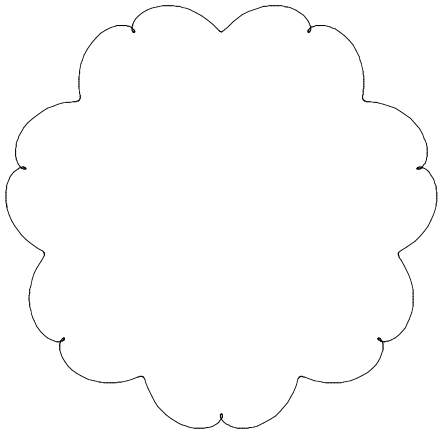


Remarquons le léger défaut apparu au coin en haut à droite de l'image de gauche. Ce défaut disparait en élargissant légérement l'intervalle d'étude pour y inclure strictement π .
Nous retiendrons qu'il y a parfois un écart entre le dessin et l'ordre graphique conduisant à ce dessin, et qu'il faut si possible adapter l'ordre pour accéder à l'image la plus proche de l'image mentale donnée par le raisonnement mathématique..

```
ListPlot[Table[{x[t], y[t]}, {t, -Pi, Pi, 2 Pi / 100}],
  Axes → False, PlotJoined → True,
  AspectRatio → Automatic];
```



```
Clear[x]; x[t_] := Cos[t] + Cos[15 t] / 17 + Sin[6 t] / 21;
Clear[y]; y[t_] := Sin[t] + Sin[15 t] / 17 + Cos[6 t] / 21;
g8 = ParametricPlot[{x[t], y[t]}, {t, -Pi, Pi + 0.01},
   Axes → False, AspectRatio → Automatic];
```



■ Question 2

Le motif de g8 est conservé globalement par une rotation d'angle $\pi / 7$ et, mieux encore, par une symétrie par rapport à une droite (du coup, 7 droites ou 14 demi-droites orientées) . Peut-on lire cela sur l'équation?

■ Réponse: toujours Moivre

La formule d'Euler dit que $cos(t)+i\,sin(t) = e^{i\,t}$. Soit ϕ l'application de $\mathbb{R}^2$ dans $\mathbb{C}$ définie par ϕ(x,y) = x+i y. La définition paramètrique de l'image par ϕ de la courbe précédente est

$$t \to e^{i\,t} + \frac{1}{17}\, e^{15\,i\,t} + \frac{1}{21}\, e^{6\,i\,(\frac{\pi}{2}-t)} = e^{i\,t} + \frac{1}{17}\, e^{15\,i\,t} - \frac{1}{21}\, e^{-6\,i\,t}$$

ce qui s'écrit sous la forme du produit

$$t \to e^{i\,t}\left(1 + \frac{1}{17}\, e^{14\,i\,t} - \frac{1}{21}\, e^{-7\,i\,t}\right)$$

Le premier terme du produit a pour période 2π et le deuxième terme $2\pi/7$. On obtient ainsi l'invariance du motif global par la rotation d'angle $2\pi / 7$.

■ Voyons maintenant les symétries. composante par composante

$$x(t) = \cos(t) + \cos(15\,t)/17 + \sin(6\,t)/21 = -x(\pi - t);$$
$$y(t) = \sin(t) + \sin(15\,t)/17 + \cos(6\,t)/21 = y(\pi - t);$$

D'où la symétrie par rapport à l'axe Oy, visible sur la figure g8 .
Ainsi, la période $2\pi / 7$ peut être rapportée à une hémi-période de $\pi / 7$ à condition de bien choisir l'intervalle d'étude (l'installer avec les images de ses extremités sur deux axes de symétrie consécutifs).

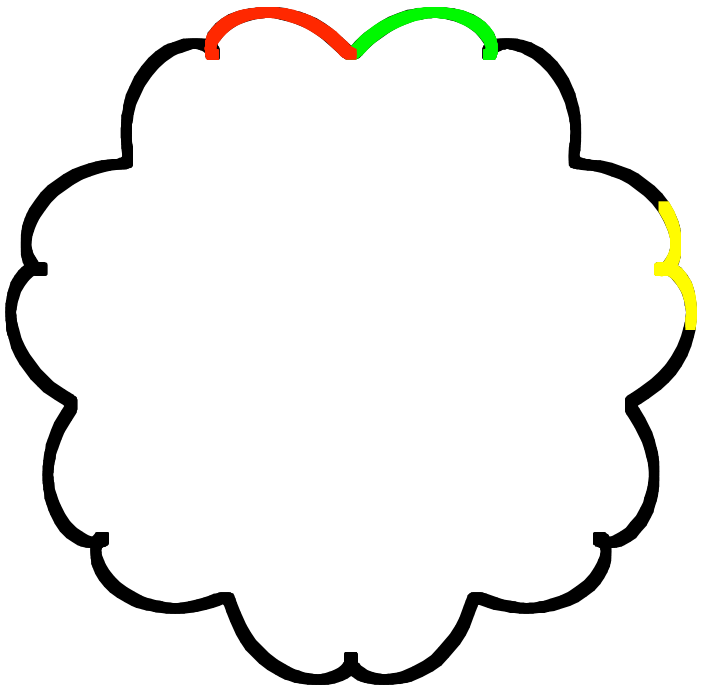
Une symétrie globale de la courbe paramétrée par rapport à l'axe des imaginaires ne signifie pas forcément la symétrie des points correspondants à t et π-t. Tout dépend du paramétrage. Mais c'est le cas ici.

```
SetOptions[ParametricPlot, DisplayFunction → Identity];

montre[g_] :=
  Show[g, DisplayFunction → $DisplayFunction];
```

```
g8 = ParametricPlot[{x[t], y[t]},
    {t, -Pi, Pi + 0.01}, PlotStyle → Thickness[0.015]];
g9 = ParametricPlot[{x[t], y[t]}, {t, 0, Pi / 7},
    {PlotStyle → {Thickness[0.015], Hue[1 / 6]}}];
g10 = ParametricPlot[{x[t], y[t]}, {t, Pi / 2 - Pi / 7,
     Pi / 2}, PlotStyle → {Thickness[0.015], Hue[1 / 3]}];
g11 = ParametricPlot[{x[t], y[t]},
    {t, Pi / 2, Pi / 2 + Pi / 7},
    PlotStyle → {Thickness[0.015], Hue[1]}];

montre[{g8, g9, g10, g11}];
```

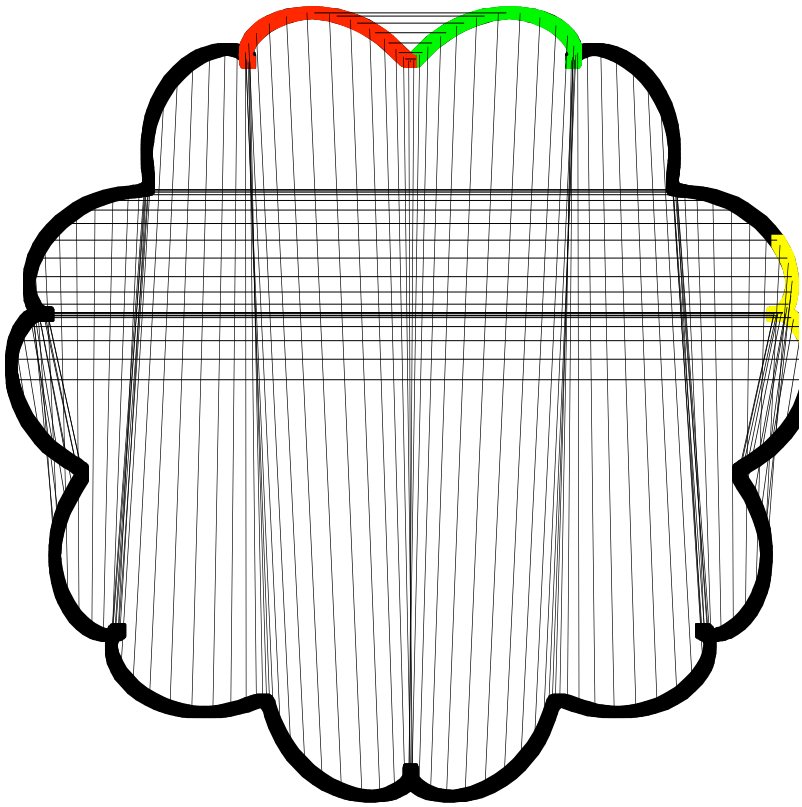


La courbe est stable par ρ, la rotation centrée àl'origine et d'angle $2\pi/7$ et par σ, la symétrie autour de l'axe vertical.

L'intervalle d'étude $[0, \pi/7]$ (d'image jaune) bien que de longueur requise ne convient pas.

Par contre, l' intervalle $[\pi/2 - \pi/7 , \pi/2]$(image verte) et $[\pi/2 , \pi/2 + \pi/7]$ (image rouge) conviennent, pour recouvrir toute la courbe par action de σ et de ρ autant de fois qu'il le faut.

```
g12 = Graphics[
   Evaluate[Table[Line[{{x[t], y[t]}, {x[-t], y[-t]}}],
     {t, -Pi, 0, 0.03}]]];
g13 = Graphics[Evaluate[Table[
     Line[{{x[t], y[t]}, {x[Pi - t], y[Pi - t]}}],
     {t, Pi / 2, 4 Pi / 7, 0.03}]]];
g14 = Graphics[Evaluate[Table[
    Line[{{x[t], y[t]}, {x[Pi - t], y[Pi - t]}}],
    {t, 0, 3 Pi / 14, 0.03}]]];
montre[{g8, g9, g10, g11, g12, g13, g14}];
```



On a rajouté au graphique précédant les lignes presque verticales reliant des couples de points M(t) et M(-t) régulièrement répartis (selon t) en dépit des apparences ,
et quelques lignes horizontales toujours régulièrement réparties (selon t) reliant de couples {M(t),M(π-t)}

■ Résumé

Les courbes ci – dessus ont une équation du type

$$t \to e^{i\,m\,t} + a\,e^{i\,n\,t} + b\,e^{i\,r\,(\frac{\pi}{2}-t)}$$

avec a et b rationnels et m, n et r entiers positifs.

Si m = 1 et si n – m et r + m ont un PGCD égal à k,
alors la courbe est stable par rotation d' angle $2\,\pi\,/\,k$.
De façon plus générale, si le PGCD de m, n – m et r + m est m',
la courbe est stable par rotation d' angle $2\,m'\,\pi\,/\,k$,
k restant le PGCD de n – m et r + m.

Si de plus, (n – m) est pair et n et r ont une parité différente,
alors la courbe est stable par symétrie par rapport à une droite,
et donc par rapport à k droites, si k est le PGCD de n – m et r + m .

Ce résumé doit être considéré comme un proposition de classification de courbes de la famille à partir de leurs paramètres. S'efforcer d'abord de faire des preuves cas par cas, comme plus-haut. Mais s'efforcer aussi, guidé par ce résumé, d'étudier des courbes dont les paramètres relèvent de configurations variées. La démonstration abstraite , seulement esquissée ici en dépit de ce résumé,, pourra alors être menée avec succés.. .

■ Aide pour la recherche de courbes

Une courbe de la famille ci-dessus est caractériée par 3 entiers et 2 réels. Voici une fonction *Mathematica* gg qui donne directement les courbes en fonction de ces cinq données et les étiquette.

```
Clear[montre]; montre[g_] := Show[g,
  DisplayFunction → $DisplayFunction, ImageSize → 432]

gg[{m_, n_, p_}, {a_, b_}] := Module[{xx, yy},
  xx[tt_] := Cos[m * tt] + a * Cos[n * tt] + b * Sin[p * tt];
  yy[tt_] := Sin[m * tt] + a * Sin[n * tt] + b * Cos[p * tt];
ParametricPlot[{xx[tt], yy[tt]},
   {tt, -Pi, Pi}, PlotLabel → {{m, n, p}, {a, b}},
   DisplayFunction → $DisplayFunction]]
```

```
ggc[{m_, n_, p_}, {a_, b_}] := Module[{xx, yy},
   xx[tt_] := Cos[m * tt] + a * Cos[n * tt] + b * Sin[p * tt];
   yy[tt_] := Sin[m * tt] + a * Sin[n * tt] + b * Cos[p * tt];
ParametricPlot[{xx[tt], yy[tt]},
    {tt, -Pi, Pi}, PlotLabel → {{m, n, p}, {a, b}},
    DisplayFunction → Identity]];
```

Le module ggc fait le même travail que gg mais il n'affiche pas. C'est pour éviter que *Mathematica* n'affiche tous les graphiques composant un graphique complexe. Pour voir le graphique complexe, utiliser **montre** et non pas **Show.**

■ Récréation

Dans la première partie, nous avons étudié les isométries laissant invariante une courbe. Ces transformations munies de la loi de composition forment un groupe. Si la notion de groupe vous est inconnue, préparez vous à l'apprendre en détaillant avec soin toutes les isométries laissant globalement invariante une courbe et composez ces applications.

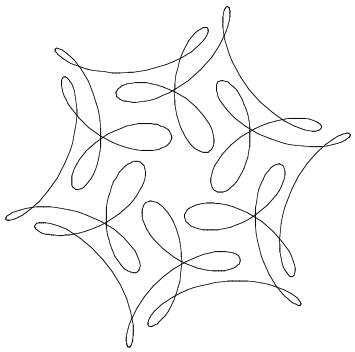
Cet outil dit des groupes, reliant le groupe des isométries d'un graphe et des propriétés arithmétiques des paramètres entiers, pour essentiel qu'il soit, n'est pas la seule information relative à une courbe.

Nous encourageons le lecteur à essayer d'en dégager d'autres .
En effet, le hasard peut fournir des courbes, mais il est plus stimulant pour un graphiste, un dessinateur, un travailleur en modélisation de dresser une liste de "qualificatifs" issus de l'observation des courbes rencontrées puis d'essayer de créer des courbes avec telle ou telle de ces qualités en anticipant un rapport entre cette propriété et des propriétés arithmétiques des paramètres. Ces aller retours entre graphes, mots, équations demandent d' observer, de corréler, de distinguer, bref, d'anticiper. Et en plus, c'est souvent beau et surprenant.

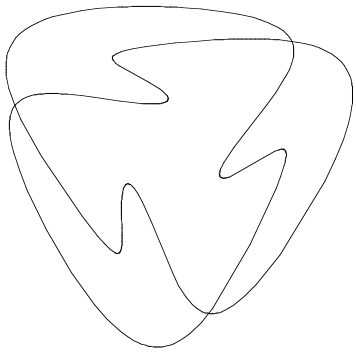
■ Distinguer deux courbes l'une de l'autre avec des mots

```
montre[GraphicsArray[{
    ggc[{1, 7, 17}, {1 / 2, 1 / 3}],
    ggc[{2, 5, 7}, {1 / 2, 1 / 5}]}]];
```

$\{\{1, 7, 17\}, \{\frac{1}{2}, \frac{1}{3}\}\}$



$\{\{2, 5, 7\}, \{\frac{1}{2}, \frac{1}{5}\}\}$



Voici nos deux courbes: Cg et Cd .
La première difficulté consiste à composer avec l'emprise du groupe. Nous proposons, pour affiner les jugements personnels de se baser sur des estimations intuitives comme :

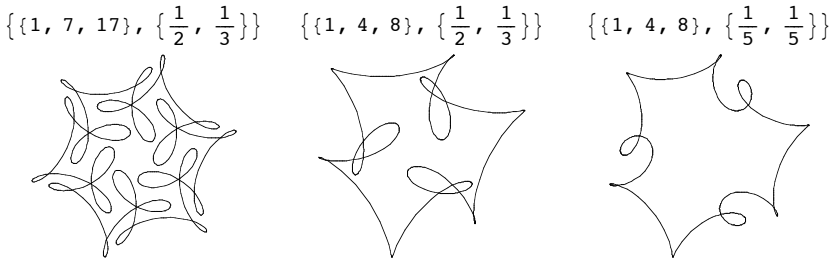
telle courbe C est plus proche de Cg que de Cd ou bien,
telle courbe C serait un bon intermédiaire entre Cg et Cd.

■ Exercice

Trouver quelques courbes de la famille qui constituent comme un chemin, permettant de passer de Cg à Cd de façon moins brutale. Evaluer les tentatives. Retenir les mots employés pour justifier ces jugements. Quels sont ceux que les mathématiques pourraient prendre en compte?

- **méthode 1: trouver une courbe ressemblant à celle de gauche avec le groupe de droite**

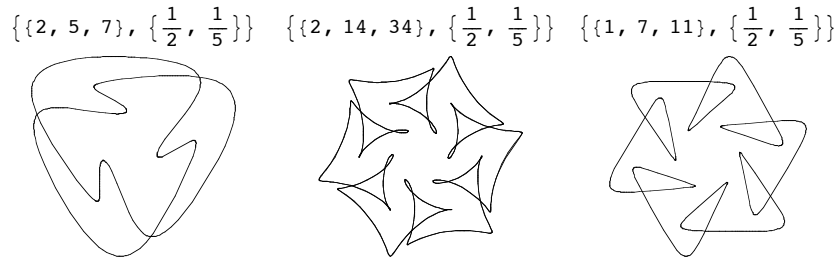
```
montre[GraphicsArray[{
    ggc[{1, 7, 17}, {1/2, 1/3}],
    ggc[{1, 4, 8}, {1/2, 1/3}],
    ggc[{1, 4, 8}, {1/5, 1/5}]}]];
```



Bien pour le groupe, mais ressemblance décevante des courbes Absence **de points triples** par exemple. **Contour très anguleux** au milieu. Faute de posséder une loi pour trouver des points triples, on tatonne. Toutefois, observons que ces notions dépendent aussi des 2 derniers paramètres, contrairement au groupe qui ne dépendait que des trois paramètres entiers,

- **méthode 2: trouver un alias de la courbe de gauche avec des rotations d'angle π / 3 (pas facile)**

```
montre[GraphicsArray[{
     ggc[{2, 5, 7}, {1/2, 1/5}],
    ggc[{2, 14, 34}, {1/2, 1/5}],
    ggc[{1, 7, 11}, {1/2, 1/5}]}]];
```



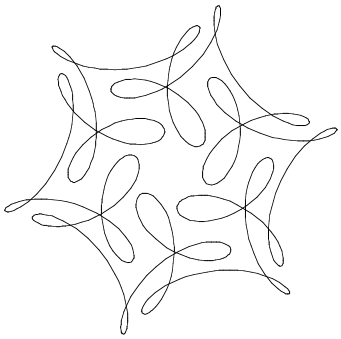
Ressemblance décevante.. A droite le point courant fait **un tour** autour du centre et **six pirouettes**. A gauche, il fait **deux tours** autour de l'origine et son contour est plus lisse .

Bref, pas immédiat, mais, indéniablement bon pour le calcul mental et pour apprivoiser les groupes.

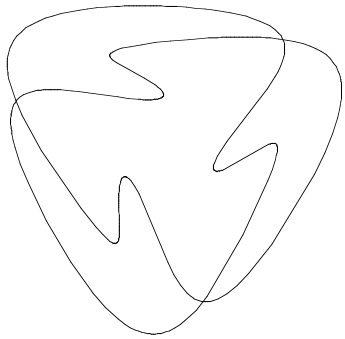
- **méthode 3: pour chercher à comparer les deux courbes du début qui ont des groupes très différents (3 ou 6 rotations) , il vaut mieux prendre un entier strictement entre 3 et 6, par exemple 5 et trouver deux courbes voisines des courbes initiales et stables par rotation d'angle $2\pi/5$.**

```
montre[GraphicsArray[
   {{ggc[{1, 7, 17}, {1 / 2, 1 / 3}], ggc[{2, 5, 7},
      {1 / 2, 1 / 5}]}, {ggc[{1, 6, 14}, {1 / 2, 1 / 3}],
     ggc[{2, 7, 13}, {1 / 3, 1 / 9}]}}]];
```

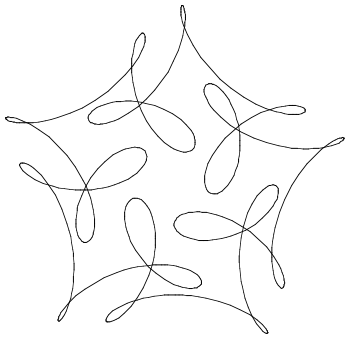
$\{\{1, 7, 17\}, \{\frac{1}{2}, \frac{1}{3}\}\}$



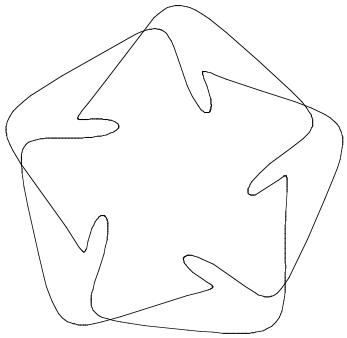
$\{\{2, 5, 7\}, \{\frac{1}{2}, \frac{1}{5}\}\}$



$\{\{1, 6, 14\}, \{\frac{1}{2}, \frac{1}{3}\}\}$



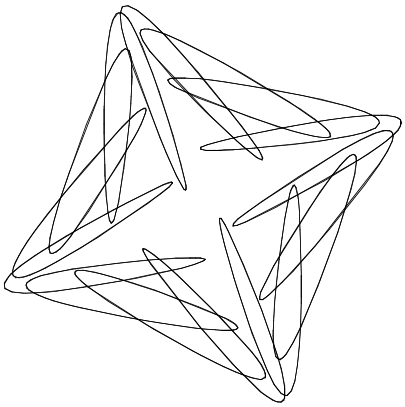
$\{\{2, 7, 13\}, \{\frac{1}{3}, \frac{1}{9}\}\}$



■ Et si l'on se trompe, c'est beau quand même, parfois

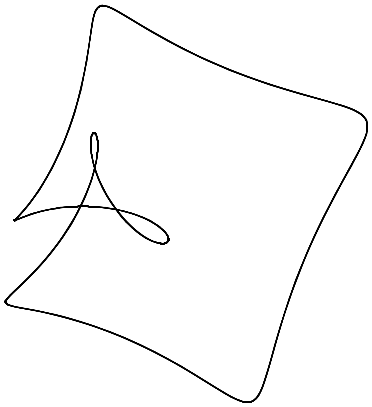
```
Clear[x]; x[t_] := Cos[t] + Cos[15 t] / 2 + Sin[17 t] / 3;
Clear[y]; y[t_] := Sin[t] + Sin[15 t] / 2 + Cos[17 t] / 3;
g15 = ParametricPlot[{x[t], y[t]}, {t, -2 Pi, 2 Pi},
   Axes → False, AspectRatio → Automatic];

montre[g15];
```



```
x[t_] := Cos[12 t] + Cos[24 t] + Sin[60 t] / 3
y[t_] := Sin[12 t] + Sin[24 t] + Cos[60 t] / 3
g16 = ParametricPlot[
   {x[t], y[t]}, {t, -Pi, Pi}, Axes → False,
   AspectRatio → Automatic, PlotPoints → 1000];

montre[g16];
```



Contenu