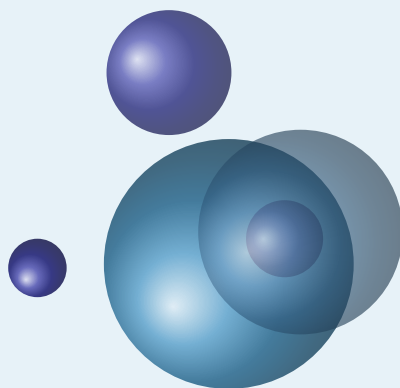
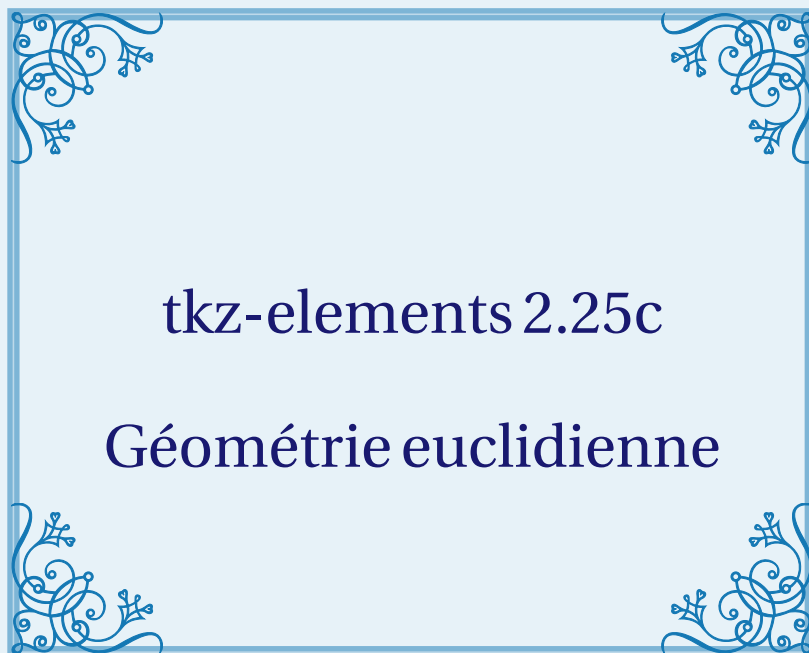


AlterMundus



Alain Matthes

28 avril 2024 Documentation V.2.25c

<http://altermundus.fr>

tkz-elements

Alain Matthes

☞ Ce document compile quelques notes sur **tkz-elements**, la version initiale d'une bibliothèque Lua conçue pour effectuer tous les calculs nécessaires à la définition d'objets dans des figures de géométrie euclidienne. Votre document doit être compilé en utilisant Lua \LaTeX .

Avec tkz-elements, les définitions et les calculs sont exclusivement réalisés en utilisant Lua.

L'approche de programmation principale offerte est orientée vers la programmation orientée objet, en utilisant des classes d'objets telles que point, ligne, triangle, cercle et ellipse. Actuellement, après que les calculs sont terminés, tkz-euclide ou TikZ est utilisé à des fins de dessin.

J'ai découvert Lua et la programmation orientée objet en développant ce package, il est donc très probable que j'aie commis quelques erreurs. Si vous souhaitez contribuer au développement de ce package ou fournir des conseils sur la marche à suivre, veuillez me contacter par e-mail.

AlterMundus

☞ Remerciements : J'ai reçu de précieux conseils, remarques et corrections de la part de Nicolas Kisselhoff, David Carlisle, Roberto Giacomelli et Qrrrbirlbel.

Un grand merci à Wolfgang Büchel pour sa contribution inestimable à la correction des exemples.

☞ J'aimerais également exprimer ma gratitude à Eric Weisstein, créateur de [MathWorld](#).

☞ Vous pouvez trouver quelques exemples sur mon site : [altermundus.fr](#).

En construction!

Veuillez signaler les fautes de frappe ou tout autre commentaire concernant cette documentation à : [Alain Matthes](#).
Ce fichier peut être redistribué et/ou modifié selon les termes de la \LaTeX Project Public License Distributed from [CTAN](#) archives.

Table des matières

1	Structure	9
2	Pourquoi tkz-elements?	10
2.1	Précision des calculs	10
2.1.1	Précision des calculs en TikZ	10
2.1.2	Précision des calculs en Lua	10
2.1.3	Utilisation d'objets	10
2.1.4	Exemple : Cercle d'Apollonius	10
3	Présentation	14
3.1	Avec Lua	14
3.2	Le processus principal	14
3.3	Exemple complet : Cercle de pappus	15
3.3.1	La figure	15
3.3.2	Le code	15
3.4	Un autre exemple : Pôle Sud	16
4	Convention d'écriture	18
4.1	Divers	18
4.2	Attribution d'un nom à un point	18
4.3	Attribution d'un nom à d'autres objets	19
4.4	Conventions d'écriture pour les attributs et les méthodes.	19
5	Organisation du travail	19
5.1	Problème d'échelle	21
5.2	Présentation du code	21
6	Transferts	22
6.1	De Lua à tkz-euclide ou TikZ	22
6.1.1	Transfert de points	22
6.1.2	Autres transferts	23
7	Classe et objet	24
7.1	Classe	24
7.2	Objet	24
7.2.1	Attributs	24
7.2.2	Méthodes	24
8	Classe point	25
8.1	Attributs d'un point	25
8.1.1	Exemple : attributs de points	26
8.1.2	Diagramme d'Argand	27
8.2	Méthodes de la classe point	28
8.2.1	Exemple : méthode north (d)	28
8.2.2	Transfert de longueur	28
8.2.3	Exemple : method polar	29
8.2.4	Méthode normalize ()	29
8.2.5	Orthogonal (d) méthode	30
8.2.6	at méthode	30
8.2.7	Exemple : rotation of points	30
8.2.8	Objet rotation	31
8.2.9	Objet symmetry	31

9	Classe line (droite)	32
9.1	Attributs d'une droite (line)	32
9.1.1	Exemple : attributs de la classe line	33
9.1.2	Méthode new	33
9.2	Méthodes de la classe line	34
9.2.1	Méthode report	35
9.2.2	Triangle avec two_angles	35
9.2.3	Triangle à trois côtés donnés	36
9.2.4	Triangle dont le côté est compris entre le côté et l'angle	36
9.2.5	À propos des triangles sacrés	36
9.2.6	Méthode point point	37
9.2.7	Méthode colinear_at	38
9.2.8	Méthode normalize	38
9.2.9	Méthode barycenter	39
9.2.10	Méthode ll_from	39
9.2.11	Méthode projection	39
9.2.12	Exemple : combination of methods	40
9.2.13	Méthode translation	40
9.2.14	Méthode distance	41
9.2.15	Méthode reflection	41
9.3	Méthode apollonius Apollonius circle $MA/MB = k$	41
10	Classe circle	43
10.1	Attributs d'un cercle	43
10.1.1	Exemple : attributs du cercle	43
10.2	Méthodes de la classe circle	44
10.2.1	Altshiller	45
10.2.2	Lemoine	45
10.2.3	Inversion : point, droite et cercle	46
10.2.4	Inversion : point	46
10.2.5	Inversion : droite	46
10.2.6	Inversion : cercle	46
10.2.7	Midcircle	47
10.3	Méthode Circles_position	50
10.4	Tangente commune : méthode tangent_common	50
10.5	Tangente commune : orthogonalité	51
10.6	Méthode in_out for circle and in_out_disk for disk	52
11	Classe triangle	54
11.1	Attributs d'un triangle	54
11.2	Attributs du triangle : angles	54
11.2.1	Exemple : attributs du triangle	55
11.3	Methods of the class triangle	56
11.3.1	Méthodes cevian et cevian_circle	57
11.3.2	Méthodes pedal et pedal_circle	58
11.3.3	Méthodes conway_points et conway_circle	58
11.3.4	Euler line	59
11.4	Ellipse d'Euler	59
11.4.1	Inellipse et circumellipse de Steiner	60
11.5	Division harmonique et bissectrice	61

12	Classe ellipse	62
12.1	Attributs d'une ellipse	62
12.1.1	Attributs d'une ellipse : exemple	62
12.2	Méthodes de la classe ellipse	63
12.2.1	Méthode new	63
12.2.2	Méthode foci	64
12.2.3	Méthode point and radii	64
13	Classe Quadrilateral	66
13.1	Attributs du quadrilatère	66
13.1.1	Attributs du quadrilatère	66
13.2	Quadrilateral methods	66
13.2.1	Quadrilatère inscrit	67
14	Classe square	68
14.1	Attributs du carré	68
14.1.1	Exemple : attributs du carré	68
14.2	Méthodes du carré	69
14.2.1	Carré avec la méthode side	69
15	Classe rectangle	70
15.1	Attributs du rectangle	70
15.1.1	Exemple	70
15.2	Méthodes relatives aux rectangles	71
15.2.1	Méthode angle	71
15.2.2	Méthode side	71
15.2.3	Méthode diagonal	72
15.2.4	Méthode gold	72
16	Classe parallelogram	73
16.1	Attributs du parallélogramme	73
16.1.1	Exemple : attributs	73
16.2	Méthodes du parallélogramme	74
16.2.1	Méthode fourth	74
17	Classe regular_polygon	75
17.1	attributs du polygone régulier	75
17.1.1	Pentagone	75
17.2	Méthodes de polygone régulier	75
18	Classe vector	76
18.1	Attributs d'un vecteur	76
18.1.1	Exemple d'attributs de la classe vector	77
18.2	Méthodes de la classe vector	77
18.2.1	Exemple de méthodes	78
19	Classe matrix	79
19.1	Création de la matrice	79
19.2	Afficher une matrice : méthode print	79
19.3	Attributs d'une matrice	79
19.3.1	Attribut set	80
19.3.2	Déterminant avec des nombres réels : det	80
19.3.3	Déterminant avec les nombres complexes	80

19.4	Métaméthodes pour les matrices	80
19.4.1	Addition and subtraction of matrices	81
19.4.2	Multiplication et puissance des matrices	81
19.4.3	Métaméthode eq	81
19.5	Méthodes de la classe matrix	81
19.5.1	Fonction new	82
19.5.2	Fonction vector	82
19.5.3	Méthode homogenization	82
19.5.4	Function htm : matrice de transformation homogène	83
19.5.5	Méthode get_htm_point	83
19.5.6	Méthode htm_apply	83
19.5.7	Fonction square	84
19.5.8	Méthode print	84
19.5.9	Afficher un tableau : fonction print_array	85
19.5.10	Obtenir un élément d'une matrice : méthode get	85
19.5.11	Matrice inverse : méthode inverse	85
19.5.12	Matrice inverse avec syntaxe de puissance	86
19.5.13	Transposition de la matrice : méthode transpose	86
19.5.14	Méthode adjugate	86
19.5.15	Méthode identity	86
19.5.16	Diagonalisation : méthode diagonalize	87
19.5.17	Méthode is_orthogonal	87
19.5.18	Méthode is_diagonal	87
20	Constantes et fonctions mathématiques	88
20.1	Longueur d'un segment	88
20.2	Division harmonique avec tkzphi	88
20.3	Fonction islinear	89
20.4	Fonction value	89
20.5	Function real	89
20.6	Transfer from lua to T _E X	89
20.7	Angles normalisés : Pente des droites (ab), (ac) and (ad)	89
20.8	Obtenir un angle	90
20.9	Produit de point ou produit scalaire	91
20.10	Alignement ou orthogonalité	91
20.11	Bissectrice et hauteur	91
20.12	Autres fonctions	92
	20.12.1 Fonction solve_quadratic	92
21	Intersections	93
21.1	Droite-droite	93
21.2	Droite-cercle	94
21.3	Cercle-cercle	95
21.4	Droite-ellipse	96
22	Étude approfondie	97
22.1	Les tables	97
	22.1.1 General tables	97
	22.1.2 Table z	98
22.2	Transferts	98
22.3	Bibliothèque des nombres complexes et point	99
	22.3.1 Exemple d'utilisation complexe	100
	22.3.2 Opérations avec les points (complexes)	100

22.4	Barycentre	101
22.4.1	Utilisation du barycentre	101
22.4.2	Centre du cercle inscrit d'un triangle	101
22.5	Notation des boucles et des tableaux	101
22.6	Méthode in_out	102
22.6.1	In_out pour une droite	102
22.7	Déterminant et produit scalaire	103
22.7.1	Déterminant	103
22.7.2	Produit scalaire	103
22.7.3	produit scalaire : test d'orthogonalité	104
22.7.4	produit scalaire : projection	104
22.8	Méthode point	104
22.9	Derrière les objets	105
23	Exemples	106
23.1	D'Alembert 1	106
23.2	D'Alembert 2	106
23.3	Alternance	107
23.4	Cercle d'Apollonius	107
23.5	Apollonius et le cercle circonscrit	108
23.6	Cercles d'Apollonius dans un triangle	109
23.7	Archimède	111
23.8	Le cercle de Bankoff	111
23.9	Cercles exinscrits	113
23.10	Orthogonal_through	114
23.11	Rapport divin	114
23.12	Cercle directeur	116
23.13	Division or	116
23.14	Ellipse	117
23.15	Ellipse avec rayons	118
23.16	Ellipse with foci	118
23.17	Relation d'Euler	119
23.18	Angle externe	120
23.19	Angle interne	120
23.20	Théorème de Feuerbach	121
23.21	Ratio d'or avec le segment	122
23.22	Gold Arbelos	122
23.23	Division harmonique v1	123
23.24	Division harmonique v2	123
23.25	Menelaus	124
23.26	Axe radical v1	124
23.27	Axe radical v2	125
23.28	Axe radical v3	126
23.29	Axe radical v4	127
23.30	Centre radical	128
23.31	Cercle radical	129
23.32	Ellipse d'Euler	130
23.33	Gold Arbelos propriétés	131
23.34	Cercle d'Apollonius v1 avec inversion	132
23.35	Cercle d'Apollonius v2	133
23.36	Cercles orthogonaux v1	134
23.37	Cercles orthogonaux v2	135
23.38	Cercle orthogonal à deux cercles	136

23.39	Midcircles	138
23.40	Faisceau : v1	139
23.41	Faisceau v2	140
23.42	Puissance v1	141
23.43	Puissance v2	142
23.44	Reim v1	142
23.45	Reim v2	143
23.46	Reim v3	144
23.47	Tangente et cercle	145
23.48	Homothétie	146
23.49	Tangente et corde	146
23.50	Trois cordes	146
23.51	Trois tangentes	148
23.52	Midarc	148
23.53	Droite de Lemoine sans macro	149
23.54	Premier cercle de Lemoine	149
23.55	Premier et deuxième cercle de Lemoine	150
23.56	Inversion	151
23.57	Point de Gergonne	152
23.58	Antiparallèle au point de Lemoine	152
23.59	Le cercle de Soddy sans fonction	152
23.60	Cercle de Soddy avec fonction	154
23.61	Chaîne de Pappus	155
23.62	Trois cercles	156
23.63	Pentagones dans un “golden arbelos”	157
24	Aide-mémoire	164

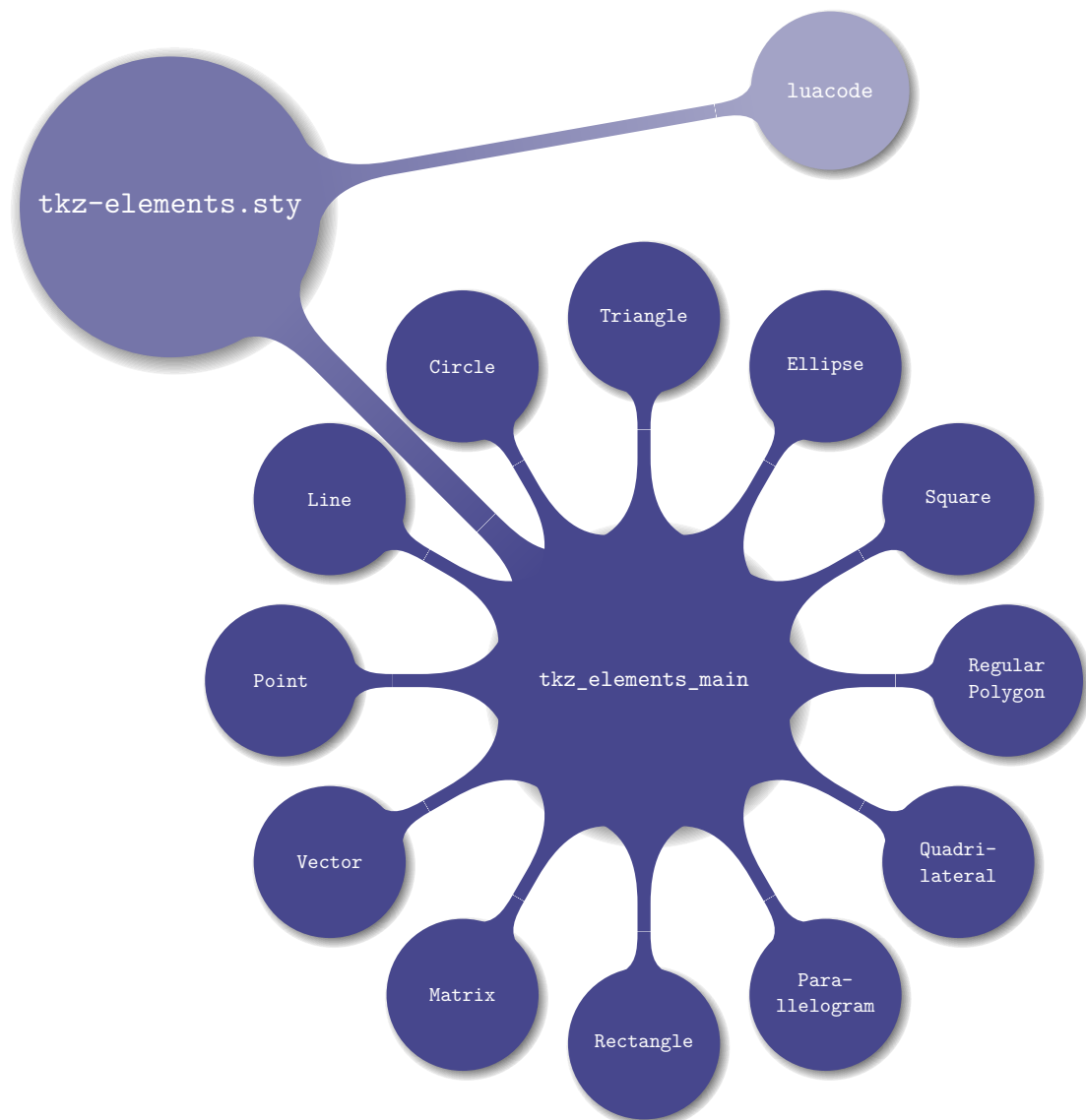
1 Structure

`tkz-elements.sty` charge le package `luacode` pour créer l'environnement `tkzelements`, qui est basé sur l'environnement `luacode`.

À l'intérieur de l'environnement `tkzelements`, l'échelle est initialisée à 1, puis toutes les valeurs dans diverses tables sont effacées.

Le package définit deux macros `\tkzGetNodes` et `\tkzUseLua`.

De plus, le package charge le fichier `tkz_elements_main.lua`. Ce fichier initialise toutes les tables qui seront utilisées par les modules dans lesquels les classes sont définies.



Les classes actuellement disponibles sont (certaines sont encore inactives) :

- actives : `point (z)`; `line (L)`; `circle (C)`; `triangle (T)`; `ellipse (E)`; `quadrilateral (Q)`; `square (S)`; `rectangle (R)`; `parallelogram (P)`; `regular_polygon (RP)`; `matrix (M)`; `vector (V)`.

Si `name` est le nom d'une classe, vous pouvez trouver sa définition dans le fichier `tkz_elements_name.lua`.



2 Pourquoi tkz-elements?

2.1 Précision des calculs

2.1.1 Précision des calculs en TikZ

Avec TikZ, l'expression `veclen(x,y)` calcule l'expression $\sqrt{x^2 + y^2}$. Ce calcul est réalisé grâce à une approximation polynomiale, s'inspirant des idées de Rouben Rostamian.

```
pgfmathparse{veclen(65,72)} \pgfmathresult
```

 $\sqrt{65^2 + 72^2} \approx 96.9884$ 

2.1.2 Précision des calculs en Lua


A `luaveclen` macro can be defined as follows :

```
\def\luaveclen#1#2{\directlua{tex.print(string.format(
'\percentchar.5f',math.sqrt((#1)*(#1)+(#2)*(#2))))}}
```

et

```
\luaveclen{65}{72}
```

donne

 $\sqrt{65^2 + 72^2} = 97$!!

L'erreur, bien qu'insignifiante lorsqu'il s'agit de placer un objet sur une page à un centième de point près, devient problématique pour les résultats des démonstrations mathématiques. De plus, ces imprécisions peuvent s'accumuler et conduire à des constructions erronées.

Pour remédier à ce manque de précision, j'ai initialement introduit le package `fp`, suivi du package `xfp`. Plus récemment, avec l'émergence de `LuaTeX`, j'ai incorporé une option `Lua` visant à effectuer des calculs avec Lua. C'était la motivation principale derrière la création du package, avec comme objectif secondaire l'introduction de la programmation orientée objet (OOP) et la simplification de la programmation avec Lua. Le concept de OOP m'a convaincu d'explorer davantage ses différentes possibilités.

À cette époque, j'avais reçu quelques exemples de programmation Lua de [Nicolas Kisselhoff](#), mais j'ai eu du mal à comprendre le code au début, donc j'ai consacré du temps à étudier Lua patiemment. Finalement, j'ai pu développer `tkz-elements`, en incorporant bon nombre de ses idées que j'ai adaptées pour le package.

2.1.3 Utilisation d'objets

Par la suite, j'ai découvert un article de [Roberto Giacomelli](#)¹ sur la programmation orientée objet utilisant Lua et les outils TikZ. Cela a servi de deuxième source d'inspiration. Non seulement cette approche permettait d'exécuter la programmation pas à pas, mais l'introduction d'objets facilitait également un lien direct entre le code et la géométrie. En conséquence, le code est devenu plus lisible, explicite et mieux structuré.

2.1.4 Exemple: Cercle d'Apollonius

Problème : L'objectif est d'identifier un cercle tangent intérieur aux trois cercles exinscrits d'un triangle.

Pour plus de détails, voir [MathWorld](#)

1. [Grafica ad oggetti con LuaTeX](#)

Cet exemple a servi de référence pour tester le package `tkz-euclide`. Initialement, avec mes premières méthodes et les outils disponibles, les résultats manquaient de précision. Cependant, avec `tkz-elements`, j'ai désormais accès à des outils plus puissants et précis, qui sont également plus faciles à utiliser.

Les principes fondamentaux de la construction de figures avec `tkz-euclide` restent intacts : les définitions, les calculs, les tracés, les étiquettes, ainsi que la programmation pas à pas, reflétant le processus de construction avec une règle et un compas.

Cette version utilise la méthode de construction la plus simple rendue possible par Lua.

```
\begin{tkzelements}
  scale          = .4
  z.A            = point: new (0,0)
  z.B            = point: new (6,0)
  z.C            = point: new (0.8,4)
  T.ABC          = triangle : new ( z.A,z.B,z.C )
  z.N            = T.ABC.eulercenter
  z.S            = T.ABC.spiekercenter
  T.feuerbach    = T.ABC : feuerbach ()
  z.Ea,z.Eb,z.Ec = get_points ( T.feuerbach )
  T.excentral    = T.ABC : excentral ()
  z.Ja,z.Jb,z.Jc = get_points ( T.excentral )
  C.JaEa         = circle: new (z.Ja,z.Ea)
  C.ortho        = circle: radius (z.S,math.sqrt(C.JaEa: power(z.S)))
  z.a            = C.ortho.through
  C.euler        = T.ABC: euler_circle ()
  C.apo          = C.ortho : inversion (C.euler)
  z.O            = C.apo.center
  z.xa,z.xb,z.xc = C.ortho : inversion (z.Ea,z.Eb,z.Ec)
\end{tkzelements}
```

La création d'un objet encapsule ses attributs (ses caractéristiques) et méthodes (c'est-à-dire les actions qui lui sont spécifiques). Ensuite, il lui est attribué une référence (un nom) qui est liée à l'objet à l'aide d'une table. Cette table fonctionne comme un tableau associatif qui relie la référence, appelée une clé, à une valeur, dans ce cas, l'objet. Une élaboration supplémentaire sur ces notions sera fournie ultérieurement.

Par exemple, soit `T` une table associant l'objet `triangle` à la clé `ABC`. `T.ABC` est également une table, et ses éléments sont accessibles en utilisant des clés qui correspondent aux attributs du triangle. Ces attributs ont été définis dans le package.

```
z.N = T.ABC.eulercenter
```

`N` est le nom du point, `eulercenter` est un attribut du triangle.²

```
T.excentral = T.ABC : excentral ()
```

Dans ce contexte, `excentral` est une méthode associée à l'objet `T.ABC`. Elle définit le triangle formé par les centres des cercles exinscrits.

Deux lignes de code sont particulièrement importantes. La première ci-dessous démontre que la précision exceptionnelle fournie par Lua permet de définir un rayon grâce à un calcul complexe. Le rayon du cercle radical est déterminé par $\sqrt{\Pi(S, \mathcal{C}(Ja, Ea))}$ (racine carrée de la puissance du point S par rapport au cercle exinscrit avec le centre Ja passant par Ea).

2. Le centre du cercle d'Euler, ou centre du cercle des neuf points, est une caractéristique de chaque triangle.

```
C.ortho = circle: radius (z.S,math.sqrt(C.JaEa: power(z.S)))
```

Enfin, il convient de noter que l'inversion du cercle d'Euler par rapport au cercle radical donne le cercle d'Apollonius³. Cette transformation nécessite un objet en tant que paramètre, qui est reconnu par son type (tous les objets sont typés dans le package), et la méthode détermine quel algorithme utiliser en fonction de ce type.

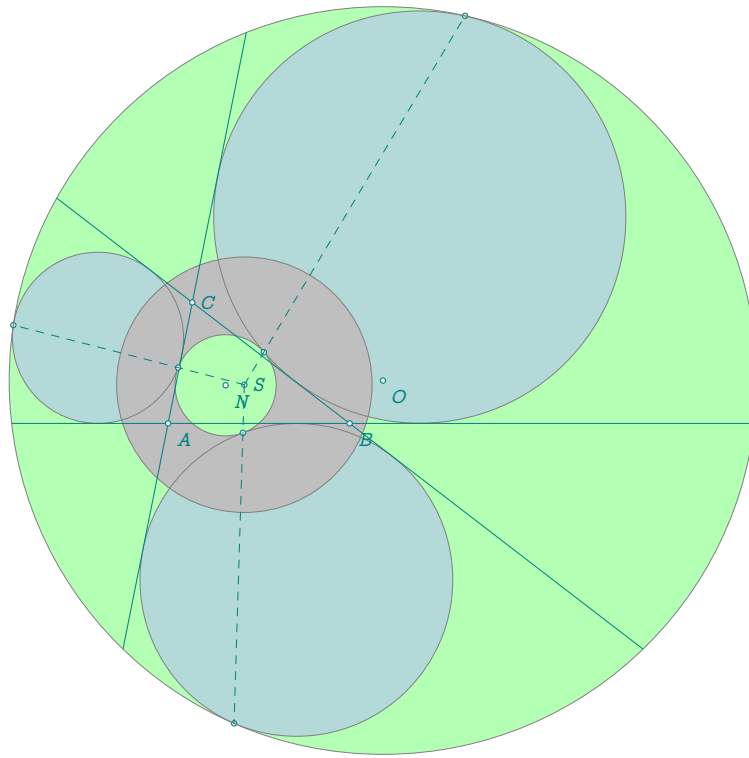
```
C.apo = C.ortho : inversion (C.euler)
```

Maintenant que tous les points ont été définis, il est temps de commencer à tracer les chemins. Pour ce faire, il faut créer des nœuds. C'est le rôle de la macro `.Se` référer à 6.1.1

La section suivante traite exclusivement des tracés et est gérée par `tkz-euclide`.

```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzFillCircles[green!30](O,xa)
  \tkzFillCircles[teal!30](Ja,Ea Jb,Eb Jc,Ec)
  \tkzFillCircles[lightgray](S,a)
  \tkzFillCircles[green!30](N,Ea)
  \tkzDrawPoints(xa,xb,xc)
  \tkzDrawCircles(Ja,Ea Jb,Eb Jc,Ec S,a O,xa N,Ea)
  \tkzClipCircle(O,xa)
  \tkzDrawLines[add=3 and 3](A,B A,C B,C)
  \tkzDrawPoints(O,A,B,C,S,Ea,Eb,Ec,N)
  \tkzDrawSegments[dashed](S,xa S,xb S,xc)
  \tkzLabelPoints(O,N,A,B)
  \tkzLabelPoints[right](S,C)
\end{tikzpicture}
```

3. Le cercle des neuf points, ou cercle d'Euler, est tangent externe aux trois cercles. Les points de tangence forment le triangle de Feuerbach.



3 Présentation

3.1 Avec Lua

La fonction principale de tkz-elements est de calculer les dimensions et de définir les points, en utilisant Lua. Vous pouvez considérer tkz-elements comme un noyau utilisé soit par tkz-euclide, soit par TikZ. Les définitions et les calculs ont lieu dans l'environnement **tkzelements**, qui est basé sur **luacode**.

Les points clés sont :

- Le fichier source doit être encodé en **UTF8**.
- La compilation est effectuée avec **Lua^{LaTeX}**.
- Vous devez charger TikZ ou tkz-euclide et tkz-elements.
- Les définitions et calculs sont effectués dans un système de coordonnées cartésien (orthonormal), en utilisant Lua à l'intérieur de l'environnement tkzelements.

Sur la droite, vous pouvez voir le modèle minimum.

Le code est divisé en deux parties, représentées par deux environnements **tkzelements** et **tikzpicture**. Dans le premier environnement, vous placez votre code Lua, tandis que dans le second, vous utilisez les commandes tkz-euclide.

```
% !TEX TS-program = lualatex
% Created by Alain Matthes
\documentclass{standalone}
\usepackage{tkz-euclide}
% or simply TikZ
\usepackage{tkz-elements}
begin{document}

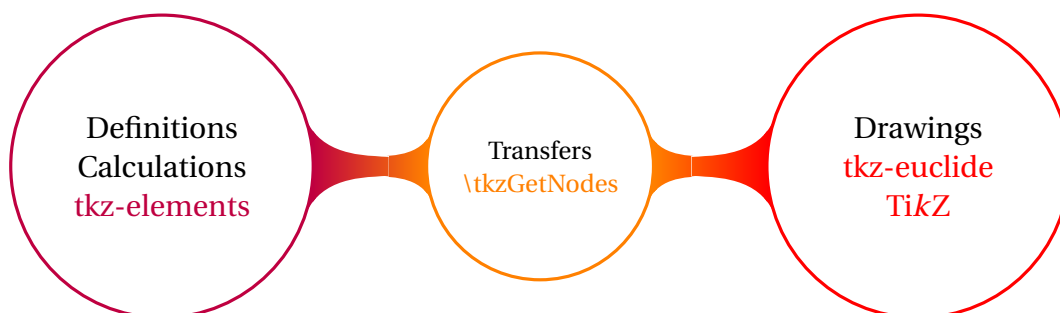
\begin{tkzelements}
  scale = 1
  % definition of some points
  z.A = point : new ( , )
  z.B = point : new ( , )

  ...code...
\end{tkzelements}

\begin{tikzpicture}
% point transfer to Nodes
\tkzGetNodes

\end{tikzpicture}
\end{document}
```

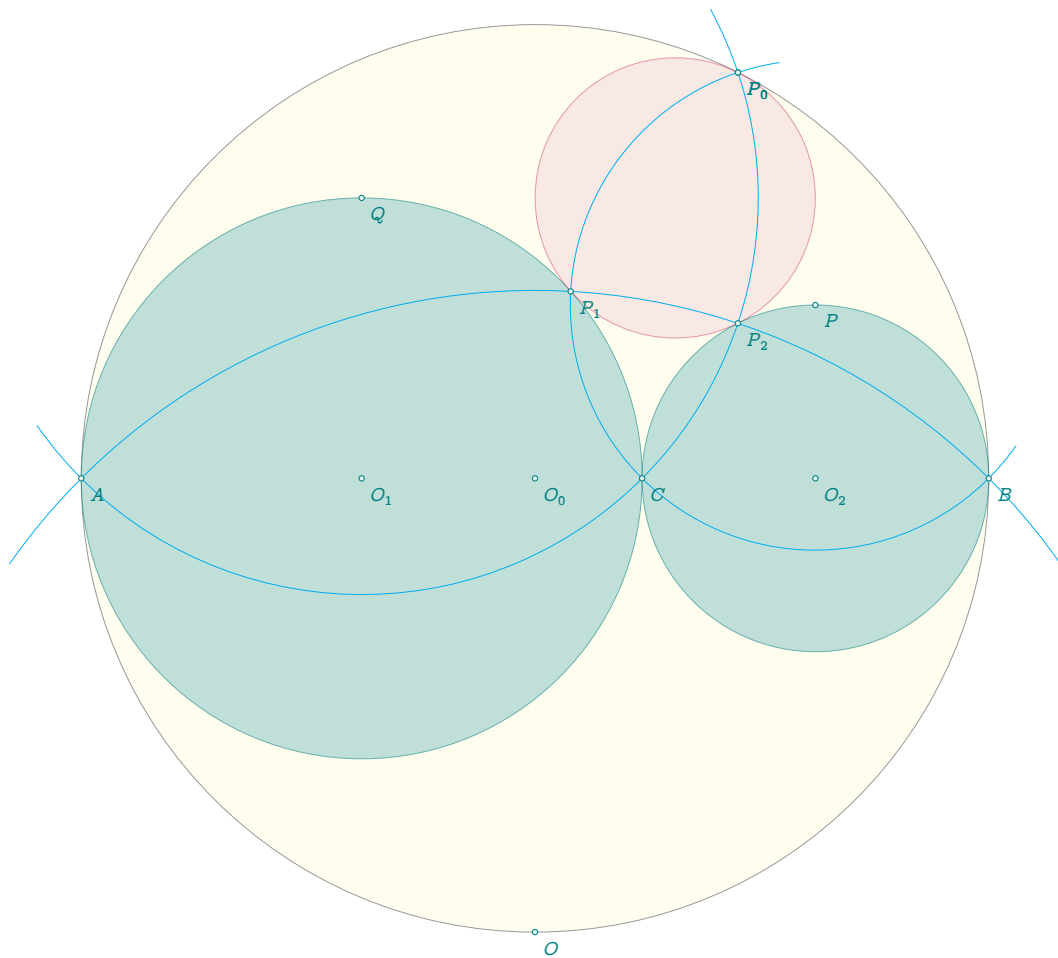
3.2 Le processus principal



Après avoir obtenu tous les points nécessaires pour le dessin, ils doivent être transformés en **nodes** afin que TikZ ou tkz-euclide puisse rendre la figure. Cela est accompli en utilisant la macro **\tkzGetNodes**. Cette macro itère à travers tous les éléments de la table **z** en utilisant la clé (qui est essentiellement le nom du point) et récupère les valeurs associées, à savoir les coordonnées du point (node).

3.3 Exemple complet : Cercle de pappus

3.3.1 La figure



3.3.2 Le code

```
% !TEX TS-program = lualatex
\documentclass{article}
\usepackage{tkz-euclide}
\usepackage{tkz-elements}
\begin{document}

\begin{tkzelements}
z.A      = point: new (0 , 0)
z.B      = point: new (10 , 0)      -- creation of two fixed points $A$ and $B$
L.AB     = line:  new ( z.A, z.B)
z.C      = L.AB:  gold_ratio ()     -- use of a method linked to "line"
z.O_0    = line:  new ( z.A, z.B).mid -- midpoint of segment with an attribute of "line"
z.O_1    = line:  new ( z.A, z.C).mid -- objects are not stored and cannot be reused.
z.O_2    = line:  new ( z.C, z.B).mid
C.AB     = circle: new ( z.O_0, z.B) -- new object "circle" stored and reused
C.AC     = circle: new ( z.O_1, z.C)
C.CB     = circle: new ( z.O_2, z.B)
z.P      = C.CB.north               -- "north" attributes of a circle
\end{tkzelements}
\end{document}
```

```

z.Q      = C.AC.north
z.O      = C.AB.south
z.c      = z.C : north (2)          -- "north" method of a point (needs a parameter)
C.PC     = circle: new ( z.P, z.C)
C.QA     = circle: new ( z.Q, z.A)
z.P_0    = intersection (C.PC,C.AB) -- search for intersections of two circles.
z.P_1    = intersection (C.PC,C.AC) -- idem
_,z.P_2  = intersection (C.QA,C.CB) -- idem
z.O_3    = triangle: new ( z.P_0, z.P_1, z.P_2).circumcenter
          -- circumcenter attribute of "triangle"
\end{tkzelements}

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircle[black,fill=yellow!20,opacity=.4] (O_0,B)
  \tkzDrawCircles[teal,fill=teal!40,opacity=.6] (O_1,C O_2,B)
  \tkzDrawCircle[purple,fill=purple!20,opacity=.4] (O_3,P_0)
  \tkzDrawArc[cyan,delta=10] (Q,A) (P_0)
  \tkzDrawArc[cyan,delta=10] (P,P_0) (B)
  \tkzDrawArc[cyan,delta=10] (O,B) (A)
  \tkzDrawPoints(A,B,C,O_0,O_1,O_2,P,Q,P_0,P_0,P_1,P_2,O)
  \tkzLabelPoints(A,B,C,O_0,O_1,O_2,P,Q,P_0,P_0,P_1,P_2,O)
\end{tikzpicture}
\end{document}

```

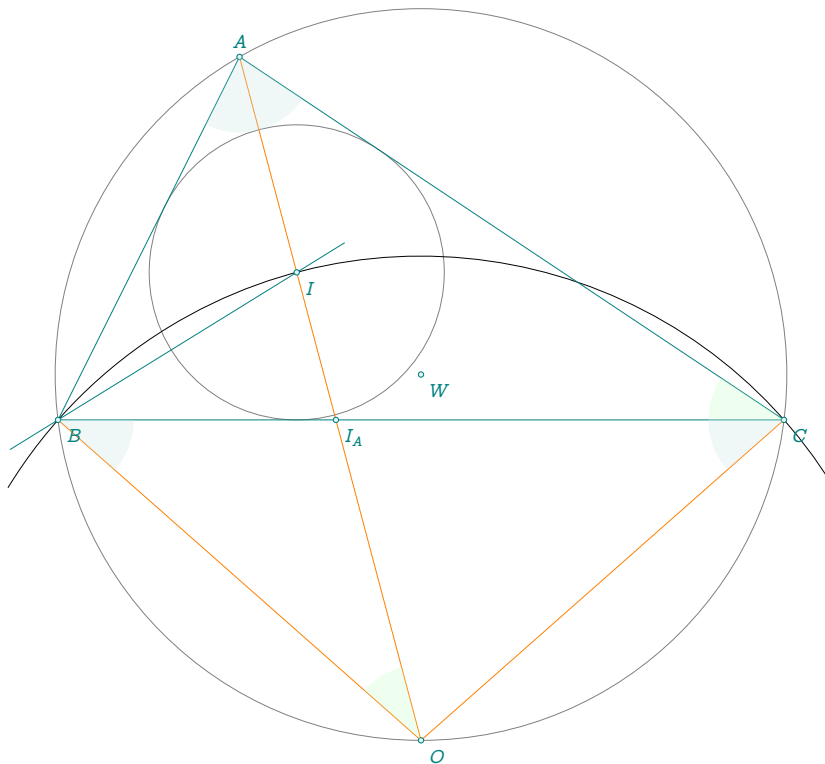
3.4 Un autre exemple : Pôle Sud

Voici un autre exemple avec des commentaires

```

% !TEX TS-program = lualatex
\documentclass{standalone}
\usepackage{tkz-euclide,tkz-elements}
\begin{document}
\begin{tkzelements}
  z.A      = point: new (2 , 4)          -- nous créons un environnement tkzelements
  z.B      = point: new (0 , 0)          -- trois points fixes sont utilisés
  z.C      = point: new (8 , 0)
  T.ABC    = triangle: new (z.A,z.B,z.C) -- nous créons un nouvel objet triangle
  C.ins    = T.ABC: in_circle ()         -- on obtient l'incircle de ce triangle
  z.I      = C.ins.center                -- le centre est un attribut du cercle
  z.T      = C.ins.through                -- à travers est également un attribut
  -- z.I,z.T = get_points (C.ins)        -- get_points est un raccourci
  C.cir    = T.ABC : circum_circle ()    -- nous obtenons le cercle circonscrit
  z.W      = C.cir.center                 -- nous obtenons le centre de ce cercle
  z.O      = C.cir.south                  -- O est le pôle sud de ce cercle
  L.AO     = line: new (z.A,z.O)          -- nous créons un objet "ligne"
  L.BC     = T.ABC.bc                     -- nous obtenons la ligne (BC)
  z.I_A    = intersection (L.AO,L.BC)     -- intersection des dernières lignes
\end{tkzelements}

```

Voici l'environnement tikzpicture pour obtenir la figure :

```

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(W,A I,T)
\tkzDrawArc(O,C) (B)
\tkzDrawPolygon(A,B,C)
\tkzDrawSegments[new] (A,O B,O C,O)
\tkzDrawLine(B,I)
\tkzDrawPoints(A,B,C,I,I_A,W,O)
\tkzFillAngles[green!20,opacity=.3] (A,O,B A,C,B)
\tkzFillAngles[teal!20,opacity=.3] (O,B,C B,C,O B,A,O O,A,C)
\tkzLabelPoints(I,I_A,W,B,C,O)
\tkzLabelPoints[above] (A)
\end{tikzpicture}

```

4 Convention d'écriture

4.1 Divers

- Variable numérique : les conventions d'écriture pour les nombres réels sont les mêmes que pour Lua.
- Nombres complexes : Similaires aux nombres réels, mais pour les définir, vous devez écrire `za = point (1,2)`. Mathématiquement, cela correspond à $1+2i$, que vous pouvez obtenir avec `tex.print(tostring(za))` (Référez-vous à 22.3)
- Booléen : vous pouvez écrire `bool = true` ou `bool = false` puis avec Lua vous pouvez utiliser le code :

```
if bool == ... then ... else ... end
```

et en dehors de l'environnement `tkzelements` vous pouvez utiliser la macro

```
\ifthenelse{\equal{\tkzUseLua{bool}}{true}}{ ... }{ ... }
```

après avoir chargé le package `ifthen`.

- Chaîne de caractères : si `st = "formule d'Euler"` alors

```
\tkzUseLua{st} donne formule d'Euler
```

4.2 Attribution d'un nom à un point

Actuellement, la seule obligation est de stocker les points dans la table `z`⁴ si vous avez l'intention de les utiliser dans TikZ ou `tkz-euclide`. Si un point n'est pas utilisé, vous pouvez le désigner comme vous le souhaitez tout en adhérant aux conventions Lua.

Les points de l'environnement `tkzelements` doivent respecter une convention de la forme `z.name`, où `name` représente le nom du **node** correspondant.

En ce qui concerne les conventions de désignation des noms, vous devez respecter les conventions Lua dans des cas particuliers.

1. L'utilisation du symbole "prime" peut poser problème. Si le nom du point contient plus d'un symbole et se termine par `p`, alors lors du passage dans TikZ ou `tkz-euclide`, les lettres `p` seront remplacées par `'` en utilisant la macro `\tkzGetNodes`;
2. Alternativement, pour un code plus explicite, supposez que vous voulez désigner un point comme "euler". Vous pourriez, par exemple, écrire `euler = ...`, et à la fin du code pour le transfert, `z.E = euler`. Il est également possible d'utiliser un nom temporaire `euler` et de le remplacer dans TikZ. Soit au moment de placer les étiquettes, soit par exemple en utilisant `pgfnodealias{E}{euler}`. Cette possibilité s'applique également dans d'autres cas : prime, double prime, etc.

Voici différentes façons de nommer un point :

```
— z.A = point : new (1,2)
— z.Bp = point : new (3,4) -> ce qui donne B' dans le tikzpicture
— z.H_a = T.ABC : altitude () -> ce qui donne H_a dans le code de l'environnement
  tikzpicture et H_a dans la figure
```

4. Pour placer le point `M` dans la table, il suffit d'écrire `z.M = ...` ou `z["M"] = ...`

4.3 Attribution d'un nom à d'autres objets

Vous avez la possibilité d'attribuer des noms à des objets autres que les points. Cependant, il est conseillé de respecter certaines conventions afin d'améliorer la lisibilité du code. Pour mes exemples, j'ai choisi les conventions suivantes : tout d'abord, je stocke les objets dans des tableaux : L pour les lignes et segments, C pour les cercles, T pour les triangles, E pour les ellipses.

- Pour les droites, j'utilise les noms des deux points qu'elles traversent. Par exemple, si une droite passe par les points A et B , je nomme la droite $L.AB$.
- Les cercles sont stockés dans une table nommée C. Par exemple, je nomme $C.AB$ le cercle de centre A passant par B . D'autres noms comme $C.euler$ ou $C.external$ sont également acceptables.
- Les triangles sont stockés dans une table nommée T. Par exemple, je nomme $T.ABC$ le triangle dont les sommets sont A , B et C . Cependant, des noms comme $T.feuerbach$ sont également acceptables.
- Les ellipses sont stockées dans une table nommée E. Pour les ellipses, je nomme $E.ABC$ l'ellipse avec le centre A passant par le sommet B et le co-sommet C .

Le respect de ces conventions peut contribuer à améliorer la lisibilité du code.

4.4 Conventions d'écriture pour les attributs et les méthodes.

You must use the conventions of Lua, so

- Pour obtenir un , pour tous les objets, la convention est identique : `objet.attribut`. Par exemple, pour le point A , nous accédons à son abscisse avec `z.A.re` et à son ordonnée avec `z.A.im`; quant à son type, nous l'obtenons avec `z.A.type`. Pour obtenir le pôle sud du cercle $C.OA$, vous devez écrire : `C.OA.south`.
- Pour utiliser une méthode telle que l'obtention du cercle inscrit d'un triangle ABC , il suffit d'écrire `C.incircle = T.ABC : in_circle ()`.
- Certaines méthodes nécessitent un paramètre. Par exemple, pour connaître la distance entre un point C et la droite (A,B) , nous écrirons `d = L.AB : distance (z.C)`.
- Utilisez le pour stocker un résultat que vous ne souhaitez pas utiliser. Si vous avez seulement besoin du deuxième point d'une intersection entre une droite et un cercle, vous écrirez `_,z.J = intersection (L.AB , C.OC)`.

5 Organisation du travail

Voici un exemple d'organisation.

The line `% !TEX TS-program = lualatex` garantit que vous compilez avec Lua \LaTeX . La classe `standalone` est utile, car tout ce que vous avez à faire ici est de créer une figure.

Vous pouvez charger `tkz-euclide` de trois façons différentes. La plus simple est `\usepackage[mini]{tkz-euclide}` et vous avez un accès complet au package. Vous avez également la possibilité d'utiliser l'option `lua`. Cela permettra, si vous souhaitez effectuer des calculs en dehors de `tkz-elements`, de les obtenir avec `lua`. Enfin, la méthode recommandée est d'utiliser l'option `mini`. Celle-ci permet de ne charger que les modules nécessaires aux tracés. Vous pouvez toujours éventuellement tracer à l'aide de TikZ.

Le package `ifthen` est utile si vous avez besoin d'utiliser des booléens.

La macro `\LuaCodeDebugOn` vous permet de rechercher et de trouver des erreurs dans le code Lua.

Bien qu'il soit possible de laisser le code Lua dans l'environnement `tkzelements`, externaliser ce code présente des avantages.

Le premier avantage est que, si vous utilisez un bon éditeur, vous avez une meilleure présentation du code. Les styles diffèrent entre Lua et \LaTeX , ce qui rend le code plus clair. C'est ainsi que j'ai procédé, puis j'ai réintégré le code dans le code principal.

Un autre avantage est que vous n'avez pas à commenter incorrectement le code. Pour le code Lua, vous commentez les lignes avec `--` (double tiret), tandis que pour \LaTeX , vous commentez avec `%`.

Un troisième avantage est que le code peut être réutilisé.

```

% !TEX TS-program = lualatex
% Created by Alain Matthes on 2024-01-09.

\documentclass[margin = 12pt]{standalone}
\usepackage{tkz-euclide}
\usepackage{tkz-elements,ifthen}

\begin{document}
\LuaCodeDebugOn
\begin{tkzelements}
  scale = 1.25
  dofile ("sangaku.lua")
\end{tkzelements}

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircle(I,F)
  \tkzFillPolygon[color = purple](A,C,D)%
  \tkzFillPolygon[color = blue!50!black](A,B,C)%
  \tkzFillCircle[color = orange](I,F)%
\end{tikzpicture}
\end{document}

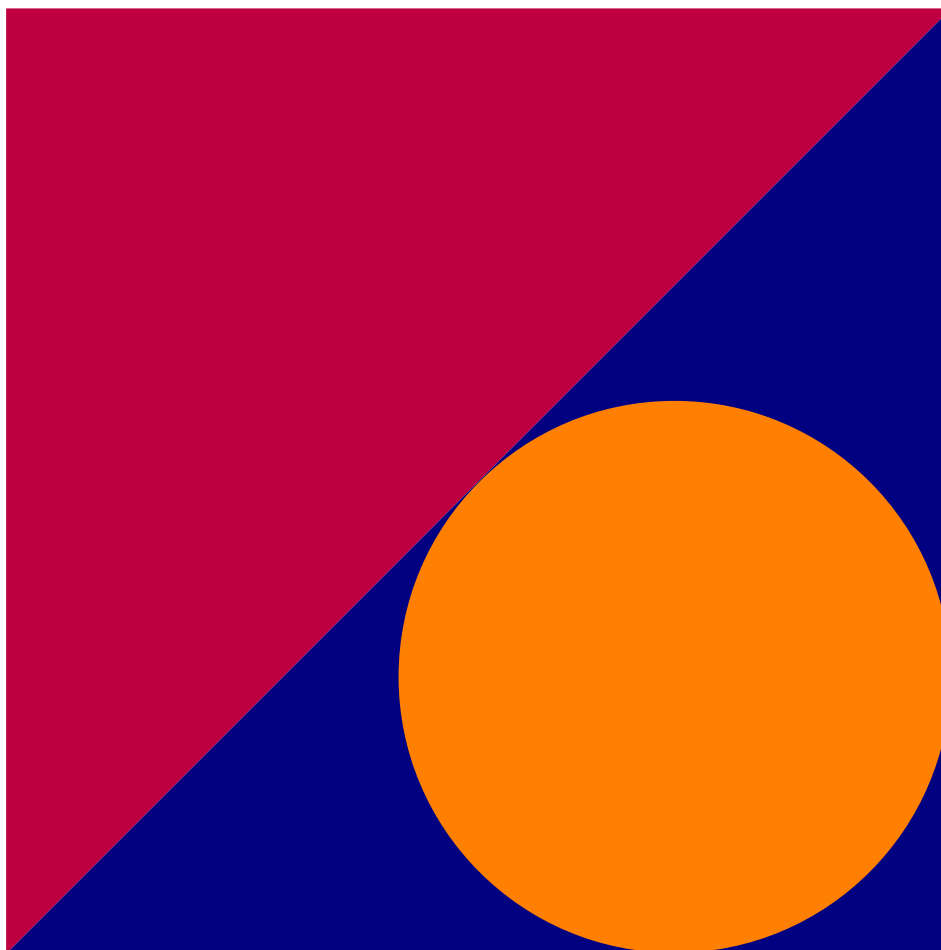
```

Et voici le code pour la partie Lua : le fichier `ex_sangaku.lua`

```

z.A      = point : new ( 0,0 )
z.B      = point : new ( 8,0 )
L.AB     = line : new ( z.A , z.B )
S        = L.AB : square ()
_,_,z.C,z.D = get_points (S)
z.F      = S.ac : projection (z.B)
L.BF     = line : new (z.B,z.F)
T.ABC    = triangle : new ( z.A , z.B , z.C )
L.bi     = T.ABC : bisector (2)
z.c      = L.bi.pb
L.Cc     = line : new (z.C,z.c)
z.I      = intersection (L.Cc,L.BF)

```



5.1 Problème d'échelle

Si nécessaire, il est préférable d'effectuer la mise à l'échelle dans la section Lua. Cette approche tend à être plus précise. Cependant, il y a un inconvénient à prendre en compte. J'ai veillé à éviter d'utiliser des valeurs numériques dans mes codes chaque fois que possible. Généralement, ces valeurs n'apparaissent que dans la définition des points fixes. Si l'option `scale` est utilisée, la mise à l'échelle est appliquée lorsque les points sont créés. Imaginons que vous vouliez organiser votre code comme suit :

```
scale = 1.5
```

```
xB = 8
```

```
z.B = point : new ( xB,0 )
```

La mise à l'échelle serait alors inefficace, car les valeurs numériques ne sont pas modifiées, seules les coordonnées des points le sont. Pour tenir compte de la mise à l'échelle, utilisez la fonction `value (v)` .

```
scale = 1.5
```

```
xB = value (8)
```

```
z.B = point : new ( xB,0 )
```

5.2 Présentation du code

Le point essentiel est que, contrairement à \LaTeX ou \TeX , vous pouvez insérer des espaces absolument partout.

6 Transferts

6.1 De Lua à tkz-euclide ou TikZ

Dans cette section, nous verrons comment transférer des points, des booléens et des valeurs numériques.

6.1.1 Transfert de points

Nous utilisons un environnement `tkzelements` en dehors d'un environnement `tikzpicture`, ce qui nous permet d'effectuer tous les calculs nécessaires. Ensuite, nous exécutons la macro qui transforme les affixes de la table `z` en `Nodes`. Enfin, nous procédons aux tracés.

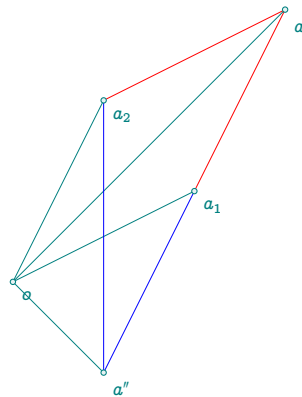
À l'heure actuelle, le programme de tracés est soit `TikZ` soit `tkz-euclide`. Cependant, vous avez la possibilité d'utiliser un autre package pour le tracé. Pour ce faire, vous devez créer une macro similaire à `\tkzGetNodes`. Bien sûr, ce package doit être capable de stocker des points comme `TikZ` ou `tkz-euclide`.

```
\def\tkzGetNodes{\directlua{%
  for K,V in pairs(z) do
    local n,sd,ft
    n = string.len(K)
    if n >1 then
      _,_,ft, sd = string.find( K , "(.+)(") " )
      if sd == "p" then K=ft.."'" end
      _,_,xft, xsd = string.find( ft , "(.+)(") " )
      if xsd == "p" then K=xft.."'".."'" end
    end
    tex.print("\\coordinate ("..K..) at ("..V.re..","..V.im..) ;\\\\"")
  end}
}
```

Voir la section Étude approfondie 22 pour une explication du code précédent.

L'environnement `tkzelements` permet d'utiliser le souligné `_` et la macro `\tkzGetNodes` permet d'obtenir des noms de nœuds contenant `prime` ou `double prime`. (Référez-vous à l'exemple suivant)

```
\begin{tkzelements}
  scale = 1.2
  z.o = point: new (0,0)
  z.a_1 = point: new (2,1)
  z.a_2 = point: new (1,2)
  z.ap = z.a_1 + z.a_2
  z.app = z.a_1 - z.a_2
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegments(o,a_1 o,a_2 o,a' o,a'')
  \tkzDrawSegments[red](a_1,a' a_2,a')
  \tkzDrawSegments[blue](a_1,a'' a_2,a'')
  \tkzDrawPoints(a_1,a_2,a',o,a'')
  \tkzLabelPoints(o,a_1,a_2,a',a'')
\end{tikzpicture}
```



6.1.2 Autres transferts

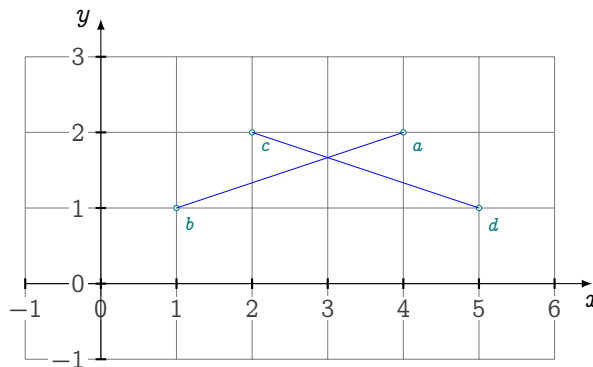
Il est parfois utile de transférer des mesures d'angle, de longueur ou des booléens. A cette fin, j'ai créé la macro (se référer à 20.6) `tkzUseLua(value)`

```
\begin{tkzelements}
  z.b = point: new (1,1)
  z.a = point: new (4,2)
  z.c = point: new (2,2)
  z.d = point: new (5,1)
  L.ab = line : new (z.a,z.b)
  L.cd = line : new (z.c,z.d)
  det = (z.b-z.a)^(z.d-z.c)
  if det == 0 then bool = true
  else bool = false
  end
  x = intersection (L.ab,L.cd)
\end{tkzelements}
```

The intersection of the two lines lies at
a point whose affix is: `\tkzUseLua{x}`

```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzInit[xmin=-1,ymin=-1,xmax=6,ymax=3]
  \tkzGrid\tkzAxeX\tkzAxeY
  \tkzDrawPoints(a,...,d)
  \ifthenelse{\equal{\tkzUseLua{bool}}{true}}{
  \tkzDrawSegments[red](a,b c,d)}{
  \tkzDrawSegments[blue](a,b c,d)}
  \tkzLabelPoints(a,...,d)
\end{tikzpicture}
```

L'intersection des deux lignes se situe en un point dont l'affixe est un point dont l'affixe est : $3+1.67i$



7 Classe et objet

7.1 Classe

La programmation orientée objet (OOP) est un modèle de programmation basé sur le concept d'objets. Un objet peut être défini comme une table de données qui possède des attributs uniques et des méthodes (opérations) qui définissent son comportement.

Une classe est essentiellement un type de données défini par l'utilisateur. Elle décrit le contenu des objets qui lui appartiennent. Une classe sert de modèle pour créer des objets, fournissant des valeurs initiales pour les attributs et des implémentations des méthodes⁵ qui sont communes à tous les objets d'un certain type.

7.2 Objet

Un objet est une instance d'une classe. Chaque objet contient des attributs et des méthodes. Les attributs sont des informations ou des caractéristiques de l'objet stockées dans la table de données (appelée champs), tandis que les méthodes définissent le comportement de l'objet.

Tous les objets du paquet sont typés. Les types d'objets actuellement définis et utilisés sont les suivants **point**, **line**, **circle**, **triangle**, **ellipse**, **quadrilateral**, **square**, **rectangle**, **parallelogram** and **regular_polygon**. Ces objets peuvent être créés directement à l'aide de la méthode `new` en donnant des points, à l'exception de la classe `classpoint` qui nécessite une paire de réels, et `classregular_polygon` qui nécessite deux points et un entier.

Les objets peuvent également être obtenus en appliquant des méthodes à d'autres objets.

Par exemple, `T.ABC : circum_circle ()` crée un objet **circle**. Certains attributs d'objets sont eux-mêmes des objets, comme `T.ABC.bc` qui crée l'objet **line**, représentant une ligne droite passant par les deux derniers points définissant le triangle.

7.2.1 Attributs

Les attributs sont accessibles en utilisant la méthode classique, ainsi `T.pc` récupère le troisième point du triangle et `C.OH.center` récupère le centre du cercle. De plus, j'ai ajouté une fonction `get_points` qui renvoie les points d'un objet. Cette fonction s'applique aux droites (`pa` et `pc`), aux triangles (`pa`, `pb` et `pc`) et aux cercles (`center` et `through`).

Exemple : `z.O,z.T = get_points (C)` récupère le centre et un point du cercle.

7.2.2 Méthodes

Une méthode est une opération (fonction ou procédure) associée (liée) à un objet.

Exemple : L'objet `point` est utilisé pour déterminer verticalement un nouvel objet `point` situé à une certaine distance de celui-ci (ici 2). Il est ensuite possible de faire tourner des objets autour de lui.

```
\begin{tkzelements}
  z.A = point (1,0)
  z.B = z.A : north (2)
  z.C = z.A : rotation (math.pi/3,z.B)
  tex.print(tostring(z.C))
\end{tkzelements}
```

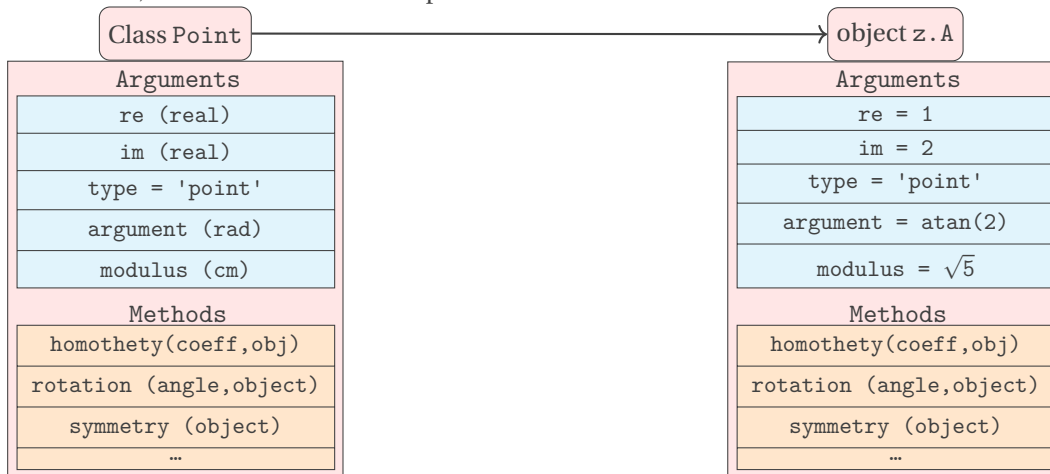
The coordinates of *C* are : -0.73205080756888 and 1.0

5. action qu'un objet est capable d'effectuer.

8 Classe point

La fondation de l'ensemble du cadre est la classe `point`. Cette classe est hybride dans le sens où elle traite à la fois des points dans un plan et des nombres complexes. Le principe est le suivant : le plan est équipé d'une base orthonormale, ce qui nous permet de déterminer la position d'un point en utilisant ses coordonnées d'abscisse et d'ordonnée. De même, tout nombre complexe peut être simplement vu comme une paire de nombres réels (sa partie réelle et sa partie imaginaire). Nous pouvons alors désigner le plan comme le plan complexe, et le nombre complexe $x + iy$ est représenté par le point du plan avec les coordonnées (x, y) . Ainsi, le point A aura ses coordonnées stockées dans l'objet `z.A`. Les coordonnées sont des attributs de l'objet "point", avec le type, l'argument et le module.

La création d'un point se fait selon la méthode suivante, mais il existe d'autres possibilités. Si un facteur d'échelle a été donné, la méthode en tient compte.



8.1 Attributs d'un point

```
Creation z.A = point: new (1,2)
```

Le point A a pour coordonnées $x = 1$ et $y = 2$. Si vous utilisez la notation `z.A`, alors A sera référencé comme un nœud dans TikZ ou dans tkz-euclide.

Ceci est la création d'un point fixe avec des coordonnées 1 et 2 et qui est nommé A . La notation `z.A` indique que les coordonnées seront stockées dans une table désignée par `z` (référence à la notation des affixes des nombres complexes) que A est le nom du point et la clé permettant d'accéder aux valeurs.

TABLE 1 – Point attributes.

Attributes	Application	Example
<code>re</code>	<code>z.A.re = 1</code>	Refer to (7.2.2)
<code>im</code>	<code>z.A.im = 2</code>	Refer to (7.2.2)
<code>type</code>	<code>z.A.type = 'point'</code>	
<code>argument</code>	<code>z.A.argument</code> ≈ 0.78539816339745	Refer to (8.1.1)
<code>modulus</code>	<code>z.A.modulus</code> $\approx 2.236065572717203$ $=\sqrt{5}$	Refer to (8.1.1)

8.1.1 Exemple : attributs de points

```

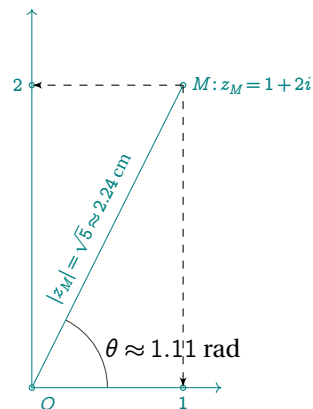
\begin{tkzelements}
  z.M = point: new (1,2)
\end{tkzelements}

\begin{tikzpicture}[scale = 1]
\pgfkeys{/pgf/number format/.cd,std,precision=2}
\let\pmpn\pgfmathprintnumber
\tkzDefPoints{2/4/M,2/0/A,0/0/O,0/4/B}
\tkzLabelPoints(O)
\tkzMarkAngle[fill=gray!30,size=1](A,O,M)
\tkzLabelAngle[pos=1,right](A,O,M){%
 $\theta \approx \pmpn{\tkzUseLua{z.M.argument}}$  rad}
\tkzDrawSegments(O,M)
\tkzLabelSegment[above,sloped](O,M){%
 $|z_M| = \sqrt{5} \approx \pmpn{\tkzUseLua{z.M.modulus}}$  cm}
\tkzLabelPoint[right](M){ $M : z_M = 1 + 2i$ }
\tkzDrawPoints(M,A,O,B)
\tkzPointShowCoord(M)
\tkzLabelPoint[below,teal](A){ $\tkzUseLua{z.M.re}$ }
\tkzLabelPoint[left,teal](B){ $\tkzUseLua{z.M.im}$ }
\tkzDrawSegments[->,add = 0 and 0.25](O,B O,A)
\end{tikzpicture}

```

Attributs de $z.M$

- $z.M.re = 1$
- $z.M.im = 2$
- $z.M.type = 'point'$
- $z.M.argument = \theta \approx 1.11 \text{ rad}$
- $z.M.modulus = |z_M| = \sqrt{5} \approx 2.24 \text{ cm}$

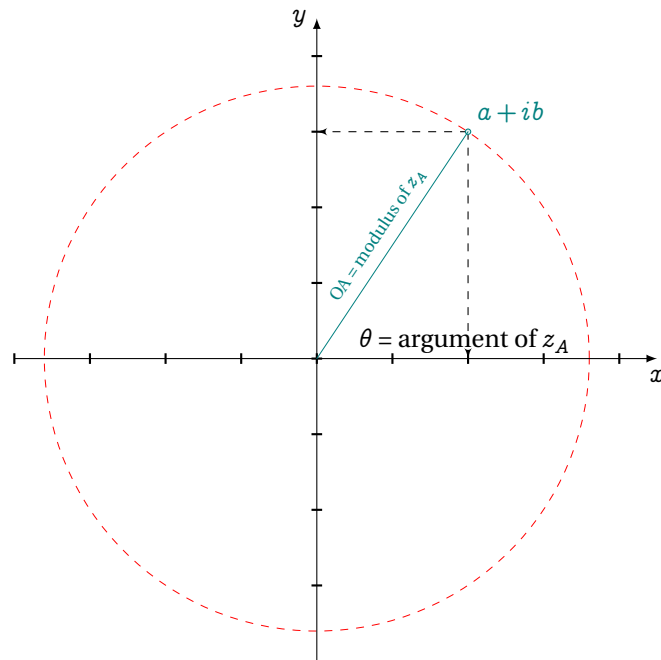


8.1.2 Diagramme d'Argand

```

\begin{tkzelements}
  z.A = point : new ( 2 , 3 )
  z.O = point : new ( 0 , 0 )
  z.I = point : new ( 1 , 0 )
\end{tkzelements}
\hspace{\fill}\begin{tikzpicture}
  \tkzGetNodes
  \tkzInit[xmin=-4,ymin=-4,xmax=4,ymax=4]
  \tkzDrawCircle[dashed,red](O,A)
  \tkzPointShowCoord(A)
  \tkzDrawPoint(A)
  \tkzLabelPoint[above right](A){\normalsize  $a+ib$ }
  \tkzDrawX\tkzDrawY
  \tkzDrawSegment(O,A)
  \tkzLabelSegment[above,anchor=south,sloped](O,A){  $OA = \text{module de } z_A$  }
  \tkzLabelAngle[anchor=west,pos=.5](I,O,A){  $\theta = \text{argument de } z_A$  }
\end{tikzpicture}

```



8.2 Méthodes de la classe point

Les méthodes décrites dans le tableau suivant sont standard et peuvent être trouvées dans la plupart des exemples à la fin de cette documentation. Le résultat des différentes méthodes présentées dans le tableau suivant est un `point`. Référez-vous à la section (22.3) pour les métaméthodes.

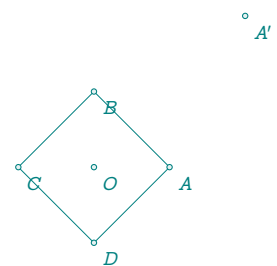
TABLE 2 – Fonctions & Méthodes de la classe point.

Fonctions	Application	
<code>new(r,r)</code>	<code>z.A = point : new(1,2)</code>	Refer to (8.2.4)
<code>polar (d,an)</code>	<code>z.A = point : polar(1,math.pi/3)</code>	Refer to (23.7)
<code>polar_deg (d,an)</code>	<code>an en deg</code>	coordonnées polaires en deg
Méthodes	Application	
Points		
<code>north(r)</code>	<code>r distance par rapport au point (1 if empty)</code>	Refer to (23.43); 7.2.2
<code>south(r)</code>		
<code>east(r)</code>		
<code>west(r)</code>		
<code>normalize()</code>	<code>z.b = z.a: normalize ()</code>	Se référer à (8.2.4)
<code>get_points (obj)</code>	récupère les points de l'objet	
<code>orthogonal (d)</code>	<code>z.B=z.A:orthogonal(d)</code>	$\overrightarrow{OB} \perp \overrightarrow{OA}$ and $OB = d$
<code>at ()</code>	<code>z.X = z.B : at (z.A)</code>	$\overrightarrow{OB} = \overrightarrow{AX}$ et $OB = d$
Transformations		
<code>symmetry(obj)</code>	<code>obj : point, line, etc.</code>	Se référer à (8.2.9)
<code>rotation(an , obj)</code>	<code>point, line, etc.</code>	Se référer à (8.2.8)
<code>homothety(r,obj)</code>	<code>z.c = z.a : homothety (2,z.b)</code>	Refer to (23.48)
Misc.		
<code>print ()</code>	affiche l'affixe du point	Se référer à (8.2.9)

8.2.1 Exemple : méthode north (d)

Cette fonction définit un point situé sur une ligne verticale passant par le point donné. Cette fonction est utile si vous voulez reporter une certaine distance (référez-vous à l'exemple suivant). Si `d` est absent, il est considéré comme égal à 1.

```
\begin{tkzelements}
  z.O = point : new ( 0, 0 )
  z.A = z.O : east ( )
  z.Ap = z.O : east (2) : north (2)
  z.B = z.O : north ( )
  z.C = z.O : west ( )
  z.D = z.O : south ( )
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C,D)
  \tkzDrawPoints(A,B,C,D,O,A')
\end{tikzpicture}
```



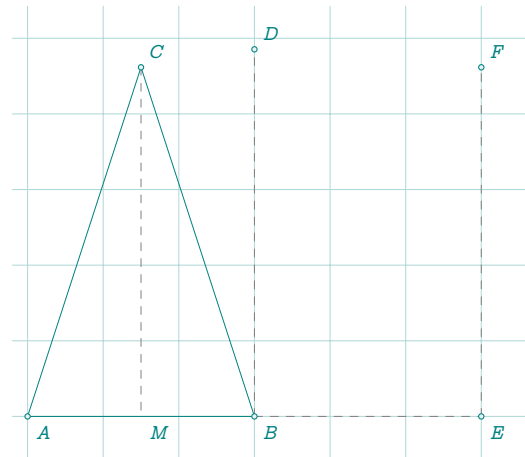
8.2.2 Transfert de longueur

Utilisation de `north` and `east` fonctions liées à des points, à des longueurs de transfert. Référez à (20.1)

```

\begin{tkzelements}
  z.A = point : new ( 0 , 0 )
  z.B = point : new ( 3 , 0 )
  L.AB = line : new ( z.A , z.B )
  T.ABC = L.AB : sublime ( )
  z.C = T.ABC.pc
  z.D = z.B: north (length(z.B,z.C))
  z.E = z.B: east (L.AB.length)
  z.M = L.AB.mid
  z.F = z.E : north (length(z.C,z.M))
\end{tkzelements}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawPolygons(A,B,C)
  \tkzDrawSegments[gray,dashed](B,D B,E E,F C,M)
  \tkzDrawPoints(A,...,F)
  \tkzLabelPoints(A,B,E,M)
  \tkzLabelPoints[above right](C,D,F)
\end{tikzpicture}

```



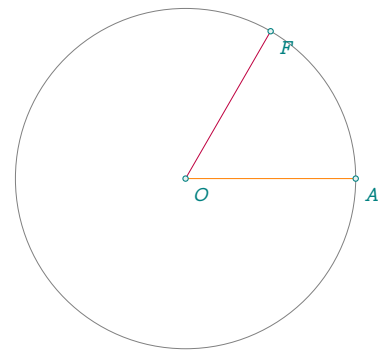
8.2.3 Exemple: méthode polar

Il s'agit de définir un point à l'aide de son module et de son argument.

```

\begin{tkzelements}
  z.O = point: new (0,0)
  z.A = point: new (3,0)
  z.F = point: polar (3, math.pi/3)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircle(O,A)
  \tkzDrawSegments[new](O,A)
  \tkzDrawSegments[purple](O,F)
  \tkzDrawPoints(A,O,F)
  \tkzLabelPoints[below right=6pt](A,O,F)
\end{tikzpicture}

```



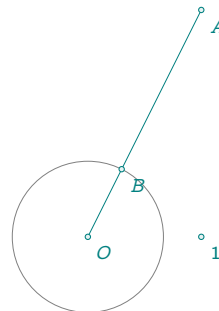
8.2.4 Méthode normalize ()

Le résultat est un point situé entre l'origine et le point initial à une distance de 1 de l'origine.

```

\begin{tkzelements}
  scale = 1.5
  z.O = point : new (0,0)
  z.A = point : new (1,2)
  z.B = z.A : normalize ( )
  z.I = point : new (1,0)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegment(O,A)
  \tkzDrawCircle(O,B)
  \tkzDrawPoints(O,A,B,I)
  \tkzLabelPoints(O,A,B)
  \tkzLabelPoint[below right](I){$1$}
\end{tikzpicture}

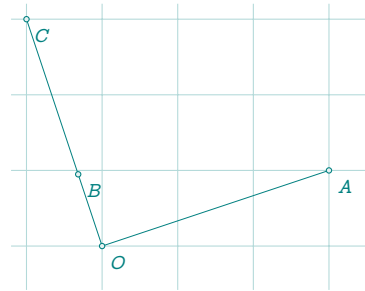
```



8.2.5 Orthogonal (d) méthode

Soit O l'origine du plan. La méthode "orthogonal (d)" est utilisée pour obtenir un point B à partir d'un point A tel que $\overrightarrow{OB} \perp \overrightarrow{OA}$ avec $OB = OA$ si d est vide, sinon $OB = d$.

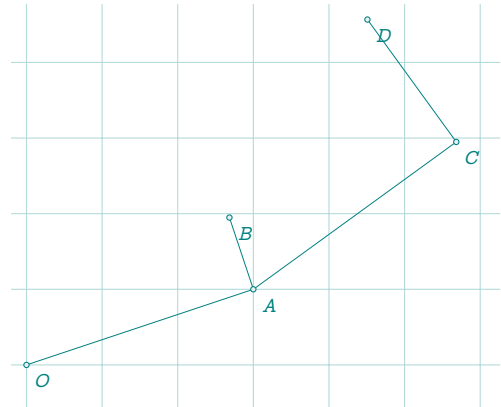
```
\begin{tkzelements}
  z.A = point : new ( 3 , 1 )
  z.B = z.A : orthogonal (1)
  z.O = point : new ( 0,0 )
  z.C = z.A : orthogonal ( )
\end{tkzelements}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawSegments(O,A O,C)
  \tkzDrawPoints(O,A,B,C)
  \tkzLabelPoints[below right](O,A,B,C)
\end{tikzpicture}
```



8.2.6 at méthode

Cette méthode est complémentaire à la précédente, donc vous ne souhaitez peut-être pas avoir $\overrightarrow{OB} \perp \overrightarrow{OA}$ mais $\overrightarrow{AB} \perp \overrightarrow{OA}$.

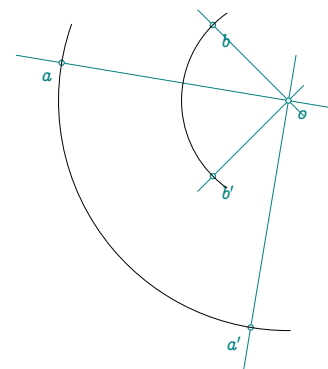
```
\begin{tkzelements}
  z.A = point : new ( 3 , 1 )
  z.B = z.A : orthogonal (1)
  z.O = point : new ( 0,0 )
  -- z.B = z.B : at (z.A) -- or
  z.B = z.A : orthogonal (1) : at (z.A)
  z.C = z.A+z.B
  z.D =(z.C-z.A):orthogonal(2) : at (z.C)
\end{tkzelements}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzLabelPoints[below right](O,A,B,C,D)
  \tkzDrawSegments(O,A A,B A,C C,D)
  \tkzDrawPoints(O,A,B,C,D)
\end{tikzpicture}
```



8.2.7 Exemple: rotation of points

Les arguments sont l'angle de rotation en radians, et ici une liste de points.

```
\begin{tkzelements}
  z.a      = point: new(0, -1)
  z.b      = point: new(4, 0)
  z.o      = point: new(6, -2)
  z.ap,z.bp = z.o : rotation (math.pi/2,z.a,z.b)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(o,a o,a' o,b o,b')
  \tkzDrawPoints(a,a',b,b',o)
  \tkzLabelPoints(b,b',o)
  \tkzLabelPoints[below left](a,a')
  \tkzDrawArc(o,a)(a')
  \tkzDrawArc(o,b)(b')
\end{tikzpicture}
```

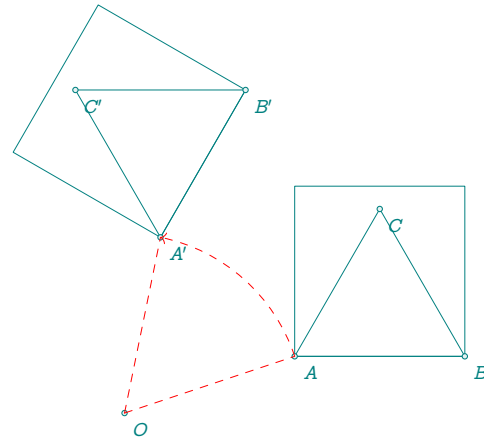


8.2.8 Objet rotation

Reduire un triangle d'un angle de $\pi/6$ autour de O .

```
\begin{tkzelements}
  scale = .75
  z.O = point : new ( -1 , -1 )
  z.A = point : new ( 2 , 0 )
  z.B = point : new ( 5 , 0 )
  L.AB = line : new (z.A,z.B)
  T.ABC = L.AB : equilateral ()
  S.fig = L.AB : square ()
  _,_,z.E,z.F = get_points ( S.fig )
  S.new = z.O : rotation (math.pi/3,S.fig)
  _,_,z.Ep,z.Fp = get_points ( S.new )
  z.C = T.ABC.pc
  T.ApBpCp = z.O : rotation (math.pi/3,T.ABC)
  z.Ap,z.Bp,z.Cp = get_points ( T.ApBpCp)
\end{tkzelements}
```

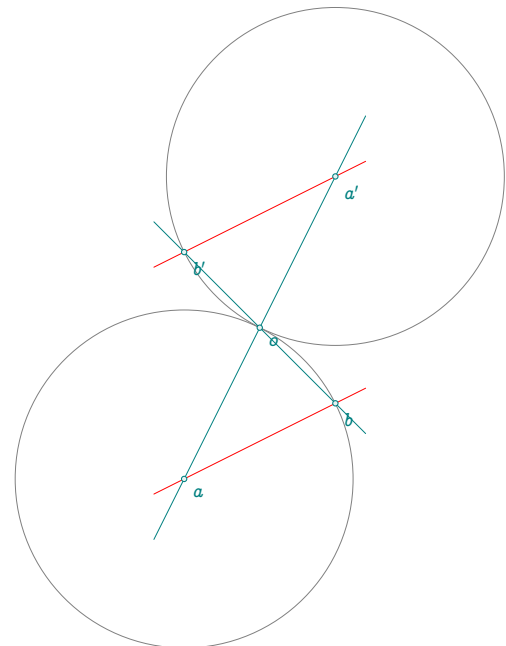
```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygons(A,B,C A',B',C' A,B,E,F A',B',E',F')
  \tkzDrawPoints (A,B,C,A',B',C',O)
  \tkzLabelPoints (A,B,C,A',B',C',O)
  \tkzDrawArc[delta=0,->](O,A)(A')
\end{tikzpicture}
```



8.2.9 Objet symmetry

```
\begin{tkzelements}
  z.a = point: new(0,-1)
  z.b = point: new(2, 0)
  L.ab = line : new (z.a,z.b)
  C.ab = circle : new (z.a,z.b)
  z.o = point: new(1,1)
  z.ap,z.bp = get_points (z.o: symmetry (C.ab))
\end{tkzelements}
```

```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles(a,b a',b')
  \tkzDrawLines(a,a' b,b')
  \tkzDrawLines[red](a,b a',b')
  \tkzDrawPoints(a,a',b,b',o)
  \tkzLabelPoints(a,a',b,b',o)
\end{tikzpicture}
```



9 Classe line (droite)

9.1 Attributs d'une droite (line)

L'écriture `L.AB = line: new (z.A,z.B)` crée un objet de la classe **line** (la notation est arbitraire pour le moment). Géométriquement, cela représente à la fois la droite passant par les points A et B ainsi que le segment $[AB]$. Ainsi, nous pouvons utiliser le milieu de `L.AB`, qui est, bien sûr, le milieu du segment $[AB]$. Ce milieu est obtenu avec `L.AB.mid`. Notez que `L.AB.pa = z.A` et `L.AB.pb = z.B`. Enfin, si une droite L est le résultat d'une méthode, vous pouvez obtenir les points avec `z.A,z.B = get_points (L)` ou avec la remarque précédente.

```
Creation L.AB = line : new ( z.A , z.B )
```

Les attributs sont les suivants :

TABLE 3 – Attributs de la droite.

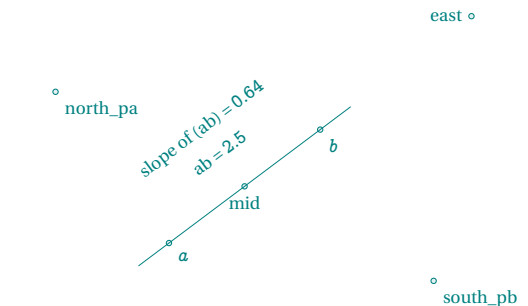
Attributs	Application	
pa	Premier point du segment	<code>z.A = L.AB.pa</code>
pb	Deuxième point du segment	
type	Le type est 'line'	<code>L.AB.type = 'line'</code>
mid	Milieu du segment	<code>z.M = L.AB.mid</code>
slope	Pente de la droite	Se référer à (9.1.1)
length	<code>l = L.AB.length</code>	Se référer à (20.6; 9.1.1)
north_pa		Se référer à (9.1.1)
north_pb		
south_pa		
south_pb		Se référer à (9.1.1)
east		
west		
vec	<code>V.AB = L.AB.vec</code>	définit \overrightarrow{AB} Se référer à (18)

9.1.1 Exemple : attributs de la classe line

```

\begin{tkzelements}
  scale = .5
  z.a = point: new (1, 1)
  z.b = point: new (5, 4)
  L.ab = line : new (z.a,z.b)
  z.m = L.ab.mid
  z.w = L.ab.west
  z.e = L.ab.east
  z.r = L.ab.north_pa
  z.s = L.ab.south_pb
  sl = L.ab.slope
  len = L.ab.length
\end{tkzelements}

```



```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPoints(a,b,m,e,r,s,w)
  \tkzLabelPoints(a,b,e,r,s,w)
  \tkzLabelPoints[above](m)
  \tkzDrawLine(a,b)
  \tkzLabelSegment[sloped](a,b){ab = \tkzUseLua{len}}
  \tkzLabelSegment[above=12pt,sloped](a,b){slope of (ab) = \tkzUseLua{sl}}
\end{tikzpicture}

```

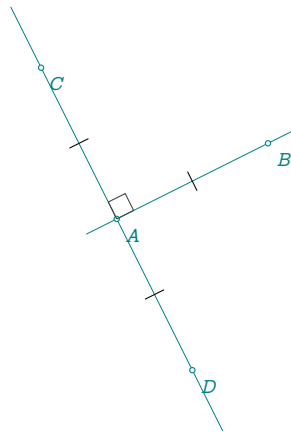
9.1.2 Méthode new

La notation peut être L ou L.AB ou L.euler. La notation est en fait libre. L.AB peut aussi représenter le segment. Avec `L.AB = ligne : nouveau (z.A,z.B)`, une ligne est définie.

```

\begin{tkzelements}
  z.A = point : new (1,1)
  z.B = point : new (3,2)
  L.AB = line : new (z.A,z.B)
  z.C = L.AB.north_pa
  z.D = L.AB.south_pa
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(A,B C,D)
  \tkzDrawPoints(A,...,D)
  \tkzLabelPoints(A,...,D)
  \tkzMarkRightAngle(B,A,C)
  \tkzMarkSegments(A,C A,B A,D)
\end{tikzpicture}

```



9.2 Méthodes de la classe line

Voici la liste des méthodes de l'objet **line**. Les résultats peuvent être des nombres réels, des points, des lignes, des cercles ou des triangles. Les triangles obtenus sont similaires aux triangles définis ci-dessous.

TABLE 4 – Méthodes de la classe line.(part 1)

Méthodes	Comments	
new(pt, pt)	L.AB = line : new(z.A,z.B)	Créer une droite (AB) ; Se référer à (10.2.1)
Points		
gold_ratio ()	z.C=L.AB : gold_ratio()	Se référer à (23.21; 3.3; 23.8)
normalize ()	z.C=L.AB : normalize()	$AC=1$ et $C \in (AB)$ Se référer à (9.2.8)
normalize_inv ()	z.C=L.AB : normalize_inv()	$CB=1$ and $C \in (AB)$
barycenter (r,r)	z.C=L.AB : barycenter (1,2)	Se référer à (9.2.9)
point (r)	z.C=L.AB : point (2)	$\vec{AC} = 2\vec{AB}$ Se référer à (23.14; 9.2.6)
midpoint ()	z.M=L.AB : midpoint ()	better is z.M = L.AB.mid
harmonic_int (pt)	z.D=L.AB : harmonic_int (z.C)	Se référer à (23.8)
harmonic_ext (pt)	z.D=L.AB : harmonic_ext (z.C)	Se référer à (23.8)
harmonic_both (r)	z.C,z.D=L.AB : harmonic_both(φ)	20.2
_east(d)	z.M=L.AB : _east(2)	$BM = 2A, B, M$ aligned
_west(d)	z.M=L.AB : _east(2)	$BM = 2A, B, M$ aligned
_north_pa(d)	z.M=L.AB: _north_pa(2)	$AM=2$ $AM \perp AB$; \vec{AB}, \vec{AM} counterclockwise ^a
_south_pa(d)	z.M=L.AB: _south_pa(2)	$AM=2$; $AM \perp AB$; \vec{AB}, \vec{AM} clockwise ^b
_north_pb(d)	z.M=L.AB: _north_pb(2)	$BM=2$; $BM \perp BA$; \vec{BA}, \vec{BM} clockwise
_south_pb(d)	z.M=L.AB: _south_pb(2)	$BM=2$; $BM \perp BA$; \vec{AB}, \vec{AM} counterclockwise
report(d,pt)	z.M=L.AB:report (2,z.N)	$MN=2$; $AB \parallel MN$; Refer to ex. (9.2.1)
Lines		
ll_from (pt)	L.CD=L.AB: ll_from(z.C)	$(CD) \parallel (AB)$
ortho_from (pt)	L.CD=L.AB: ortho_from(z.C)	$(CD) \perp (AB)$
mediator ()	L.uv=L.AB: mediator()	(u, v) mmédiatrice de (A, B)
Triangles		
equilateral (<swap>)	T.ABC=L.AB:equilateral()	$(\vec{AB}, \vec{AC}) > 0$ ou < 0 avec swap ^c
isosceles (an<,swap>)	T.ABC=L.AB:isosceles(math.pi/6)	
two_angles (an,an)	T.ABC=L.AB:two_angles(an,an)	note ^d Refer to (9.2.2)
school ()	30°,60°, 90°	
sss (r,r)	$AC = r$ $BC = r$	
as (r,an)	$AC = r$ $\widehat{BAC} = an$	
sa (r,an)	$AC = r$ $\widehat{ABC} = an$	
Sacred triangles		
gold (<swap>)	T.ABC=L.AB:gold()	right in B and $AC = \varphi \times AB$
euclide (<swap>)	T.ABC=L.AB:euclide()	$AB = AC$; $(\vec{AB}, \vec{AC}) = \pi/5$
golden (<swap>)	T.ABC=L.AB:golden()	$(\vec{AB}, \vec{AC}) = 2 \times \pi/5$
divine ()		
egyptian ()		
cheops ()		
Squares		
square ()	S.AB=L.AB : square ()	créer un carré S.AB. ^e

a. dans le sens inverse des aiguilles d'une montre

b. sens des aiguilles d'une montre

c. Les triangles sont définis dans le sens direct de la rotation, sauf si l'option "swap" est présente..

d. Le côté donné est compris entre les deux angles

e. _,_,z.C,z.D = get_points(S.AB)

TABLE 5 – Méthodes de la classe line.(part 2)

Méthodes	Comments	
Cercles		
circle ()	C.AB = L.AB : circle ()	centre pa passant par pb
circle_swap ()	C.BA = L.AB : circle_swap ()	centre pb passant pa
apollonius (r)	C.apo = L.AB : apollonius (2)	Ensemble des points tq. $MA/MB = 2$
Transformations		
reflection (obj)	new obj = L.AB : reflection (obj)	
translation (obj)	new obj = L.AB : translation (obj)	
projection (obj)	z.H = L.AB : projection (z.C)	$CH \perp (AB)$ and $H \in (AB)$
Divers		
distance (pt)	d = L.Ab : distance (z.C)	Se référer à 9.2.14
in_out (pt)	b = L.AB: in_out(z.C)	b=true if $C \in (AB)$
slope ()	a = L.AB : slope()	meilleur est L.AB.slope
in_out_segment (pt)	b = L.AB : in_out_segment(z.C)	b=true if $C \in [AB]$

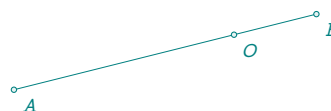
9.2.1 Méthode report

Si le point est absent, le transfert est effectué à partir du premier point qui définit la ligne.

```

\begin{tkzelements}
z.A = point : new (1,-1)
z.B = point : new (5,0)
L.AB = line : new ( z.A , z.B )
z.M = point : new (2,3)
z.N = L.AB : report (3,z.M)
z.O = L.AB : report (3)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawSegments(A,B M,N)
\tkzDrawPoints(A,B,M,N,O)
\tkzLabelPoints(A,B,M,N,O)
\end{tikzpicture}

```



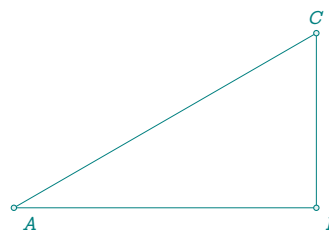
9.2.2 Triangle avec two_angles

Les angles sont situés de part et d'autre du segment donné

```

\begin{tkzelements}
z.A = point : new ( 0 , 0 )
z.B = point : new ( 4 , 0 )
L.AB = line : new ( z.A , z.B )
T.ABC = L.AB : two_angles (math.pi/6,math.pi/2)
z.C = T.ABC.pc
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygons(A,B,C)
\tkzDrawPoints(A,B,C)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C)
\end{tikzpicture}

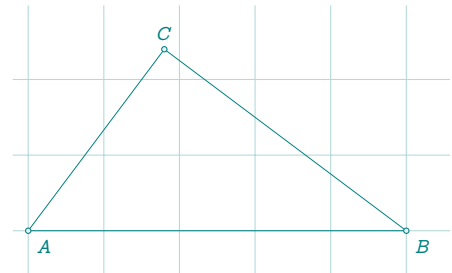
```



9.2.3 Triangle à trois côtés donnés

Dans l'exemple suivant, une petite difficulté se présente. Les longueurs données ne sont pas affectées par l'échelle, il est donc nécessaire d'utiliser la fonction `value(x)`, qui modifiera les longueurs en fonction de l'échelle.

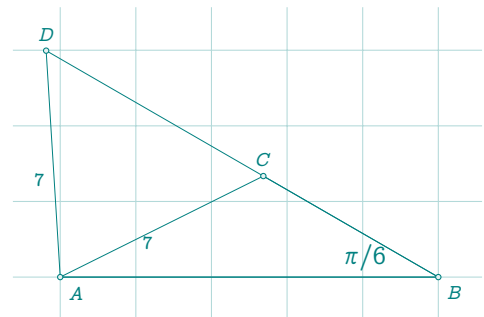
```
\begin{tkzelements}
  z.A = point : new ( 0 , 0 )
  z.B = point : new ( 5 , 0 )
  L.AB = line : new ( z.A , z.B )
  T.ABC = L.AB : sss (value(3),value(4))
  z.C = T.ABC.pc
\end{tkzelements}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawPolygons(A,B,C)
  \tkzDrawPoints(A,B,C)
  \tkzLabelPoints(A,B)
  \tkzLabelPoints[above](C)
\end{tikzpicture}
```



9.2.4 Triangle dont le côté est compris entre le côté et l'angle

Dans certains cas, deux solutions sont possibles.

```
\begin{tkzelements}
  scale =1
  z.A = point : new ( 0 , 0 )
  z.B = point : new ( 5 , 0 )
  L.AB = line : new ( z.A , z.B )
  T.ABC,T.ABD = L.AB : ssa (value(3),math.pi/6)
  z.C = T.ABC.pc
  z.D = T.ABD.pc
\end{tkzelements}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawPolygons(A,B,C A,B,D)
  \tkzDrawPoints(A,B,C,D)
  \tkzLabelPoints(A,B)
  \tkzLabelPoints[above](C,D)
  \tkzLabelAngle[teal](C,B,A){$\pi/6$}
  \tkzLabelSegment[below left](A,C){$7$}
  \tkzLabelSegment[below left](A,D){$7$}
\end{tikzpicture}
```



9.2.5 À propos des triangles sacrés

Les longueurs des côtés sont proportionnelles aux longueurs données dans le tableau. Elles dépendent de la longueur du segment initial.

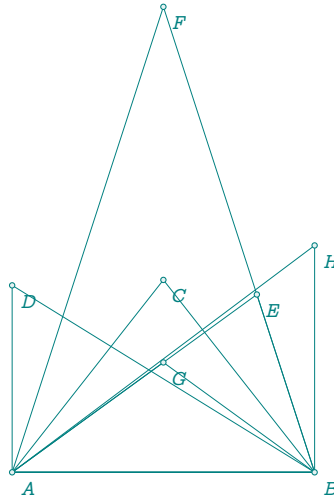
TABLE 6 – Triangles sacrés.

Nom	définition
gold (<swap>)	Triangle rectangle avec $a = \varphi$, $b = 1$ and $c = \sqrt{\varphi}$
golden (<swap>)	Triangle rectangle $b = \varphi$ $c = 1$; moitié du rectangle d'or
divine ()	Isocèle $a = \varphi$, $b = c = 1$ et $\beta = \gamma = \pi/5$
pythagoras ()	$a = 5$, $b = 4$, $c = 3$ et autres noms : isis or egyptian
sublime ()	Isocèle $a = 1$, $b = c = \varphi$ et $\beta = \gamma = 2\pi/5$; other name : euclid
cheops ()	Isocèle $a = 2$, $b = c = \varphi$ et height = $\sqrt{\varphi}$

```

\begin{tkzelements}
  z.A = point : new ( 0 , 0 )
  z.B = point : new ( 4 , 0 )
  L.AB = line : new ( z.A , z.B )
  T.ABC = L.AB : cheops ()
  z.C = T.ABC.pc
  T.ABD = L.AB : gold ()
  z.D = T.ABD.pc
  T.ABE = L.AB : euclide ()
  z.E = T.ABE.pc
  T.ABF = L.AB : golden ()
  z.F = T.ABF.pc
  T.ABG = L.AB : divine ()
  z.G = T.ABG.pc
  T.ABH = L.AB : pythagoras ()
  z.H = T.ABH.pc
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygons(A,B,C A,B,D A,B,E A,B,F A,B,G A,B,H)
  \tkzDrawPoints(A,...,H)
  \tkzLabelPoints(A,...,H)
\end{tikzpicture}

```



9.2.6 Méthode point point

Cette méthode est très utile. Elle permet de placer un point sur la ligne considérée. Si $r = 0$ le point est pa, si $r = 1$ c'est pb.

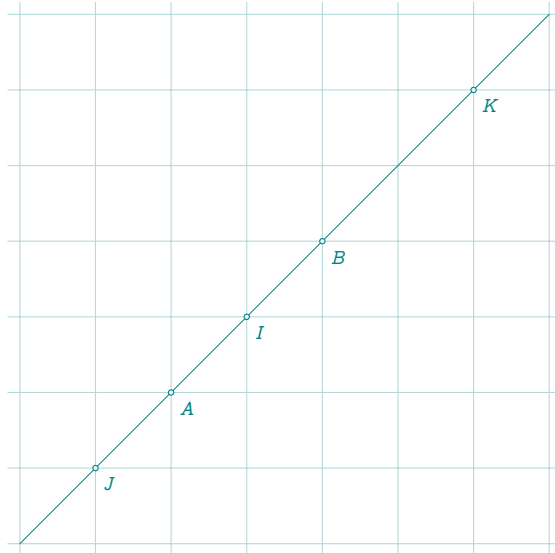
Si $r = .5$ le point obtenu est le milieu du segment. r peut être négatif ou supérieur à 1.

Cette méthode existe pour tous les objets sauf les quadrilatères.

```

\begin{tkzelements}
  z.A = point : new (-1,-1)
  z.B = point : new (1,1)
  L.AB = line : new (z.A,z.B)
  z.I = L.AB : point (0.5)
  z.J = L.AB : point (-0.5)
  z.K = L.AB : point (2)
\end{tkzelements}
\begin{tikzpicture}[gridded]
\tkzGetNodes
  \tkzDrawLine(J,K)
  \tkzDrawPoints(A,B,I,J,K)
  \tkzLabelPoints(A,B,I,J,K)
\end{tikzpicture}

```



9.2.7 Méthode `colinear_at`

Si le coefficient est absent alors il vaut par défaut 1 et dans l'exemple suivant, nous obtiendrons : $CE = AB$ et $(AB) \parallel (CE)$. Pour le point D : $CD = .5AB$ et $(AB) \parallel (CD)$

```

\begin{tkzelements}
  z.A = point: new (0 , 0)
  z.B = point: new (4 , 0)
  z.C = point: new (1 , 3)
  L.AB = line : new (z.A,z.B)
  z.D = L.AB : colinear_at (z.C, .5)
  z.E = L.AB : colinear_at (z.C)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegments(A,B C,E)
  \tkzDrawPoints(A,B,C,D,E)
  \tkzLabelPoints(A,B,C,D,E)
\end{tikzpicture}

```



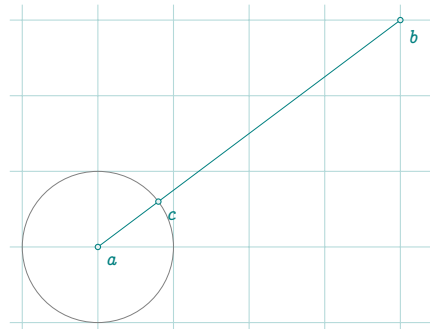
9.2.8 Méthode `normalize`

```

\begin{tkzelements}
  z.a = point: new (1, 1)
  z.b = point: new (5, 4)
  L.ab = line : new (z.a,z.b)
  z.c = L.ab : normalize ()
\end{tkzelements}

\begin{tikzpicture}[gridded]
\tkzGetNodes
\tkzDrawSegments(a,b)
\tkzDrawCircle(a,c)
\tkzDrawPoints(a,b,c)
\tkzLabelPoints(a,b,c)
\end{tikzpicture}

```

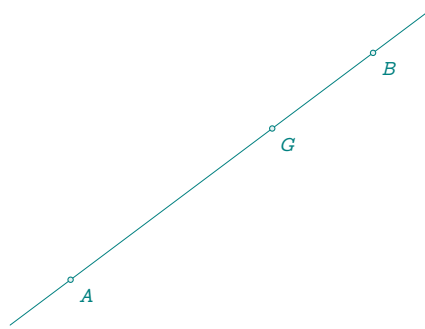


9.2.9 Méthode barycenter

```

\begin{tkzelements}
  z.A = point : new ( 0 , -1 )
  z.B = point : new ( 4 , 2 )
  L.AB = line : new ( z.A , z.B )
  z.G = L.AB : barycenter (1,2)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLine(A,B)
  \tkzDrawPoints(A,B,G)
  \tkzLabelPoints(A,B,G)
\end{tikzpicture}

```

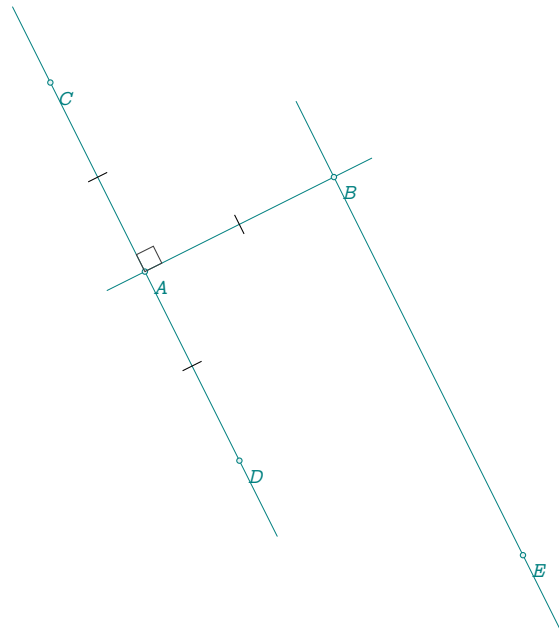


9.2.10 Méthode ll_from

```

\begin{tkzelements}
  scale = 1.25
  z.A = point : new (1,1)
  z.B = point : new (3,2)
  L.AB = line : new (z.A,z.B)
  z.C = L.AB.north_pa
  z.D = L.AB.south_pa
  L.CD = line : new (z.C,z.D)
  _,z.E = get_points ( L.CD: ll_from (z.B))
  -- z.E = L2.pb
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(A,B C,D B,E)
  \tkzDrawPoints(A,...,E)
  \tkzLabelPoints(A,...,E)
  \tkzMarkRightAngle(B,A,C)
  \tkzMarkSegments(A,C A,B A,D)
\end{tikzpicture}

```

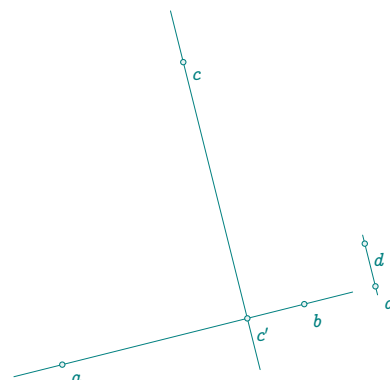


9.2.11 Méthode projection

```

\begin{tkzelements}
  scale = .8
  z.a = point: new (0, 0)
  z.b = point: new (4, 1)
  z.c = point: new (2, 5)
  z.d = point: new (5, 2)
  L.ab = line: new (z.a,z.b)
  z.cp,z.dp = L.ab: projection(z.c,z.d)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(a,b c,c' d,d')
  \tkzDrawPoints(a,...,d,c',d')
  \tkzLabelPoints(a,...,d,c',d')
\end{tikzpicture}

```

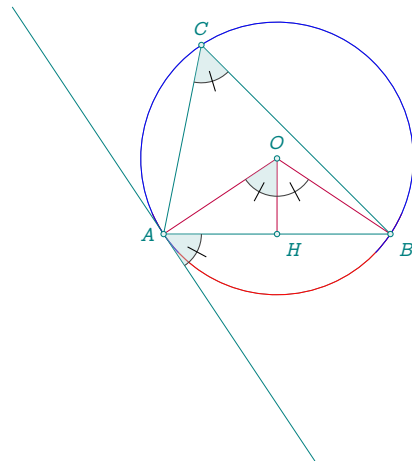


9.2.12 Example: combination of methods

```

\begin{tkzelements}
  z.A      = point: new (0 , 0)
  z.B      = point: new (6 , 0)
  z.C      = point: new (1 , 5)
  T.ABC    = triangle: new (z.A,z.B,z.C)
  L.AB     = T.ABC.ab
  z.O      = T.ABC.circumcenter
  C.OA     = circle: new (z.O,z.A)
  z.H      = L.AB: projection (z.O)
  L.ab     = C.OA: tangent_at (z.A)
  z.a,z.b  = L.ab.pa,L.ab.pb
  -- or z.a,z.b = get_points (L.ab)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawCircle(O,A)
  \tkzDrawSegments[purple](O,A O,B O,H)
  \tkzDrawArc[red](O,A)(B)
  \tkzDrawArc[blue](O,B)(A)
  \tkzDrawLine[add = 2 and 1](A,a)
  \tkzFillAngles[teal!30,opacity=.4](A,C,B b,A,B A,O,H)
  \tkzMarkAngles[mark=|](A,C,B b,A,B A,O,H H,O,B)
  \tkzDrawPoints(A,B,C,H,O)
  \tkzLabelPoints(B,H)
  \tkzLabelPoints[above](O,C)
  \tkzLabelPoints[left](A)
\end{tikzpicture}

```

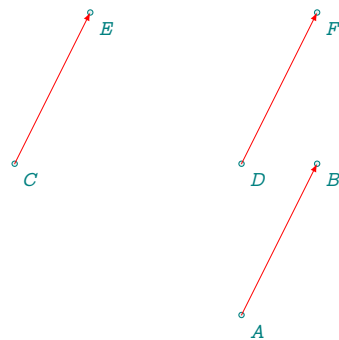


9.2.13 Méthode translation

```

\begin{tkzelements}
  z.A = point: new (0,0)
  z.B = point: new (1,2)
  z.C = point: new (-3,2)
  z.D = point: new (0,2)
  L.AB = line : new (z.A,z.B)
  z.E,z.F = L.AB : translation (z.C,z.D)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPoints(A,...,F)
  \tkzLabelPoints(A,...,F)
  \tkzDrawSegments[->,red,> =latex](C,E D,F A,B)
\end{tikzpicture}

```

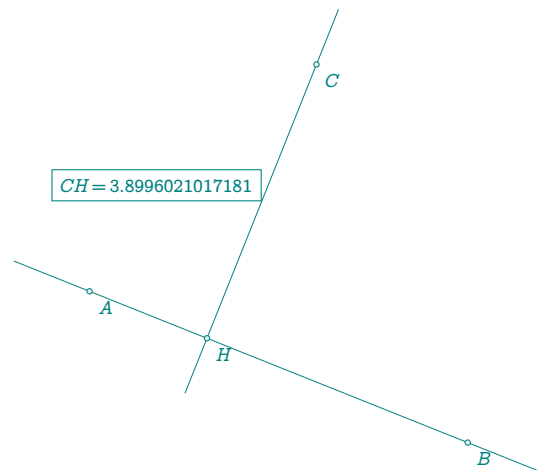


9.2.14 Méthode distance

```

\begin{tkzelements}
  z.A = point : new ( 0 , 0 )
  z.B = point : new ( 5 , -2 )
  z.C = point : new ( 3 , 3 )
  L.AB = line : new ( z.A,z.B )
  d = L.AB : distance ( z.C )
  z.H = L.AB : projection ( z.C )
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(A,B C,H)
  \tkzDrawPoints(A,B,C,H)
  \tkzLabelPoints(A,B,C,H)
  \tkzLabelSegment[above left,
  draw](C,H){$CH = \tkzUseLua{d}$}
\end{tikzpicture}

```

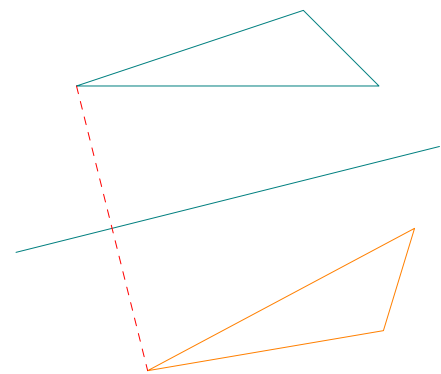


9.2.15 Méthode reflection

```

\begin{tkzelements}
  z.A = point : new ( 0 , 0 )
  z.B = point : new ( 4 , 1 )
  z.E = point : new ( 0 , 2 )
  z.F = point : new ( 3 , 3 )
  z.G = point : new ( 4 , 2 )
  L.AB = line : new ( z.A , z.B )
  T.EFG = triangle : new ( z.E,z.F,z.G )
  T.new = L.AB : reflection ( T.EFG )
  z.Ep,z.Fp,z.Gp = get_points(T.new)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLine(A,B)
  \tkzDrawPolygon(E,F,G)
  \tkzDrawPolygon[new](E',F',G')
  \tkzDrawSegment[red,dashed](E,E')
\end{tikzpicture}

```

9.3 Méthode apollonius Apollonius circle $MA/MB = k$

```

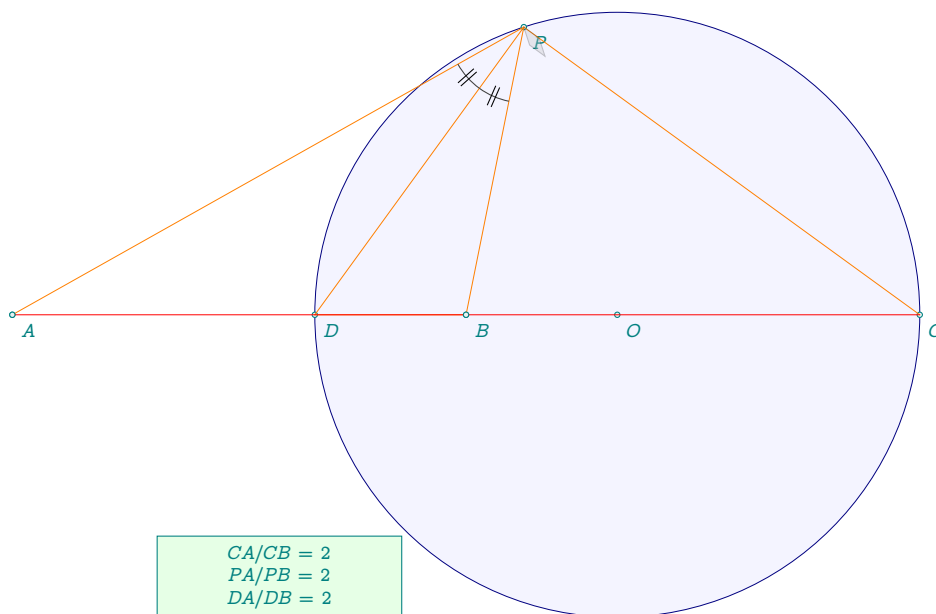
\begin{tkzelements}
  z.A = point : new ( 0 , 0 )
  z.B = point : new ( 6 , 0 )
  L.AB =line: new ( z.A,z.B )
  C.apo = L.AB : apollonius (2)
  z.O,z.C = get_points ( C.apo )
  z.D = C.apo : antipode ( z.C )
  z.P = C.apo : point ( 0.30 )
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzFillCircle[blue!20,opacity=.2](O,C)
  \tkzDrawCircle[blue!50!black](O,C)

```

```

\tkzDrawPoints(A,B,O,C,D,P)
\tkzLabelPoints[below right](A,B,O,C,D,P)
\tkzDrawSegments[orange](P,A P,B P,D B,D P,C)
\tkzDrawSegments[red](A,C)
\tkzDrawPoints(A,B)
\tkzLabelCircle[draw,fill=green!10,%
  text width=3cm,text centered,left=24pt](O,D)(60)%
  {$CA/CB=2$\\$PA/PB=2$\\$DA/DB=2$}
\tkzMarkRightAngle[opacity=.3,fill=lightgray](O,P,C)
\tkzMarkAngles[mark=||](A,P,D D,P,B)
\end{tikzpicture}

```



Remarque : $\text{\tkzUseLua}\{\text{length}(z.P, z.A) / \text{length}(z.P, z.B)\} = 2.0$

10 Classe circle

10.1 Attributs d'un cercle

Cette classe est définie par deux points : le centre et un point par lequel le cercle passe

```
Creation C.OA = circle: new (z.O,z.A)
```

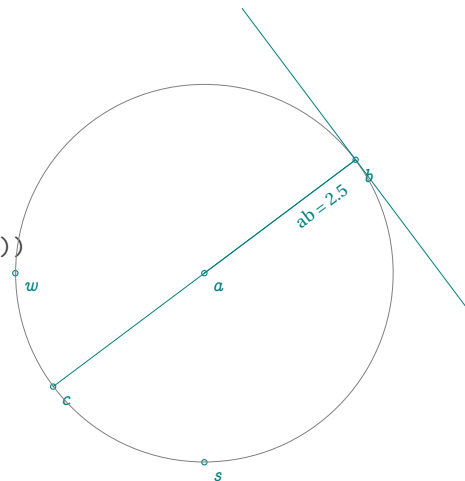
TABLE 7 – Attributs du cercle.

Attributs	Application	
center	<code>z.A = C.AB.center</code>	
through	<code>z.B = C.AB.through</code>	
type	<code>C.AB.type</code>	<code>C.OA.type = 'circle'</code>
radius	<code>C.AB.radius</code>	<code>r = C.OA.radius</code> r nombre réel
north	<code>C.AB.north</code>	<code>z.N = C.OA.north</code>
south	<code>C.AB.south</code>	<code>z.S = C.OA.south</code>
east	<code>C.AB.east</code>	<code>z.E = C.OA.east</code>
west	<code>C.AB.west</code>	<code>z.W = C.OA.west</code>
opp	<code>z.Ap = C.AB.opp</code>	Se référer à (10.1.1)
ct	<code>L = C.AB.ct</code>	Se référer à (10.1.1)

10.1.1 Exemple : attributs du cercle

Trois attributs sont utilisés (south, west, radius).

```
\begin{tkzelements}
  scale = .5
  z.a = point: new (1, 1)
  z.b = point: new (5, 4)
  C.ab = circle : new (z.a,z.b)
  z.s = C.ab.south
  z.w = C.ab.west
  r = C.ab.radius
  z.c = C.ab.opp
  z.r,z.t = get_points (C.ab.ct : ortho_from (z.b))
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(a,b,c,s,w)
\tkzLabelPoints(a,b,c,s,w)
\tkzDrawCircle(a,b)
\tkzDrawSegments(a,b r,t b,c)
\tkzLabelSegment[sloped] (a,b){ab = \tkzUseLua{r}}
\end{tikzpicture}
```



10.2 Méthodes de la classe circle

TABLE 8 – Méthodes du cercle.

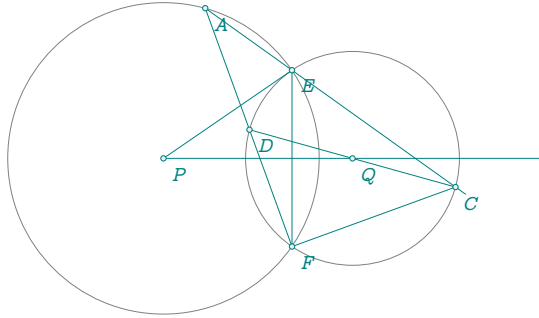
Methods	Comments	
new(O,A)	C.OA = circle : new (z.O,z.A)	circle center O through A
radius(O,r)	C.OA = circle : radius (z.O,2)	circle center O rayon =2 cm
diameter(A,B)	C.OA = circle :diameter(z.A,z.B)	circle diameter $[AB]$
Points		
antipode (pt)	z.C = C.OA: antipode (z.B)	$[BC]$ est un diamètre
inversion (pt)	z.Bp = C.AC: inversion (z.B)	
midarc (pt,pt)	z.D = C.AB: midarc (z.B,z.C)	D est le milieu de l'arc \widehat{BC}
point (r)	z.E = C.AB: point (0.25)	r entre 0 et 1
random_pt(lower, upper)		
internal_similitude (C)	z.I = C.one : internal_similitude (C.two)	
external_similitude (C)	z.J = C.one : external_similitude (C.two)	
radical_center (C1<,C2>)	or only (C1)	Se référer à 23.30
Lines		
radical_axis (C)	Se référer à (23.2; 23.26; 23.27; 23.28; 23.29)	
tangent_at (pt)	z.P = C.OA: tangent_at (z.M)	Se référer à (10.2.2; 9.2.12)
tangent_from (pt)	z.M,z.N = C.OA: tangent_from (z.P)	Refer to ((iii))
inversion (line)	L or C=C.AC:inversion (L.EF)	Refer to (10.2.5)
common_tangent (C)	z.a,z.b = C.AC: common_tangent (C.EF)	Refer to (10.4; 10.5)
Cercles		
orthogonal_from (pt)	C=C.OA:orthogonal_from (z.P)	Se référer à (10.2.1; 10.5; 23.36; 23.40)
orthogonal_through (pta,ptb)	C=C.OA:orthogonal_through (z.z1,z.z2)	Refer to (23.10)
inversion (...)	C.AC:inversion(pt, pts,L or C)	Refer to 10.2.3, 10.2.4, 10.2.5, 10.2.6
midcircle (C)	C.inv=C.OA: midcircle (C.EF)	Se référer à 10.2.7
radical_circle (C1<,C2>)	or only (C1)	Se référer à 23.31
Divers		
power (pt)	p = C.OA: power (z.M)	Se référer à (23.42; 23.43; 23.34)
in_out (pt)	C.OA : in_out (z.M)	Se référer à (10.6)
in_out_disk (pt)	C.OA : in_out_disk (z.M)	Refer to (10.6)
draw ()	pour une utilisation ultérieure	
circles_position (C1)	result = chaîne de caractères	Se référer à (10.3)

10.2.1 Altshiller

```

\begin{tkzelements}
z.P = point : new (0,0)
z.Q = point : new (5,0)
z.I = point : new (3,2)
C.QI = circle : new (z.Q,z.I)
C.PE = C.QI : orthogonal_from (z.P)
z.E = C.PE.through
C.QE = circle : new (z.Q,z.E)
_,z.F = intersection (C.PE,C.QE)
z.A = C.PE: point (1/9)
L.AE = line : new (z.A,z.E)
_,z.C = intersection (L.AE,C.QE)
L.AF = line : new (z.A,z.F)
L.CQ = line : new (z.C,z.Q)
z.D = intersection (L.AF,L.CQ)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(P,E Q,E)
\tkzDrawLines[add=0 and 1](P,Q)
\tkzDrawLines[add=0 and 2](A,E)
\tkzDrawSegments(P,E E,F F,C A,F C,D)
\tkzDrawPoints(P,Q,E,F,A,C,D)
\tkzLabelPoints(P,Q,E,F,A,C,D)
\end{tikzpicture}

```

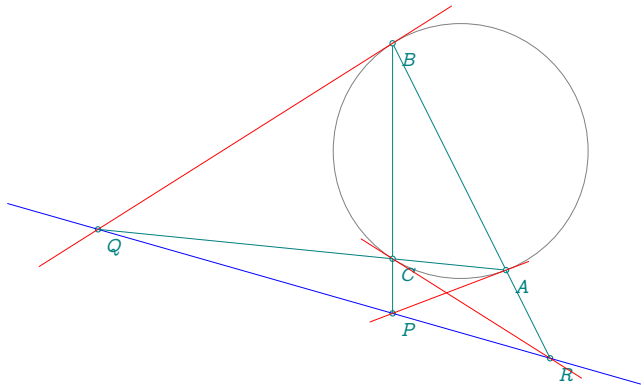


10.2.2 Lemoine

```

\begin{tkzelements}
scale = 1.6
z.A = point: new (1,0)
z.B = point: new (5,2)
z.C = point: new (1.2,2)
T = triangle: new(z.A,z.B,z.C)
z.O = T.circumcenter
C.OA = circle: new (z.O,z.A)
L.tA = C.OA: tangent_at (z.A)
L.tB = C.OA: tangent_at (z.B)
L.tC = C.OA: tangent_at (z.C)
z.P = intersection (L.tA,T.bc)
z.Q = intersection (L.tB,T.ca)
z.R = intersection (L.tC,T.ab)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon[teal](A,B,C)
\tkzDrawCircle(O,A)
\tkzDrawPoints(A,B,C,P,Q,R)
\tkzLabelPoints(A,B,C,P,Q,R)
\tkzDrawLine[blue](Q,R)
\tkzDrawLines[red](A,P B,Q R,C)
\tkzDrawSegments(A,R C,P C,Q)
\end{tikzpicture}

```



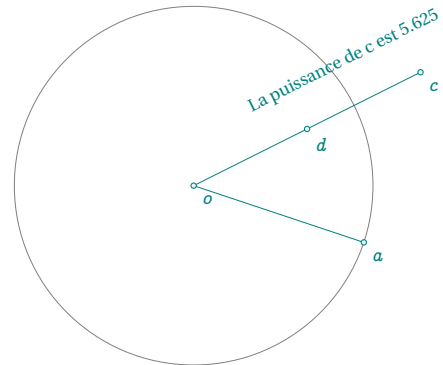
10.2.3 Inversion: point, droite et cercle

La méthode inversion peut être utilisée sur un point, une ligne ou un cercle. Selon le type d'objet, la fonction détermine l'algorithme correct à utiliser.

10.2.4 Inversion: point

La méthode inversion peut être utilisée sur un point, un groupe de points, une ligne ou un cercle. Selon le type d'objet, la fonction détermine l'algorithme correct à utiliser.

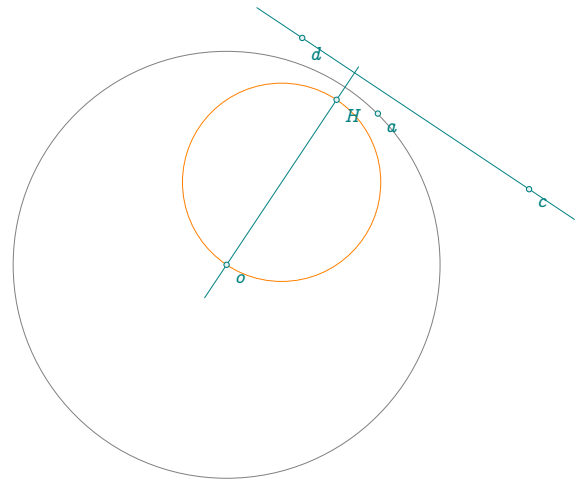
```
\begin{tkzelements}
  z.o = point:    new (-1,2)
  z.a = point:    new (2,1)
  C.oa = circle:  new (z.o,z.a)
  z.c = point:    new (3,4)
  z.d = C.oa:     inversion (z.c)
  p    = C.oa:    power (z.c)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircle(o,a)
  \tkzDrawSegments(o,a o,c)
  \tkzDrawPoints(a,o,c,d)
  \tkzLabelPoints(a,o,c,d)
  \tkzLabelSegment[sloped,above=1em](c,d){%
    Power of c with respect to C is \tkzUseLua{p}}
\end{tikzpicture}
```



10.2.5 Inversion: droite

Le résultat est soit une droite, soit un cercle.

```
\begin{tkzelements}
  z.o = point:    new (-1,1)
  z.a = point:    new (1,3)
  C.oa = circle:  new (z.o,z.a)
  z.c = point:    new (3,2)
  z.d = point:    new (0,4)
  L.cd = line:    new (z.c,z.d)
  C.OH = C.oa:   inversion (L.cd)
  z.O,z.H = get_points(C.OH)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles(o,a O,H)
  \tkzDrawLines(c,d o,H)
  \tkzDrawPoints(a,o,c,d,H)
  \tkzLabelPoints(a,o,c,d,H)
\end{tikzpicture}
```



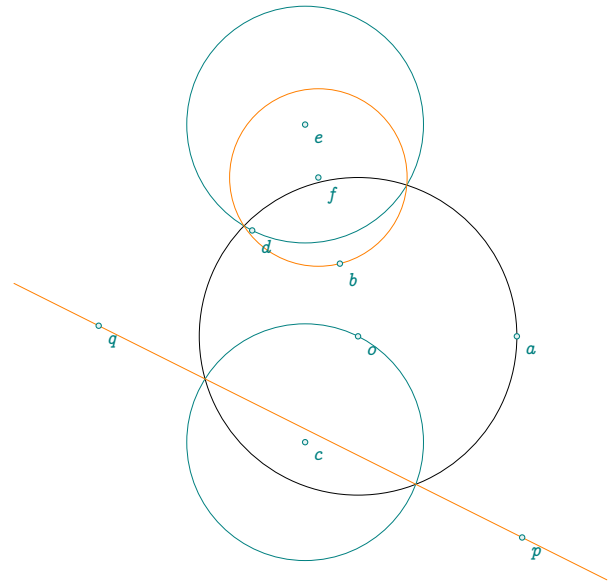
10.2.6 Inversion: cercle

Le résultat est soit une droite, soit un cercle.

```

\begin{tkzelements}
scale = .7
z.o,z.a = point: new (-1,3),point: new (2,3)
z.c      = point: new (-2,1)
z.e,z.d = point: new (-2,7),point: new (-3,5)
C.oa     = circle: new (z.o,z.a)
C.ed     = circle: new (z.e,z.d)
C.co     = circle: new (z.c,z.o)
obj      = C.oa: inversion (C.co)
  if obj.type == "line"
  then z.p,z.q = get_points(obj)
  else z.f,z.b = get_points(obj) end
obj      = C.oa: inversion(C.ed)
if obj.type == "line"
then z.p,z.q = get_points(obj)
else z.f,z.b = get_points(obj) end
color = "orange"
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles[black](o,a)
\tkzDrawCircles[teal](c,o e,d)
\tkzDrawCircles[\tkzUseLua{color}](f,b)
\tkzDrawLines[\tkzUseLua{color}](p,q)
\tkzDrawPoints(a,...,f,o,p,q)
\tkzLabelPoints(a,...,f,o,p,q)
\end{tikzpicture}

```



10.2.7 Midcircle

D'après Eric Danneels et Floor van Lamoen : Un *midcircle* de deux cercles donnés est un cercle qui échange les deux cercles donnés par inversion. Les *midcircles* sont dans le même crayon de cercles que les cercles donnés. Le centre du ou des *midcircles* est l'un ou les deux centres de similitude. Nous pouvons distinguer quatre cas :

- (i) Les deux cercles donnés se croisent : il y a deux *midcircles* avec des centres aux centres de similitude des cercles donnés;
- (ii) Un des cercles donnés est à l'intérieur de l'autre cercle donné. Alors il y a un *midcircle* avec un centre de similitude à l'intérieur du centre de similitude des cercles donnés;
- (iii) Un des cercles donnés est à l'extérieur de l'autre cercle donné. Alors il y a un *midcircle* avec un centre au centre de similitude externe des cercles donnés. De toute évidence, les cas de tangence peuvent être considérés comme des cas limites des cas ci-dessus;
- (iv) Si les cercles se croisent en un seul point, le *midcircle* unique a un centre au centre de similitude externe ou au centre de similitude interne.

Examinons chacun de ces cas :

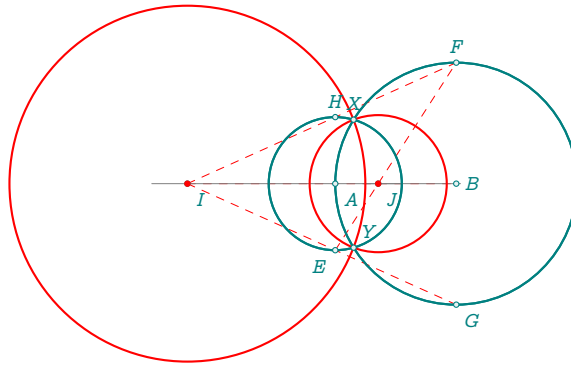
- (i) Si les deux cercles donnés se croisent, alors il existe deux cercles d'inversion passant par leurs points communs, avec des centres aux centres de similitude. Les deux *midcircles* bissectent leurs angles et sont orthogonaux l'un à l'autre. Les centres des *midcircles* sont le centre de similitude interne et le centre de similitude externe I et J .

Considérons deux cercles qui se croisent (\mathcal{A}) et (\mathcal{B}). Nous pouvons obtenir les centres de similitude de ces deux cercles en construisant EH et FG deux diamètres parallèles des cercles (\mathcal{A}) et (\mathcal{B}). La ligne (GE) intercepte la ligne (AB) en J et la ligne (EF) intercepte la ligne (AB) en I . Les cercles (\mathcal{I}) et (\mathcal{J}) sont orthogonaux et sont les *midcircles* de (\mathcal{A}) et (\mathcal{B}). La division ($A, B; I, J$) est harmonique.

```

\begin{tkzelements}
scale = .8
z.A = point : new ( 1 , 0 )
z.B = point : new ( 3 , 0 )
z.O = point : new ( 2.1, 0 )
z.P = point : new ( 1 , 0 )
C.AO = circle : new (z.A,z.O)
C.BP = circle : new (z.B,z.P)
z.E = C.AO.south
z.H = C.AO.north
z.F = C.BP.north
z.G = C.BP.south
C.IT,C.JV = C.AO : midcircle (C.BP)
z.I,z.T = get_points ( C.IT )
z.J,z.V = get_points ( C.JV )
z.X,z.Y = intersection (C.AO,C.BP)
\end{tkzelements}

```

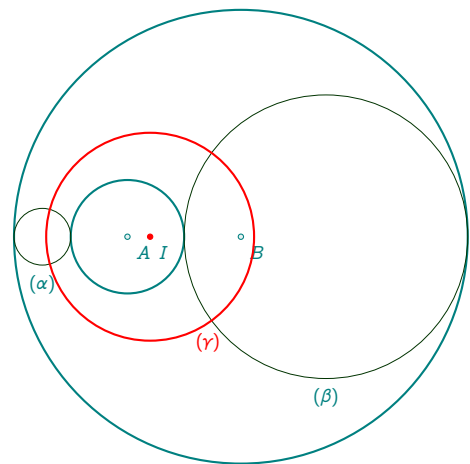


- (ii) Un cercle donné se trouve à l'intérieur de l'autre cercle donné.

```

\begin{tkzelements}
scale = .75
z.A = point : new ( 3 , 0 )
z.B = point : new ( 5 , 0 )
z.O = point : new ( 2 , 0 )
z.P = point : new ( 1 , 0 )
L.AB = line : new (z.A,z.B)
C.AO = circle : new (z.A,z.O)
C.BP = circle : new (z.B,z.P)
z.R,z.S = intersection (L.AB,C.BP)
z.U,z.V = intersection (L.AB,C.AO)
C.SV = circle : diameter (z.S,z.V)
C.UR = circle : diameter (z.U,z.R)
z.x = C.SV.center
z.y = C.UR.center
C.IT = C.AO : midcircle (C.BP)
z.I,z.T = get_points ( C.IT )
\end{tkzelements}

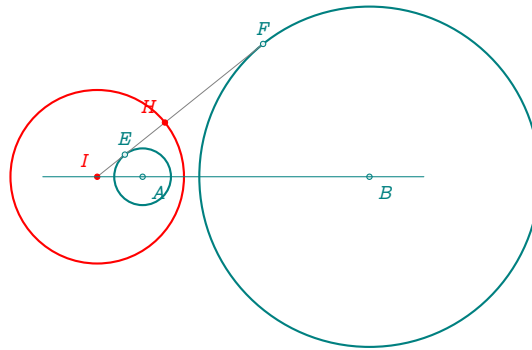
```



Ce cas est un peu plus compliqué. Nous allons construire les deux cercles (α) et (β) tangents aux deux cercles donnés. Ensuite, nous construisons le cercle radical orthogonal aux cercles (α) et (β) . Son centre est le centre radical ainsi que le centre de similitude interne des cercles de centres A et B .

- (iii) Lorsque les deux cercles donnés sont externes l'un à l'autre, nous construisons le centre de similitude externe des deux cercles donnés. I est le centre de similitude externe des deux cercles donnés. Pour obtenir le cercle d'inversion, il suffit de noter que $IH^2 = IE \times IF$.

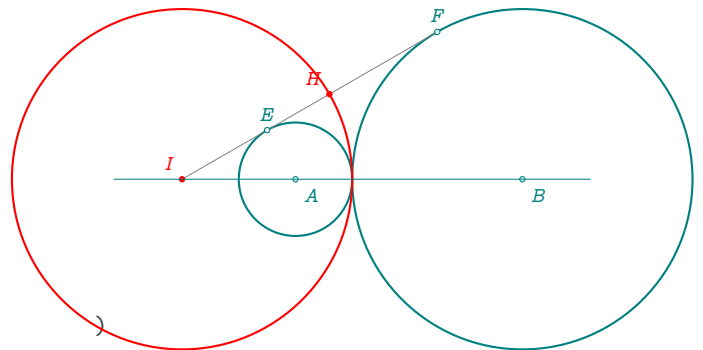
```
\begin{tkzelements}
scale=.75
local a,b,c,d
z.A = point : new ( 0 , 0 )
z.B = point : new ( 4 , 0 )
z.a = point : new ( .5 , 0 )
z.b = point : new ( 1 , 0 )
C.Aa = circle : new (z.A,z.a)
C.Bb = circle : new (z.B,z.b)
L.AB = line : new (z.A,z.B)
z.E = C.Aa.north
z.F = C.Bb.north
L.EF = line : new (z.E,z.F)
C.IT = C.Aa : midcircle (C.Bb)
z.I,z.T = get_points ( C.IT )
L.TF = C.Bb : tangent_from (z.I)
z.H = intersection (L.TF,C.IT)
z.E = intersection (L.TF,C.Aa)
z.F=L.TF.pb
\end{tkzelements}
```



- (iv) Considérons deux cercles tangents (\mathcal{A}) et (\mathcal{B}),

— (\mathcal{B}) étant externe et tangent à (\mathcal{A}). La construction est identique à la précédente.

```
\begin{tkzelements}
scale=.75
local a,b,c,d
z.A = point : new ( 0 , 0 )
z.B = point : new ( 4 , 0 )
z.a = point : new ( 1 , 0 )
z.b = point : new ( 1 , 0 )
C.Aa = circle : new (z.A,z.a)
C.Bb = circle : new (z.B,z.b)
L.AB = line : new (z.A,z.B)
z.E = C.Aa.north
z.F = C.Bb.north
L.EF = line : new (z.E,z.F)
C.IT = C.Aa : midcircle (C.Bb)
z.I,z.T = get_points ( C.IT )
L.TF = C.Bb : tangent_from (z.I)
z.H = intersection (L.TF,C.IT)
z.E = intersection (L.TF,C.Aa)
z.F=L.TF.pb
\end{tkzelements}
```

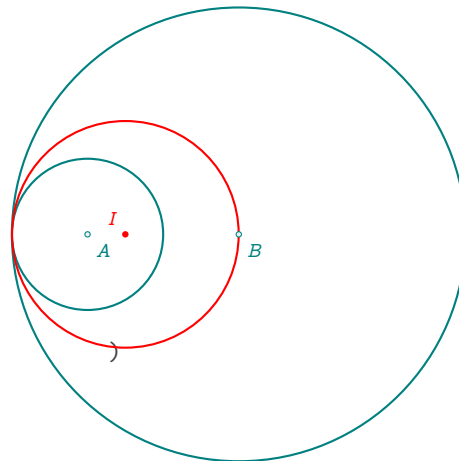


— When one of the given circles is inside and tangent to the other, the construction is easy.

```

\begin{tkzelements}
z.A      = point : new ( 2 , 0 )
z.B      = point : new ( 4 , 0 )
z.a      = point : new ( 1 , 0 )
z.b      = point : new ( 1 , 0 )
C.Aa     = circle : new ( z.A,z.a)
C.Bb     = circle : new ( z.B,z.b)
C.IT     = C.Aa : midcircle (C.Bb)
z.I,z.T  = get_points (      C.IT
\end{tkzelements}

```



10.3 Méthode Circles_position

Cette fonction renvoie une chaîne de caractères indiquant la position du cercle par rapport à un autre. Utile pour créer une fonction. Les cas sont les suivants :

- outside
- outside tangent
- inside tangent
- inside
- intersect

```

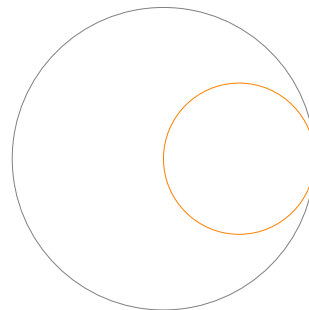
\begin{tkzelements}
z.A      = point : new ( 0 , 0 )
z.a      = point : new ( 3 , 0 )
z.B      = point : new ( 2 , 0 )
z.b      = point : new ( 3 , 0 )
C.Aa     = circle: new ( z.A,z.a)
C.Bb     = circle: new ( z.B,z.b)
position = C.Aa : circles_position (C.Bb)
if position == "inside tangent"
then color = "orange"
else color = "blue" end
\end{tkzelements}

```

```

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircle(A,a)
\tkzDrawCircle[color=\tkzUseLua{color}](B,b)
\end{tikzpicture}

```



10.4 Tangente commune : méthode tangent_common

Soit une tangente commune aux deux cercles en T et T' (plus proche de C). Laissez passer une sécante parallèle à cette tangente par C . Ensuite, le segment $[TT']$ est vu depuis l'autre point commun D à un angle égal à la moitié de l'angle des deux cercles.

```

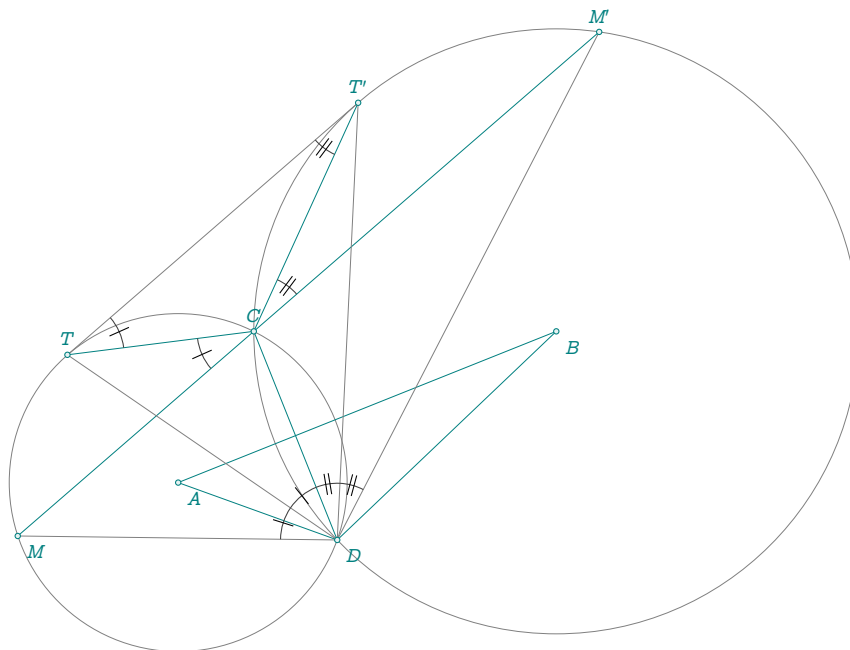
\begin{tkzelements}
z.A      = point : new ( 0 , 0 )
z.B      = point : new ( 5 , 2 )
L.AB     = line : new ( z.A , z.B )

```

```

z.C = point : new ( 1 , 2 )
C.AC = circle : new (z.A,z.C)
C.BC = circle : new (z.B,z.C)
z.T,z.Tp = C.AC : common_tangent (C.BC)
L.TTp = line : new (z.T,z.Tp)
z.M = C.AC : point (0.45)
L.MC =line : new (z.M,z.C)
z.Mp = intersection (L.MC, C.BC)
L.mm = L.TTp : ll_from (z.C)
_,z.M = intersection (L.mm, C.AC)
z.Mp = intersection (L.mm, C.BC)
_,z.D = intersection (C.AC,C.BC)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(A,C B,C)
\tkzDrawSegments(M,M' A,D B,D A,B C,D T,C T',C)
\tkzDrawSegments[gray](D,M D,M' T,T' D,T D,T')
\tkzDrawPoints(A,B,C,D,M,M',T,T')
\tkzLabelPoints(A,B,D,M)
\tkzLabelPoints[above](C,M',T,T')
\tkzMarkAngles[mark=|,size=.75](T,C,M C,T,T' C,D,T T,D,M)
\tkzMarkAngles[mark=||,size=.75](M',C,T' T,T',C T',D,C M',D,T')
\end{tikzpicture}

```



10.5 Tangente commune : orthogonalité

Pour que deux cercles soient orthogonaux, il faut et il suffit qu'une sécante passant par l'un de leurs points communs soit vue de l'autre point commun à angle droit.

```

\begin{tkzelements}
z.A = point : new ( 0 , 0 )
z.B = point : new ( 4 , 2 )

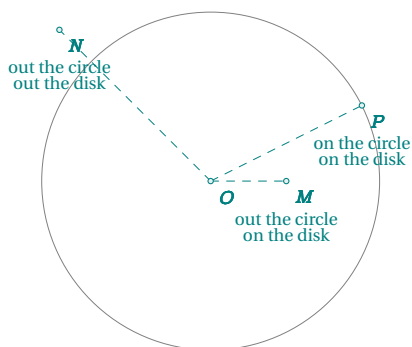
```



```

z.N = point : new (-2,2)
z.M = point : new (1,0)
z.P = point : new (2,1)
BCm = C.OA : in_out (z.M)
BDm = C.OA : in_out_disk (z.M)
BCn = C.OA : in_out (z.N)
BDn = C.OA : in_out_disk (z.N)
BCp = C.OA : in_out (z.P)
BDp = C.OA : in_out_disk (z.P)
\end{tkzelements}
\def\tkzPosPoint#1#2#3#4{%
\tkzLabelPoints(O,M,N,P)
  \ifthenelse{\equal{\tkzUseLua{#1}}{true}}{
    \tkzLabelPoint[below=#4pt,font=\scriptsize](#2){on the #3}}{%
    \tkzLabelPoint[below=#4pt,font=\scriptsize](#2){out the #3}}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawSegments[dashed](O,M O,N O,P)
\tkzDrawCircle(O,A)
\tkzDrawPoints(O,M,N,P)
\tkzPosPoint{BCm}{M}{circle}{8}
\tkzPosPoint{BCn}{N}{circle}{8}
\tkzPosPoint{BCp}{P}{circle}{8}
\tkzPosPoint{BDm}{M}{disk}{14}
\tkzPosPoint{BDn}{N}{disk}{14}
\tkzPosPoint{BDp}{P}{disk}{14}
\end{tikzpicture}

```



11 Classe triangle

11.1 Attributs d'un triangle

L'objet triangle est créé à l'aide de la méthode `new`, par exemple avec

```
Création T.ABC = triangle : new ( z.A , z.B , z.C )
```

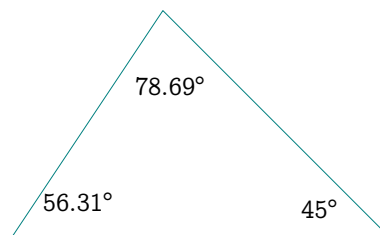
(Se référer à des exemples : 23.3; 23.4; 23.9). Plusieurs attributs sont alors créés.

TABLE 9 – Attributs du triangles.

Attributs	Application
<code>pa</code>	<code>T.ABC.pa</code>
<code>pb</code>	<code>T.ABC.pb</code>
<code>pc</code>	<code>T.ABC.pc</code>
<code>type</code>	'triangle'
<code>circumcenter</code>	<code>T.ABC.circumcenter</code>
<code>centroid</code>	<code>T.ABC.centroid</code>
<code>incenter</code>	<code>T.ABC.incenter</code>
<code>orthocenter</code>	<code>T.ABC.orthocenter</code>
<code>eulercenter</code>	<code>T.ABC.eulercenter</code>
<code>spiekercenter</code>	<code>T.ABC.spiekercenter</code>
<code>a</code>	C'est la longueur du côté opposé au premier sommet.
<code>b</code>	C'est la longueur du côté opposé au deuxième sommet.
<code>c</code>	C'est la longueur du côté opposé au troisième sommet
<code>alpha</code>	Angle du premier sommet
<code>beta</code>	Angle du deuxième sommet
<code>gamma</code>	Angle du troisième sommet
<code>ab</code>	Droite définie par les deux premiers points du triangle
<code>bc</code>	Droite définie par les deux derniers points
<code>ca</code>	Droite définie par le dernier et le premier point du triangle

11.2 Attributs du triangle : angles

```
\begin{tkzelements}
  z.A      = point: new(0,0)
  z.B      = point: new(5,0)
  z.C      = point: new(2,3)
  T.ABC    = triangle: new (z.A,z.B,z.C)
\end{tkzelements}
\def\wangle#1{\tkzDN[2]{%
  \tkzUseLua{math.deg(T.ABC.#1)}}}
\begin{tikzpicture}
\tkzGetNodes
  \tkzDrawPolygons(A,B,C)
  \tkzLabelAngle(B,A,C){$\wangle{\alpha}^{\circ}$}
  \tkzLabelAngle(C,B,A){$\wangle{\beta}^{\circ}$}
  \tkzLabelAngle(A,C,B){$\wangle{\gamma}^{\circ}$}
\end{tikzpicture}
```

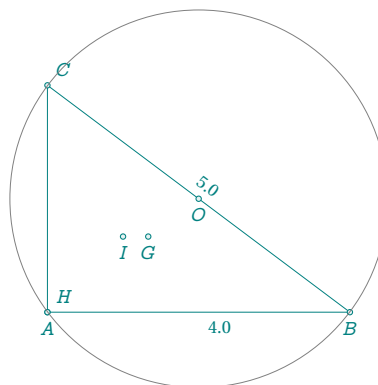


11.2.1 Exemple : attributs du triangle

```

\begin{tkzelements}
  z.A = point: new (0 , 0)
  z.B = point: new (4 , 0)
  z.C = point: new (0 , 3)
  T.ABC = triangle : new (z.A,z.B,z.C)
  z.O = T.ABC.circumcenter
  z.I = T.ABC.incenter
  z.H = T.ABC.orthocenter
  z.G = T.ABC.centroid
  a = T.ABC.a
  b = T.ABC.b
  c = T.ABC.c
  alpha = T.ABC.alpha
  beta = T.ABC.beta
  gamma = T.ABC.gamma
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawPoints(A,B,C,O,G,I,H)
  \tkzLabelPoints[below] (A,B,O,G,I)
  \tkzLabelPoints[above right] (H,C)
  \tkzDrawCircles(O,A)
  \tkzLabelSegment[sloped] (A,B){\tkzUseLua{c}}
  \tkzLabelSegment[sloped,above] (B,C){\tkzUseLua{a}}
\end{tikzpicture}

```



11.3 Methods of the class triangle

TABLE 10 – Méthodes du triangle.

Méthodes	Comments
<code>new (a, b, c)</code>	<code>T.ABC = triangle : new (z.A, z.B, z.C)</code>
...	T oo T. name avec ce que vous voulez comme nom, est possible.
Points	
<code>lemoine_point ()</code>	<code>T.ABC : lemoine_point ()</code> intersection des symédianes
<code>symmedian_point ()</code>	Le point de Lemoine ou le point de Grebe
<code>bevan_point ()</code>	Centre du triangle excentral
<code>mittenpunkt_point ()</code>	Point symédian du triangle excentral
<code>gergonne_point ()</code>	Intersection des trois céviennes qui mènent aux points de contact
<code>nagel_point ()</code>	Intersection des trois céviennes qui mènent aux points de contact
<code>feuerbach_point ()</code>	Le point où l'incircle et le cercle d'Euler sont tangents.
<code>spieker_center ()</code>	Centre du triangle médial
<code>barycenter (ka, kb, kc)</code>	<code>T.ABC: barycenter (2, 1, 1)</code> barycentre de $(\{A, 2\}, \{B, 1\}, \{C, 1\})$
<code>base (u, v)</code>	<code>z.D = T.ABC: base(1, 1) -></code> ABDC est un parallélogramme
<code>projection (p)</code>	Pprojection d'un point sur les côtés
<code>euler_points ()</code>	Points d'Euler du cercle d'Euler
<code>nine_points ()</code>	9 Points du cercle d'Euler
<code>parallelogram ()</code>	<code>z.D = T.ABC : parallelogram () -></code> ABCD est un parallélogramme
<code>conway_points ()</code>	Voir l'exemple (11.3.3)
Lines	
<code>altitude (n)</code>	<code>L.AHa = T.ABC : altitude ()</code> n vide ou 0 ligne de A^a
<code>bisector (n)</code>	<code>L.Bb = T.ABC : bisector (1)</code> n = 1 line from B^b
<code>bisector_ext(n)</code>	n=2 là partir du troisième sommet.
<code>symmedian_line (n)</code>	Céviennes par rapport au point de Lemoine.
<code>euler_line ()</code>	the line through N, G, H and O if the triangle is not equilateral ^c
<code>antiparallel(pt, n)</code>	n=0 antiparallèle par pt à (BC) , n=1 to (AC) etc.
Circles	
<code>euler_circle ()</code>	<code>C.NP = T.ABC : euler_circle () -></code> N euler point ^d
<code>circum_circle ()</code>	<code>C.OA = T.ABC : circum ()</code> Triangle's circumscribed circle
<code>in_circle ()</code>	Cercle inscrit du triangle
<code>ex_circle (n)</code>	Cercle tangent aux trois côtés du triangle; n = 1 swap; n = 2 2 swap
<code>first_lemoine_circle ()</code>	The center is the midpoint between Lemoine point and the circumcenter. ^e
<code>second_lemoine_circle ()</code>	Voir l'exemple 23.58
<code>spieker_circle ()</code>	Le cercle du triangle médian
<code>cevian_circle ()</code>	Cercle circonscrit au triangle cévien Voir l'exemple (11.3.1)
<code>pedal_circle ()</code>	Cercle circonscrit au triangle podaire Voir l'exemple (11.3.2)
<code>conway_circle ()</code>	Cercle circonscrit aux points de Conway Voir l'exemple (11.3.3)

^{a.} `z.Ha = L.AHa.pb` récupère le point commun du côté opposé et de la hauteur. La méthode `orthic` est utile.

^{b.} `_, z.b = get_points(L.Bb)` retrouve le point commun du côté opposé et de la bissectrice.

^{c.} N centre du cercle de neuf points, G centroïde, H orthocentre, O centre du cercle circonscrit.

^{d.} Le milieu de chaque côté du triangle, le pied de chaque altitude, le milieu du segment de droite allant de chaque sommet du triangle à l'orthocentre.

^{e.} En passant par le point de Lemoine, tracez des lignes parallèles aux côtés du triangle. Les points d'intersection des droites parallèles avec les côtés du triangle ABC se trouvent alors sur un cercle appelé premier cercle de Lemoine.

Remarque : Si vous n'avez pas besoin d'utiliser l'objet `triangle` plusieurs fois, vous pouvez obtenir une bissectrice ou une altitude avec les fonctions suivantes

`bisector (z.A, z.B, z.C)` et `altitude (z.A, z.B, z.C)` Refer to (29)

TABLE 11 – Méthodes de la classe triangle (Suite)

Méthodes	Comments
Triangles	
orthic ()	$T = T.ABC$: orthic () triangle joignant les pieds des hauteurs
medial ()	$T = T.ABC$: medial () triangle dont les sommets sont situés aux milieux
incentral ()	Triangle cevien du triangle par rapport à son centre
excentral ()	Triangle dont les sommets correspondent aux excentres ^a
extouch ()	Triangle formé par les points de tangence avec les cercles exinscrits
intouch ()	Triangle de contact formé par les points de tangence de l'incircle
tangential ()	Triangle formé par les lignes tangentes au cercle circonscrit aux sommets
feuerbach ()	Triangle formé par les points de tangence du cercle d'Euler avec les cercles exinscrits
anti ()	Triangle anticomplémentaire Le triangle donné est son triangle médian.
cevian (pt)	Triangle formé par les points finaux des trois céviennes par rapport à pt..
pedal (pt)	Triangle formé par les projections sur les côtés de pt..
symmedian ()	Triangle formé avec les points d'intersection des symédiannes.
euler ()	Triangle formé avec les points d'Euler
Ellipses	
steiner_inellipse ()	Se référer à ex. (11.4.1)
steiner_circumellipse ()	Se référer à ex. (11.4.1)
euler_ellipse ()	Se référer à ex. (11.4)
Divers	
area ()	$A = T.ABC$: area ()
barycentric_coordinates (pt)	Triplet de nombres correspondant à des masses placées aux sommets
in_out (pt)	Booléen. Tester si pt est à l'intérieur du triangle
check_equilateral ()	Booléen. Tester si le triangle est équilatéral

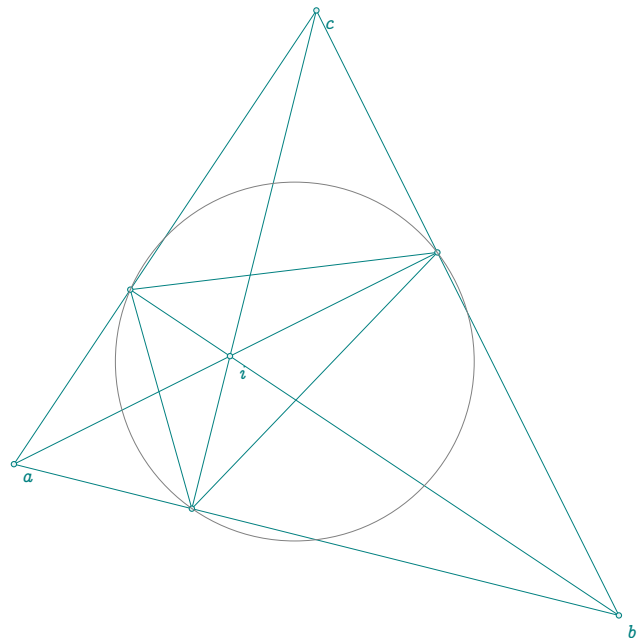
^a. centre des cercle exinscrits

11.3.1 Méthodes cevian et cevian_circle

```

\begin{tkzelements}
  scale = 2
  z.a = point: new (1,2)
  z.b = point: new (5,1)
  z.c = point: new (3,5)
  T = triangle: new (z.a,z.b,z.c)
  z.i = T.orthocenter
  T.cevian = T : cevian (z.i)
  z.ta,z.tb,z.tc = get_points (T.cevian)
  C.cev = T : cevian_circle (z.i)
  z.w = C.cev.center
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygons(a,b,c ta,tb,tc)
\tkzDrawSegments(a,ta b,tb c,tc)
\tkzDrawPoints(a,b,c,i,ta,tb,tc)
\tkzLabelPoints(a,b,c,i)
\tkzDrawCircles(w,ta)
\end{tikzpicture}

```

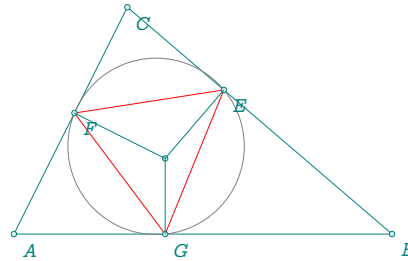


11.3.2 Méthodes pedal et pedal_circle

```

\begin{tkzelements}
  z.A = point: new(0,0)
  z.B = point: new(5,0)
  z.C = point: new(1.5,3)
  z.O = point: new (2,1)
  T.ABC = triangle: new (z.A,z.B,z.C)
  T.pedal = T.ABC : pedal (z.O)
  z.E,z.F,z.G = get_points(T.pedal)
  C.pedal = T.ABC : pedal_circle (z.O)
  z.w = C.pedal.center
  z.T = C.pedal.through
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawPolygon[red](E,F,G)
\tkzDrawCircle(w,T)
\tkzDrawPoints(A,B,C,E,F,G,O)
\tkzLabelPoints(A,B,C,E,F,G)
\tkzDrawSegments(O,E O,F O,G)
\end{tikzpicture}

```



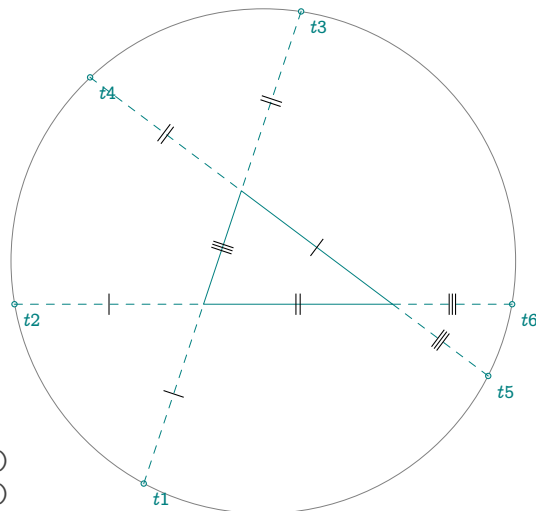
11.3.3 Méthodes conway_points et conway_circle

En géométrie plane, le théorème du cercle de Conway stipule que lorsque les côtés se rencontrant à chaque sommet d'un triangle sont prolongés par la longueur du côté opposé, les six points d'extrémité des trois segments de droite résultants se trouvent sur un cercle dont le centre est le centre d'incidence du triangle.

```

\begin{tkzelements}
  z.A = point:new (0,0)
  z.C = point:new (5,0)
  z.B = point:new (1,3)
  T.ABC = triangle : new (z.A,z.B,z.C)
  C.conway = T.ABC : conway_circle ()
  z.w,z.t = get_points(C.conway)
  z.t1,z.t2,z.t3,z.t4,
  z.t5,z.t6= T.ABC : conway_points ()
\end{tkzelements}
\hspace*{5cm}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawCircles(w,t)
\tkzDrawPoints(t1,t2,t3,t4,t5,t6)
\tkzLabelPoints(t1,t2,t3,t4,t5,t6)
\tkzDrawSegments[dashed](t1,A t2,A t3,B)
\tkzDrawSegments[dashed](t4,B t5,C t6,C)
\tkzMarkSegments(B,C t1,A t2,A)
\tkzMarkSegments[mark=||](A,C t3,B t4,B)
\tkzMarkSegments[mark=|||](A,B t5,C t6,C)
\end{tikzpicture}

```

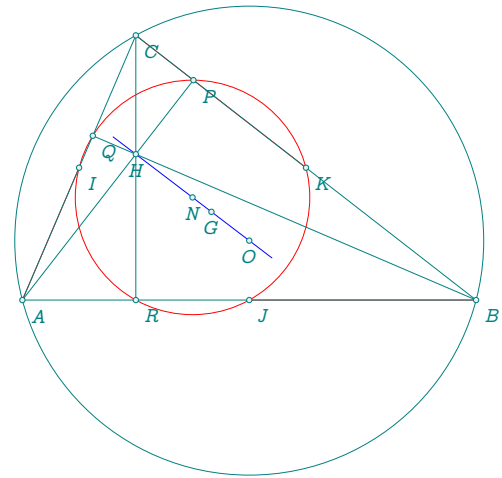


11.3.4 Euler line

```

\begin{tkzelements}
z.A      = point: new (0 , 0)
z.B      = point: new (6 , 0)
z.C      = point: new (1.5 , 3.5)
T.ABC    = triangle: new (z.A,z.B,z.C)
z.O      = T.ABC.circumcenter
z.G      = T.ABC.centroid
z.N      = T.ABC.eulercenter
z.H      = T.ABC.orthocenter
z.P,z.Q,z.R = get_points (T.ABC: orthic())
z.K,z.I,z.J = get_points (T.ABC: medial ())
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawLines[blue] (O,H)
\tkzDrawCircle[red] (N,I)
\tkzDrawCircles[teal] (O,A)
\tkzDrawSegments(A,P B,Q C,R)
\tkzDrawSegments[red] (A,I B,J C,K)
\tkzDrawPolygons(A,B,C)
\tkzDrawPoints(A,B,C,N,I,J,K,O,P,Q,R,H,G)
\tkzLabelPoints(A,B,C,I,J,K,P,Q,R,H)
\tkzLabelPoints[below] (N,O,G)
\end{tikzpicture}

```



11.4 Ellipse d'Euler

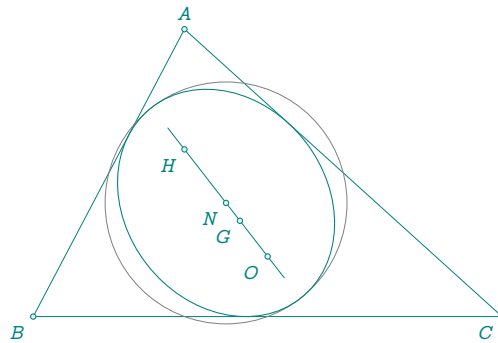
Exemple d'obtention du cercle d'Euler et de l'ellipse d'Euler.

```

\begin{tkzelements}
z.A      = point: new (2,3.8)
z.B      = point: new (0 , 0)
z.C      = point: new (6.2 , 0)
L.AB     = line : new ( z.A , z.B )
T.ABC    = triangle: new (z.A,z.B,z.C)
z.K      = midpoint (z.B,z.C)
E.euler  = T.ABC : euler_ellipse ()
z.N      = T.ABC.eulercenter
C.euler  = circle : new (z.N,z.K)
ang      = math.deg(E.euler.slope)
z.O      = T.ABC.circumcenter
z.G      = T.ABC.centroid
z.H      = T.ABC.orthocenter
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawCircle(N,K)
\tkzDrawEllipse[teal] (N,\tkzUseLua{E.euler.Rx},
\tkzUseLua{E.euler.Ry},\tkzUseLua{ang})
\tkzDrawLine(O,H)
\tkzDrawPoints(A,B,C,N,O,H,G)
\tkzLabelPoints[below left] (B,C,N,O,H,G)
\end{tikzpicture}

```

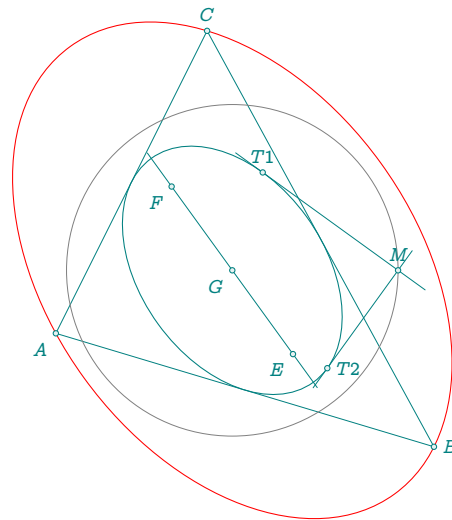


```
\tkzLabelPoints[above](A)
\end{tikzpicture}
```

11.4.1 Inellipse et circumellipse de Steiner

Dans cet exemple, les ellipses de Steiner intérieure et extérieure, appelées respectivement "inellipse" et "circumellipse" (Mathworld.com), ainsi que le cercle orthoptique, sont représentés... Le triangle doit être un acutangle.

```
\begin{tkzelements}
  scale      = .5
  z.A        = point: new (1 , 4)
  z.B        = point: new (11 , 1)
  z.C        = point: new (5 , 12)
  T.ABC      = triangle: new(z.A,z.B,z.C)
  E          = T.ABC: steiner_inellipse ()
  z.G        = E.center
  ang        = math.deg(E.slope)
  z.F        = E.Fa
  z.E        = E.Fb
  C          = E: orthoptic_circle ()
  z.w        = C.center
  z.o        = C.through
  EE         = T.ABC : steiner_circumellipse ()
  z.M        = C : point (Q)
  L.T1,L.T2 = E : tangent_from (z.M)
  z.T1       = L.T1.pb
  z.T2       = L.T2.pb
\end{tkzelements}
```



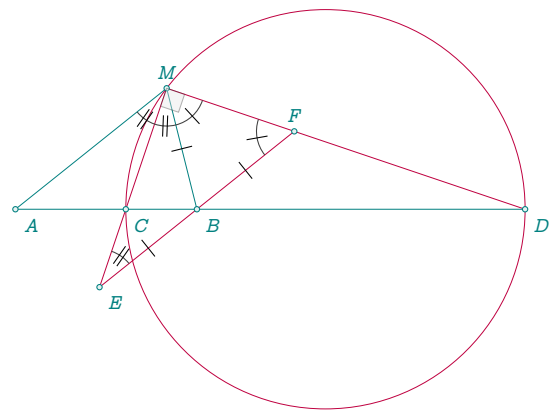
```
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawCircles(w,o)
\tkzDrawEllipse[teal](G,\tkzUseLua{E.Rx},
\tkzUseLua{E.Ry},\tkzUseLua{ang})
\tkzDrawEllipse[red](G,\tkzUseLua{EE.Rx},
\tkzUseLua{EE.Ry},\tkzUseLua{ang})
\tkzDrawLines(F,E M,T1 M,T2) %
\tkzDrawPoints(A,B,C,F,E,G,M,T1,T2)
\tkzLabelPoints[above](C,M,T1)
\tkzLabelPoints[right](T2,B)
\tkzLabelPoints[below left](A,F,E,G)
\end{tikzpicture}
```

11.5 Division harmonique et bissectrice

```

\begin{tkzelements}
  scale      = .4
  z.A        = point: new (0 , 0)
  z.B        = point: new (6 , 0)
  z.M        = point: new (5 , 4)
  T.AMB      = triangle : new (z.A,z.M,z.B)
  L.AB       = T.AMB.ca
  L.bis      = T.AMB : bisector (1)
  z.C        = L.bis.pb
  L.bisext   = T.AMB : bisector_ext (1)
  z.D        = intersection (L.bisext,L.AB)
  L.CD       = line: new (z.C,z.D)
  z.O        = L.CD.mid
  L.AM       = line: new (z.A,z.M)
  L.LL       = L.AM : ll_from (z.B)
  L.MC       = line: new (z.M,z.C)
  L.MD       = line: new (z.M,z.D)
  z.E        = intersection (L.LL,L.MC)
  z.F        = intersection (L.LL,L.MD)
\end{tkzelements}

```



```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,M)
  \tkzDrawCircle[purple](O,C)
  \tkzDrawSegments[purple](M,E M,D E,F)
  \tkzDrawSegments(D,B)
  \tkzDrawPoints(A,B,M,C,D,E,F)
  \tkzLabelPoints[below right](A,B,C,D,E)
  \tkzLabelPoints[above](M,F)
  \tkzMarkRightAngle[opacity=.4,fill=gray!20](C,M,D)
  \tkzMarkAngles[mark=||,size=.5](A,M,E E,M,B B,E,M)
  \tkzMarkAngles[mark=|,size=.5](B,M,F M,F,B)
  \tkzMarkSegments(B,E B,M B,F)
\end{tikzpicture}

```

12 Classe ellipse

12.1 Attributs d'une ellipse

Les premiers attributs sont les trois points qui définissent l'ellipse : le centre de l', le **vertex** et le **covertex**. La première méthode pour définir une ellipse consiste à donner son centre, puis le point nommé **vertex** qui définit le grand axe, et enfin le point nommé **covertex** qui définit le petit axe.

TABLE 12 – Attributs de l'ellipse.

Attribut	Application
center	centre de l'ellipse
vertex	point du grand axe et de l'ellipse
covertex	point du petit axe et de l'ellipse
type	Le type est 'ellipse'
Rx	Rayon entre le centre et le sommet
Ry	Rayon du centre au sommet au covertex
slope	Pente de la droite passant par les foyers
Fa	Premier foyer
Fb	Deuxième foyer
south	Voir l'exemple suivant 12.1.1
north	
west	
east	

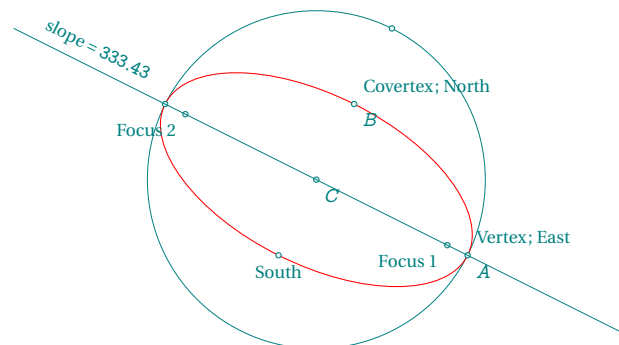
12.1.1 Attributs d'une ellipse : exemple

```

\begin{tkzelements}
  z.C = point: new (3 , 2)
  z.A = point: new (5 , 1)
  L.CA = line : new (z.C,z.A)
  z.b = L.CA.north_pa
  L = line : new (z.C,z.b)
  z.B = L : point (0.5)
  E = ellipse: new (z.C,z.A,z.B)
  a = E.Rx
  b = E.Ry
  z.F1 = E.Fa
  z.F2 = E.Fb
  slope = math.deg(E.slope)
  z.E = E.east
  z.N = E.north
  z.W = E.west
  z.S = E.south
  z.Co = E.covertex
  z.Ve = E.vertex
\end{tkzelements}

\begin{tikzpicture}
  \pgfkeys{/pgf/number format/.cd, fixed, precision=2}
  \tkzGetNodes
  \tkzDrawCircles[teal](C,A)
  \tkzDrawEllipse[red](C,\tkzUseLua{a},\tkzUseLua{b},
  \tkzUseLua{slope})
  \tkzDrawPoints(C,A,B,b,W,S,F1,F2)
  \tkzLabelPoints(C,A,B)
  \tkzDrawLine[add = .5 and .5](A,W)

```



```

\tkzLabelSegment[pos=1.5,above,sloped](A,W){%
  slope = \pgfmathprintnumber{\tkzUseLua{slope}}
\tkzLabelPoint[below](S){South}
\tkzLabelPoint[below left](F1){Focus 1}
\tkzLabelPoint[below left](F2){Focus 2}
\tkzLabelPoint[above right](Ve){Vertex ; East}
\tkzLabelPoint[above right](Co){Covertex ; North}
\end{tikzpicture}

```

12.2 Méthodes de la classe ellipse

Avant d'examiner les méthodes et fonctions liées aux ellipses, jetons un coup d'œil à la façon de dessiner des ellipses avec `tkz-elements`. La macro `\tkzDrawEllipse` nécessite 4 arguments : le centre de l'ellipse, le grand rayon (sur l'axe des foyers), le petit rayon et l'angle formé par l'axe des foyers. Les trois derniers arguments doivent être transférés de `tkzelements` à `tikzpicture`. Pour ce faire, vous devrez utiliser une macro : `\tkzUseLua` définie dans `tkz-elements`. Reportez-vous à 6.1.2 ou 20.6 ou aux exemples suivants.

TABLE 13 – Ellipse methods.

Methods	Example
<code>new (pc, pa ,pb)</code>	<code>E = ellipse : new (center, vertex, covertex)</code>
<code>foci (f1,f2,v)</code>	<code>E = ellipse : foci (focus 1, focus 2, vertex)</code>
<code>radii (c,a,b,sl)</code>	<code>E = ellipse : radii (center, radius a, radius b, slope)</code>
<code>in_out (pt)</code>	pt in/out of the ellipse
<code>tangent_at (pt)</code>	Refer to ex. 9.2.6
<code>tangent_from (pt)</code>	Refer to ex. 9.2.6
<code>point (t)</code>	vertex = point (0) covertex = point (0.25) ex Refer to 9.2.6
<code>orthoptic_circle ()</code>	Refer to ex. 11.4.1

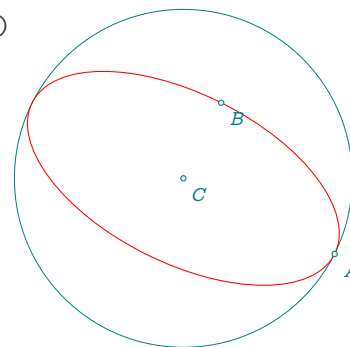
12.2.1 Méthode new

La principale méthode pour créer une nouvelle ellipse est `new`. Les arguments sont au nombre de trois : `center`, `vertex` et `covertex`. Pour les attributs, voir 12.

```

\begin{tkzelements}
  z.C      = point: new (3 , 2)
  z.A      = point: new (5 , 1)
  z.B      = z.C : homothety(0.5,
    z.C : rotation (math.pi/2,z.A))
  E        = ellipse: new (z.C,z.A,z.B)
  a        = E.Rx
  b        = E.Ry
  slope    = math.deg(E.slope)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles[teal](C,A)
  \tkzDrawEllipse[red](C,\tkzUseLua{a},
    \tkzUseLua{b},\tkzUseLua{slope})
  \tkzDrawPoints(C,A,B)
  \tkzLabelPoints(C,A,B)
\end{tikzpicture}

```

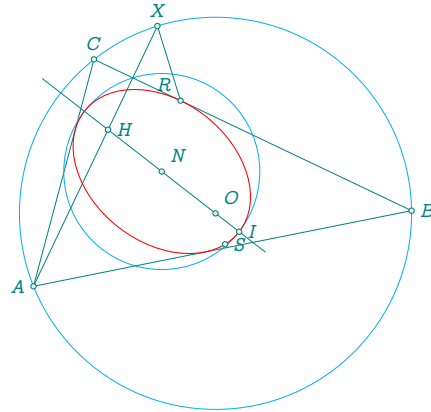


The function `tkzUseLua` (variable) is used to transfer values to `TikZ` or `tkz-euclide`.

12.2.2 Méthode foci

Les deux premiers points sont les foyers de l'ellipse, et le troisième est le sommet. Nous pouvons déduire toutes les autres caractéristiques à partir de ces points. *La fonction lance la méthode new, définissant toutes les caractéristiques de l'ellipse.*

```
\begin{tkzelements}
  z.A      = point: new (0 , 0)
  z.B      = point: new (5 , 1)
  L.AB     = line : new (z.A,z.B)
  z.C      = point: new (.8 , 3)
  T.ABC    = triangle: new (z.A,z.B,z.C)
  z.N      = T.ABC.eulercenter
  z.H      = T.ABC.orthocenter
  z.O      = T.ABC.circumcenter
  _,_,z.Mc = get_points (T.ABC: medial ())
  L.euler  = line: new (z.H,z.O)
  C.circum = circle: new (z.O,z.A)
  C.euler  = circle: new (z.N,z.Mc)
  z.i,z.j  = intersection (L.euler,C.circum)
  z.I,z.J  = intersection (L.euler,C.euler)
  E        = ellipse: foci (z.H,z.O,z.I)
  L.AH     = line: new (z.A,z.H)
  z.X      = intersection (L.AH,C.circum)
  L.XO     = line: new (z.X,z.O)
  z.R,z.S  = intersection (L.XO,E)
  a,b      = E.Rx,E.Ry
  ang      = math.deg(E.slope)
\end{tkzelements}
```



```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawCircles[cyan] (O,A N,I)
  \tkzDrawSegments(X,R A,X)
  \tkzDrawEllipse[red] (N,\tkzUseLua{a},
    \tkzUseLua{b},\tkzUseLua{ang})
  \tkzDrawLines[add=.2 and .5] (I,H)
  \tkzDrawPoints(A,B,C,N,O,X,H,R,S,I)
  \tkzLabelPoints[above] (C,X)
  \tkzLabelPoints[above right] (N,O)
  \tkzLabelPoints[above left] (R)
  \tkzLabelPoints[left] (A)
  \tkzLabelPoints[right] (B,I,S,H)
\end{tikzpicture}
```

12.2.3 Méthode point and radii

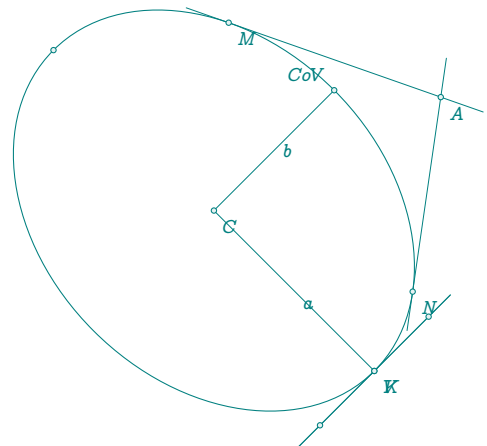
La méthode point définit un point M de l'ellipse dont les coordonnées sont $(a \times \cos(\phi), b \times \sin(\phi))$. ϕ est l'angle entre (centre, vertex) et (centre, M).

L'environnement **tkzelements** utilise en tant que **lua** le radian comme unité pour les angles.


```

\begin{tkzelements}
  z.C      = point: new (2 , 3)
  z.A      = point: new (6 , 5)
  a        = value(4)
  b        = value(3)
  ang      = math.deg(-math.pi/4)
  E        = ellipse: radii (z.C,a,b,-math.pi/4)
  z.V      = E : point (0)
  z.K      = E : point (1)
  z.CoV    = E : point (0.25)
  z.X      = E : point (0.5)
  L        = E :tangent_at (z.V)
  z.x,z.y  = get_points(L)
  L.ta,L.tb = E :tangent_from (z.A)
  z.M      = L.ta.pb
  z.N      = L.tb.pb
  L.K      = E :tangent_at (z.K)
  z.ka,z.kb = get_points(L.K)
\end{tkzelements}

```



```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegments(C,V C,CoV)
  \tkzDrawLines(x,y A,M A,N ka,kb)
  \tkzLabelSegment(C,V){$a$}
  \tkzLabelSegment[right](C,CoV){$b$}
  \tkzDrawEllipse[teal](C,\tkzUseLua{a},\tkzUseLua{b},\tkzUseLua{ang})
  \tkzDrawPoints(C,V,CoV,X,x,y,M,N,A,K)
  \tkzLabelPoints(C,V,A,M,N,K)
  \tkzLabelPoints[above left](CoV)
\end{tikzpicture}

```

13 Classe Quadrilateral

13.1 Attributs du quadrilatère

Points are created in the direct direction. A test is performed to check whether the points form a rectangle, otherwise compilation is blocked.

```
Creation Q.new = rectangle : new (z.A,z.B,z.C,z.D)
```

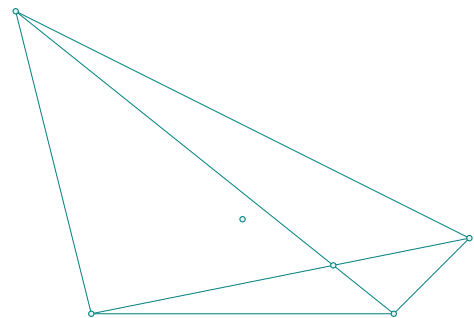
TABLE 14 – Attributs du rectangle.

Attributs	Application	
pa	z.A = Q.new.pa	
pb	z.B = Q.new.pb	
pc	z.C = Q.new.pc	
pd	z.D = Q.new.pd	
type	Q.new.type= 'quadrilateral'	
i	z.I = Q.new.i	intersection of diagonals
g	z.G = Q.new.g	barycenter
a	AB = Q.new.a	barycenter
b	BC = Q.new.b	barycenter
c	CD = Q.new.c	barycenter
d	DA = Q.new.d	barycenter
ab	Q.new.ab	line passing through two vertices
ac	Q.new.ca	idem.
ad	Q.new.ad	idem.
bc	Q.new.bc	idem.
bd	Q.new.bd	idem.
cd	Q.new.cd	idem.

13.1.1 Attributs du quadrilatère

```
\begin{tkzelements}
z.A      = point : new ( 0 , 0 )
z.B      = point : new ( 4 , 0 )
z.C      = point : new ( 5 , 1 )
z.D      = point : new ( -1 , 4 )
Q.ABCD   = quadrilateral : new ( z.A , z.B , z.C , z.D )
z.I      = Q.ABCD.i
z.G      = Q.ABCD.g
\end{tkzelements}
```

```
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawSegments(A,C B,D)
\tkzDrawPoints(A,B,C,D,I,G)
\end{tikzpicture}
```



13.2 Quadrilateral methods

TABLE 15 – Méthodes des quadrilatères.

Méthodes	Comments
iscyclic ()	inscrit? (Voir l'exemple suivant)

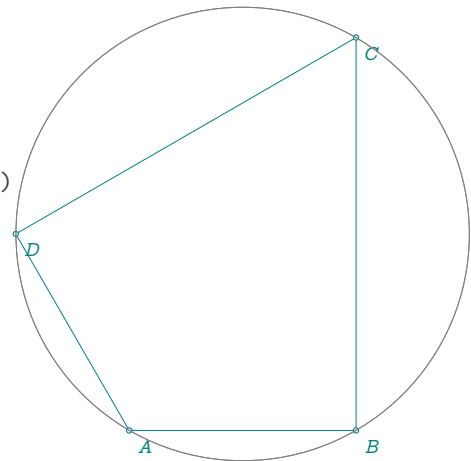
13.2.1 Quadrilatère inscrit

```

\begin{tkzelements}
z.A      = point : new ( 0 , 0 )
z.B      = point : new ( 4 , 0 )
z.D      = point : polar ( 4 , 2*math.pi/3 )
L.DB     = line : new (z.D,z.B)
T.equ    = L.DB : equilateral ()
z.C      = T.equ.pc
Q.new    = quadrilateral : new (z.A,z.B,z.C,z.D)
bool     = Q.new : iscyclic ()
if bool == true then
C.cir    = triangle : new (z.A,z.B,z.C): circum_circle ()
z.O      = C.cir.center
end
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B,C,D)
\tkzDrawCircle(O,A)
\ifthenelse{\equal{\tkzUseLua{bool}}{true}}{
\tkzDrawCircle(O,A)}{}
\end{tikzpicture}

```



14 Classe square

14.1 Attributs du carré

Des points sont créés dans la direction directe. Un test est effectué pour vérifier si les points forment un carré. Dans le cas contraire, la compilation est bloquée.

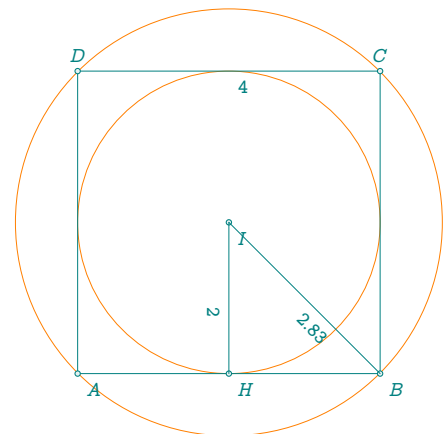
```
Creation S.AB = square : new (z.A,z.B,z.C,z.D)
```

TABLE 16 – Attributs du carré.

Attributs	Application	
pa	$z.A = S.AB.pa$	
pb	$z.B = S.AB.pb$	
pc	$z.C = S.AB.pc$	
pd	$z.D = S.AB.pd$	
type	$S.AB.type = 'square'$	
side	$s = S.AB.center$	$s =$ longueur du côté
center	$z.I = S.AB.center$	centre du carré
extradius	$S.AB.exradius$	rayon du cercle circonscrit
inradius	$S.AB.inradius$	rayon du cercle inscrit
proj	$S.AB.proj$	projection du centre sur un côté
ab	$S.AB.ab$	ligne passant par deux sommets
ac	$S.AB.ca$	idem.
ad	$S.AB.ad$	idem.
bc	$S.AB.bc$	idem.
bd	$S.AB.bd$	idem.
cd	$S.AB.cd$	idem.

14.1.1 Exemple : attributs du carré

```
\begin{tkzelements}
z.A      = point : new ( 0 , 0 )
z.B      = point : new ( 4 , 0 )
z.C      = point : new ( 4 , 4 )
z.D      = point : new ( 0 , 4 )
S.new    = square : new ( z.A , z.B ,z.C,z.D)
z.I      = S.new.center
z.H      = S.new.proj
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles[orange] (I,A I,H)
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D,H,I)
\tkzLabelPoints(A,B,H,I)
\tkzLabelPoints[above] (C,D)
\tkzDrawSegments(I,B I,H)
\tkzLabelSegment[sloped] (I,B){\pmpn{\tkzUseLua{S.new.exradius}}}
\tkzLabelSegment[sloped] (I,H){\pmpn{\tkzUseLua{S.new.inradius}}}
\tkzLabelSegment[sloped] (D,C){\pmpn{\tkzUseLua{S.new.side}}}
\end{tikzpicture}
```



14.2 Méthodes du carré

TABLE 17 – Méthodes du carré.

Méthodes	Comments
rotation (zi,za)	S.IA = square : rotation (z.I,z.A) I centre du carré A premier sommet
side (za,zb)	S.AB = square : side (z.A,z.B) AB est le premier côté (direct)

14.2.1 Carré avec la méthode side

```

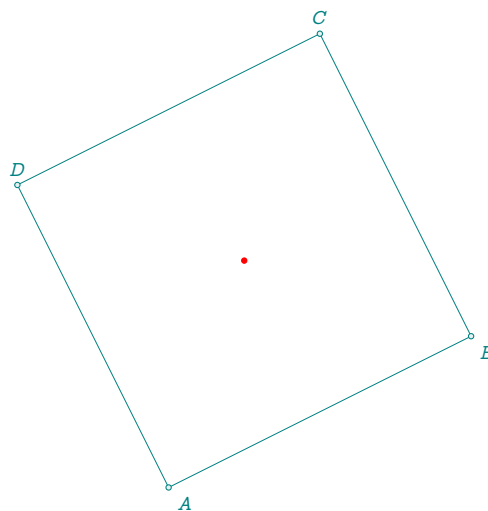
\begin{tkzelements}
  scale      = 2
  z.A        = point : new ( 0 , 0 )
  z.B        = point : new ( 2 , 1 )
  S.side     = square : side (z.A,z.B)
  z.B        = S.side.pb
  z.C        = S.side.pc
  z.D        = S.side.pd
  z.I        = S.side.center
\end{tkzelements}

```

```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C,D)
  \tkzDrawPoints(A,B,C,D)
  \tkzLabelPoints(A,B)
  \tkzLabelPoints[above](C,D)
  \tkzDrawPoints[red](I)
\end{tikzpicture}

```



15 Classe rectangle

15.1 Attributs du rectangle

Points are created in the direct direction. A test is performed to check whether the points form a rectangle, otherwise compilation is blocked.

```
Creation R.ABCD = rectangle : new (z.A,z.B,z.C,z.D)
```

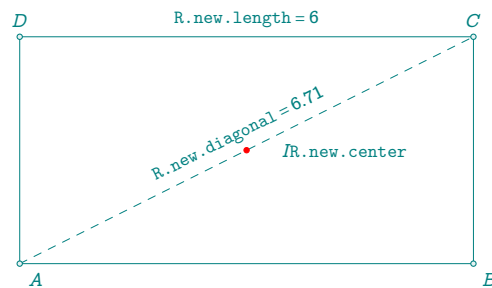
TABLE 18 – Attributs du rectangle.

Attributs	Application	
pa	<code>z.A = R.ABCD.pa</code>	
pb	<code>z.B = R.ABCD.pb</code>	
pc	<code>z.C = R.ABCD.pc</code>	
pd	<code>z.D = R.ABCD.pd</code>	
type	<code>R.ABCD.type= 'rectangle'</code>	
center	<code>z.I = R.ABCD.center</code>	centre du rectangle
length	<code>R.ABCD.length</code>	la longueur
width	<code>R.ABCD.width</code>	la largeur
diagonal	<code>R.ABCD.diagonal</code>	longueur de la diagonale
ab	<code>R.ABCD.ab</code>	droite passant par deux sommets
ac	<code>R.ABCD.ca</code>	idem.
ad	<code>R.ABCD.ad</code>	idem.
bc	<code>R.ABCD.bc</code>	idem.
bd	<code>R.ABCD.bd</code>	idem.
cd	<code>R.ABCD.cd</code>	idem.

15.1.1 Exemple

```
\begin{tkzelements}
z.A = point : new ( 0 , 0 )
z.B = point : new ( 4 , 0 )
z.C = point : new ( 4 , 4 )
z.D = point : new ( 0 , 4 )
R.new = rectangle : new (z.A,z.B,z.C,z.D)
z.I = R.new.center
\end{tkzelements}
```

```
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D)
\tkzDrawPoints[red](I)
\end{tikzpicture}
```



15.2 Méthodes relatives aux rectangles

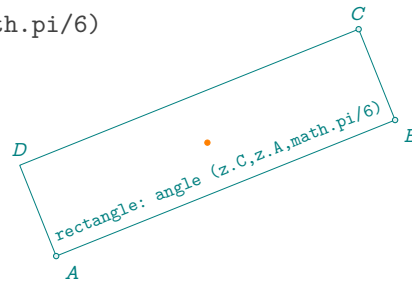
TABLE 19 – Méthodes pour les rectangles.

Méthodes	Comments
angle (zi,za,angle)	R.ang = rectangle : angle (z.I,z.A);z.A vertex; ang angle entre 2 sommets
gold (za,zb)	R.gold = rectangle : gold (z.A,z.B) longueur/largeur = ϕ
diagonal (za,zc)	R.diag = rectangle : diagonal (z.I,z.A) I centre du carré A premier sommet
side (za,zb,d)	S.IA = rectangle : side (z.I,z.A) I centre du carré A premier sommet
get_lengths ()	S.IA = rectangle : get_lengths () I centre du carré A premier sommet

15.2.1 Méthode angle

```
\begin{tkzelements}
scale = .5
z.A = point : new ( 0 , 0 )
z.B = point : new ( 4 , 0 )
z.I = point : new ( 4 , 3 )
P.ABCD = rectangle : angle ( z.I , z.A , math.pi/6)
z.B = P.ABCD.pb
z.C = P.ABCD.pc
z.D = P.ABCD.pd
\end{tkzelements}
```

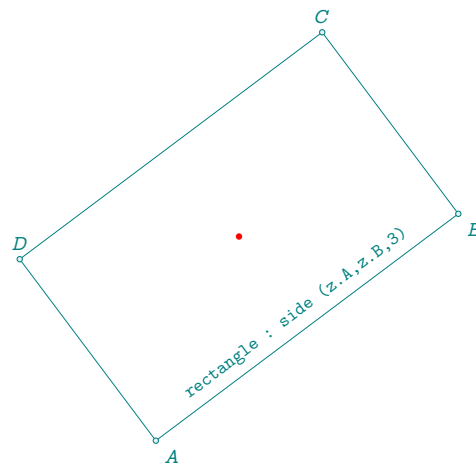
```
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C)
\tkzLabelPoints(A,B,C,D)
\tkzDrawPoints[new](I)
\end{tikzpicture}
```



15.2.2 Méthode side

```
\begin{tkzelements}
z.A = point : new ( 0 , 0 )
z.B = point : new ( 4 , 3 )
R.side = rectangle : side (z.A,z.B,3)
z.C = R.side.pc
z.D = R.side.pd
z.I = R.side.center
\end{tkzelements}
```

```
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D)
\tkzDrawPoints[red](I)
\end{tikzpicture}
```

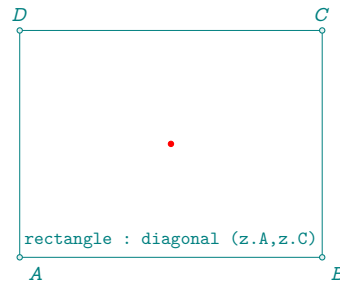


15.2.3 Méthode diagonal

```

\begin{tkzelements}
z.A      = point : new ( 0 , 0 )
z.C      = point : new ( 4 , 3 )
R.diag   = rectangle : diagonal (z.A,z.C)
z.B      = R.diag.pb
z.D      = R.diag.pd
z.I      = R.diag.center
\end{tkzelements}

```



```

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D)
\tkzDrawPoints[red](I)
\tkzLabelSegment[sloped,above](A,B){|rectangle : diagonal (z.A,z.C)|}
\end{tikzpicture}

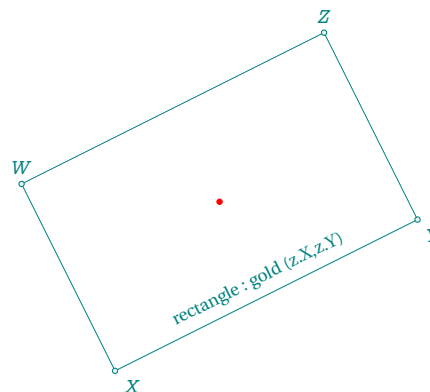
```

15.2.4 Méthode gold

```

\begin{tkzelements}
z.X      = point : new ( 0 , 0 )
z.Y      = point : new ( 4 , 2 )
R.gold   = rectangle : gold (z.X,z.Y)
z.Z      = R.gold.pc
z.W      = R.gold.pd
z.I      = R.gold.center
\end{tkzelements}

```



```

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(X,Y,Z,W)
\tkzDrawPoints(X,Y,Z,W)
\tkzLabelPoints(X,Y)
\tkzLabelPoints[above](Z,W)
\tkzDrawPoints[red](I)
\tkzLabelSegment[sloped,above](X,Y){rectangle : gold (z.X,z.Y)}
\end{tikzpicture}

```


16 Classe parallelogram

16.1 Attributs du parallélogramme

Points are created in the direct direction. A test is performed to check whether the points form a parallelogram, otherwise compilation is blocked.

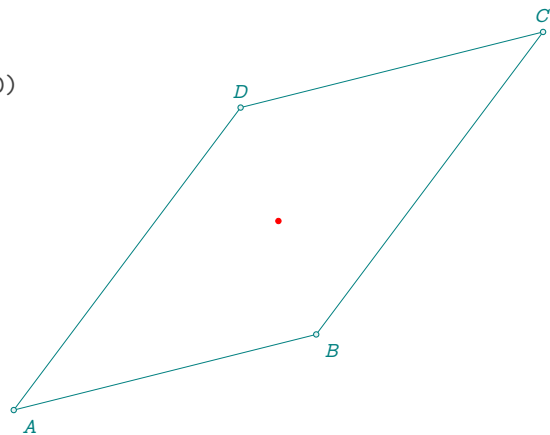
```
Creation P.new = parallelogram : new (z.A,z.B,z.C,z.D)
```

TABLE 20 – Attributs du parallélogramme.

Attributs	Application	
pa	z.A = P.new.pa	
pb	z.B = P.new.pb	
pc	z.C = P.new.pc	
pd	z.D = P.new.pd	
type	P.new.type= 'parallelogram'	
i	z.I = P.new.i	intersection des diagonales
ab	P.new.ab	ligne passant par deux sommets
ac	P.new.ca	idem.
ad	P.new.ad	idem.
bc	P.new.bc	idem.
bd	P.new.bd	idem.
cd	P.new.cd	idem.

16.1.1 Exemple : attributs

```
\begin{tkzelements}
z.A      = point : new ( 0 , 0 )
z.B      = point : new ( 4 , 1 )
z.C      = point : new ( 7 , 5 )
z.D      = point : new ( 3 , 4 )
P.new    = parallelogram : new (z.A,z.B,z.C,z.D)
z.B      = P.new.pb
z.C      = P.new.pc
z.D      = P.new.pd
z.I      = P.new.center
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D)
\tkzDrawPoints[red](I)
\end{tikzpicture}
```



16.2 Méthodes du parallélogramme

TABLE 21 – Méthodes du parallélogramme.

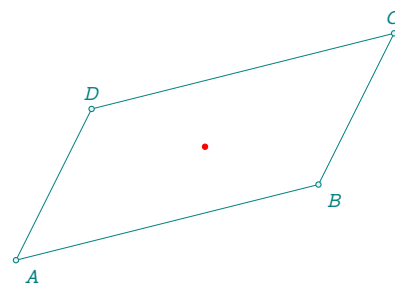
Methods	Comments
fourth (za,zb,zc)	complète un triangle par un parallélogramme (voir l'exemple suivant)

16.2.1 Méthode fourth

```

\begin{tkzelements}
  scale = .75
  z.A    = point : new ( 0 , 0 )
  z.B    = point : new ( 4 , 1 )
  z.C    = point : new ( 5 , 3 )
  P.four = parallelogram : fourth (z.A,z.B,z.C)
  z.D    = P.four.pd
  z.I    = P.four.center
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D)
\tkzDrawPoints[red](I)
\end{tikzpicture}

```



17 Classe regular_polygon

17.1 attributs du polygone régulier

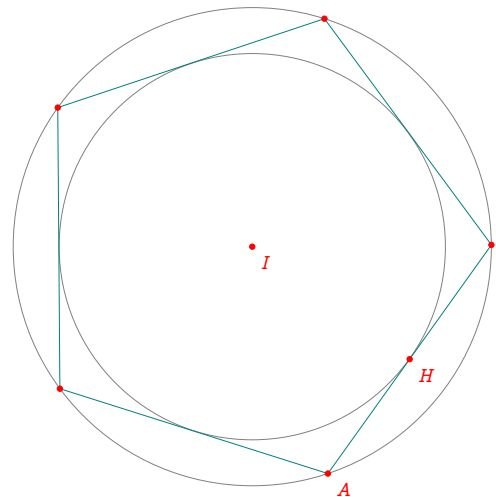
```
Creation RP.IA = regular_polygon : new (z.I,z.A,6)
```

TABLE 22 – Attributs du polygone régulier.

Attributs	Application
center	<code>z.I = RP.IA.center</code>
table	tableau contenant tous les affixes des sommets
through	premier sommet
circle	définit le cercle de centre I passant par A
type	<code>RP.IA.type= 'regular_polygon'</code>
side	<code>s = RP.IA.side</code> ; s = longueur du côté
extradius	<code>S.AB.exradius</code> ; rayon du cercle circonscrit
inradius	<code>S.AB.inradius</code> ; rayon du cercle inscrit
proj	<code>RP.IA.proj</code> ; projection du centre sur un côté
angle	<code>RP.IA.angle</code> ; angle formé par le centre et 2 sommets consécutifs

17.1.1 Pentagone

```
\begin{tkzelements}
z.O = point: new (0,0)
z.I = point: new (1,3)
z.A = point: new (2,0)
RP.five = regular_polygon : new (z.I,z.A,5)
RP.five : name ("P_")
C.ins = circle: radius (z.I,RP.five.inradius)
z.H = RP.five.proj
\end{tkzelements}
\begin{tikzpicture}
\def\nb{\tkzUseLua{RP.five.nb}}
\tkzGetNodes
\tkzDrawCircles(I,A I,H)
\tkzDrawPolygon(P_1,P_...,P_\nb)
\tkzDrawPoints[red](P_1,P_...,P_\nb,H,I)
\tkzLabelPoints[red](I,A,H)
\end{tikzpicture}
```



17.2 Méthodes de polygone régulier

TABLE 23 – Méthodes de polygone régulier.

Méthodes	Comments
<code>new(O,A,n)</code>	<code>RP.five = regular_polygon : new (z.I,z.A,5)</code> ; I centre A premier sommet 5 côtés
Circle	
<code>incircle ()</code>	<code>C.IH = RP.five : incircle ()</code>
Points	
<code>name (string)</code>	Se référer à 17.1.1

18 Classe vector

En fait, il s'agit plus d'une classe de segments orientés que de vecteurs au sens mathématique strict.

Un vecteur est défini en donnant deux points (c'est-à-dire deux affixes). `V.AB = vector : new (z.A, z.B)` crée le vecteur \vec{AB} , c'est-à-dire le segment orienté d'origine A représentant un vecteur. Quelques opérations rudimentaires sont définies, telles que la somme, la soustraction et la multiplication par un scalaire.

La somme est définie comme suit :

Soit $V.AB + V.CD$, on obtient un vecteur $V.AE$ défini comme suit

Si $\vec{CD} = \vec{BE}$ alors $\vec{AB} + \vec{CD} = \vec{AB} + \vec{BE} = \vec{AE}$

```
Création V.AB = vector: new (z.A, z.B)
```

```
z.A = ...
z.B = ...
z.C = ...
z.D = ...
V.AB = vector : new (z.A, z.B)
V.CD = vector : new (z.C, z.D)
V.AE = V.AB + V.CD -- possible V.AB : add (V.CD)
z.E = V.AE.head -- nous récupérons le point final ("head")
```

18.1 Attributs d'un vecteur

TABLE 24 – Attributs d'un vecteur.

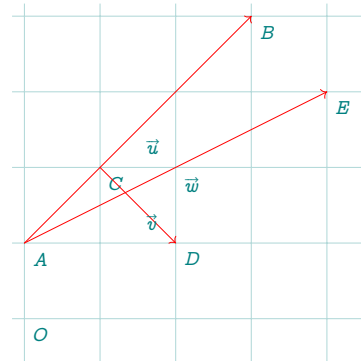
Attributs	Application	Example
tail	<code>V.AB.t = z.A</code>	Refer to (7.2.2)
head	<code>V.AB.head = z.B</code>	Refer to (7.2.2)
type	<code>V.AB.type = 'vector'</code>	
slope	<code>V.AB.slope</code>	Se référer à (18.1.1)
length	<code>V.AB.norm</code>	Se référer à (18.1.1)
mtx	<code>V.AB.mtx</code>	Le résultat est une matrice en colonnes $\{V.AB.t\}, \{V.AB.h\}$

18.1.1 Exemple d'attributs de la classe vector

```

\begin{tkzelements}
  z.O      = point: new (0,0)
  z.A      = point: new (0,1)
  z.B      = point: new (3,4)
  L.AB     = line : new ( z.A , z.B )
  z.C      = point: new (1,2)
  z.D      = point: new (2,1)
  u        = vector : new (z.A,z.B)
  v        = vector : new (z.C,z.D)
  w =u+v
  z.E = w.head
\end{tkzelements}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzLabelPoints(A,B,C,D,O,E)
  \tkzDrawSegments[->,red](A,B C,D A,E)
  \tkzLabelSegment(A,B){$\overrightarrow{u}$}
  \tkzLabelSegment(C,D){$\overrightarrow{v}$}
  \tkzLabelSegment(A,E){$\overrightarrow{w}$}
\end{tikzpicture}
$\overrightarrow{w}$ has slope :
$\tkzDN{\tkzUseLua{math.deg(w.slope)}}^\circ$

```



\vec{w} a une pente : 26.57°

18.2 Méthodes de la classe vector

TABLE 25 – Methods of the class vector.

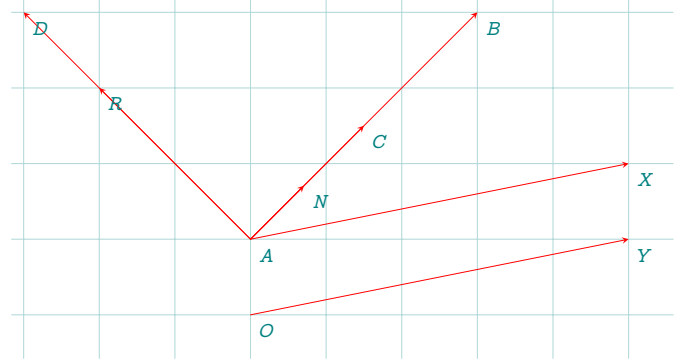
Metamethods	Application	
<code>__add (u,v)</code>	$V.AB + V.CD$	
<code>__sub (u,v)</code>	$V.AB - V.CD$	
<code>__unm (u)</code>	$V.CD = -V.AB$	
<code>__mul (k,u)</code>	$V.CD = k*V.AB$	
Methods	Application	
<code>new(pt, pt)</code>	$V.AB = \text{vector: new (z.A,z.B)}$	
<code>normalize(V)</code>	$V.AB : \text{normalize} ()$	
<code>orthogonal(d)</code>	$V.AB : \text{orthogonal} (d)$	
<code>scale(d)</code>	$V.CD = V.AB : \text{scale} (2)$	$\overrightarrow{CD} = 2\overrightarrow{AB}$
<code>at (V)</code>	$V.DB = V.AC : \text{at} (z.D)$	$\overrightarrow{DB} = \overrightarrow{AC}$

18.2.1 Exemple de méthodes

```

\begin{tkzelements}
  z.O = point: new (0,0)
  z.A = point: new (0,1)
  z.B = point: new (3,4)
  V.AB = vector: new (z.A,z.B)
  V.AC = V.AB : scale (.5)
  z.C = V.AC.head
  V.AD = V.AB : orthogonal ()
  z.D = V.AD.head
  V.AN = V.AB : normalize ()
  z.N = V.AN.head
  V.AR = V.AB : orthogonal(2*math.sqrt(2))
  z.R = V.AR.head
  V.AX = 2*V.AC - V.AR
  z.X = V.AX.head
  V.OY = V.AX : at (z.O)
  z.Y = V.OY.head
\end{tkzelements}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawSegments[>=stealth,->,red](A,B A,C A,D A,N A,R A,X O,Y)
  \tkzLabelPoints(A,B,C,D,O,N,R,X,Y)
\end{tikzpicture}

```



19 Classe matrix

La classe `matrix` est actuellement expérimentale, et ses noms d'attributs et de méthodes n'ont pas encore été finalisés, ce qui indique que cette classe est encore en évolution. Certaines connexions ont été établies avec d'autres classes, telles que la classe `point`. De plus, un nouvel attribut, `mtx`, a été inclus, associant une matrice colonne au point, où les éléments correspondent aux coordonnées du point dans la base d'origine. De même, un attribut a été ajouté à la classe `vector`, où `mtx` représente une matrice colonne composée des deux affixes qui composent le vecteur.

Cette classe `matrix` a été créée pour éviter le besoin d'une bibliothèque externe et a été adaptée aux transformations planes. Elle vous permet d'utiliser des nombres complexes.

Pour afficher des matrices, vous devrez charger le package `amsmath`.

☞ Bien que certaines méthodes soient valides pour n'importe quelle taille de matrice, la majorité sont réservées aux matrices carrées d'ordre 2 et 3.

19.1 Création de la matrice

- La première méthode est la suivante : (Se référer à 19.5.1)

```
M = matrix: new ({a,b},{c,d})
or M = matrix: new {a,b},{c,d}
a, b, c, et d étant des nombres réels ou complexes.
```

$$M = \begin{bmatrix} 3 & 2.25 \\ 4 & 3.90 \end{bmatrix}$$

- Il est également possible d'obtenir une matrice carrée avec : (Se référer à 19.5.7)

```
M = matrix : square (2,a,b,c,d)
```

- Dans le cas d'un vecteur colonne : (Se référer à 19.5.2)

```
V = matrix : vector (1,2,3)
```

$$V = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

- Matrice de transformation homogène (Se référer à 19.5.4)

L'objectif est de générer une matrice aux coordonnées homogènes capable de transformer un système de coordonnées par rotation, translation et mise à l'échelle. Pour ce faire, il est nécessaire de définir à la fois l'angle de rotation, les coordonnées de la nouvelle origine et les facteurs d'échelle.

```
H = matrix : htm (math.pi/3,1,2,2,1)
```

$$H = \begin{bmatrix} 1 & -0.87 & 1 \\ 0.87 & 0.50 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

19.2 Afficher une matrice : méthode print

Cette méthode (référez-vous à 19.5.8) est nécessaire pour contrôler les résultats, donc voici quelques explications sur la façon de l'utiliser. Elle peut être utilisée sur des matrices réelles ou complexes, carrées ou non. Quelques options vous permettent de formater les résultats. Vous devez charger le package `amsmath` pour utiliser la méthode "print". Sans ce package, il est possible d'afficher le contenu de la matrice sans formatage avec `print_array(M)`.

```
\begin{tkzelements}
M = matrix : new {{1,-1},{2,0}}
M : print ()
\end{tkzelements}
```

$$\begin{bmatrix} 1 & -1 \\ 2 & 0 \end{bmatrix}$$

19.3 Attributs d'une matrice

TABLE 26 – Attributs d'une matrice.

Attributs	Application	
set	M.set = {{a,b},{c,d}}	table
rows	M.rows	nombre de lignes
cols	M.cols	nombre de colonnes
type	M.type = "matrix"	le type d'objet
det	M.det	déterminant d'une matrice carrée ou nil

19.3.1 Attribut set

A simple array such as $\{\{1,2\},\{2,-1\}\}$ is often considered a "matrix". In "tkz-elements", we'll consider M defined by `matrix : new ({{1,1},{0,2}})` as a matrix and `M.set` as an array (`M.set = {{1,1},{0,2}}`).

You can access a particular element of the matrix, for example : `M.set [2] [1]` gives 0.

`\tkzUseLua{M.set [2] [1]}` is the expression that displays 2.

The number of rows is accessed with `M.rows` and the number of columns with `M.cols`, here's an example :

```
\begin{tkzelements}
M = matrix : new ({{1,2,3},{4,5,6}})
M : print ()
tex.print("Rows: " .. M.rows)
tex.print("Cols: " .. M.cols)
\end{tkzelements}
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \text{Rows : 2 Cols : 3}$$

19.3.2 Déterminant avec des nombres réels : det

La matrice doit être carrée. Cette bibliothèque a été créée pour des matrices de dimension 2 ou 3, mais il est possible de travailler avec des tailles plus grandes. Le déterminant est un attribut de l'objet "matrix", mais le déterminant peut aussi être obtenu avec la fonction `determinant (M)`.

```
\begin{tkzelements}
M = matrix : square (3,1,1,0,2,-1,-2,1,-1,2)
M : print ()
tex.print ('\\')
tex.print ("Son déterminant est : " .. M.det)
\end{tkzelements}
```

$$\begin{bmatrix} 1 & 1 & 0 \\ 2 & -1 & -2 \\ 1 & -1 & 2 \end{bmatrix} \text{Son déterminant est : -10.0}$$

19.3.3 Déterminant avec les nombres complexes

```
\begin{tkzelements}
a = point :new (1,-2)
b = point :new (0,1)
c = point :new (1,1)
d = point :new (1,-1)
A = matrix : new ({{a, b}, {c,d}})
tex.print(tostring(A.det))
\end{tkzelements}
```

$$-4.00i$$

19.4 Métaméthodes pour les matrices

Les conditions sur les matrices doivent être valides pour que certaines opérations soient possibles.

TABLE 27 – Métaméthodes matricielles.

Métaméthodes	Application
<code>__add(M1,M2)</code>	$M1 + M2$
<code>__sub(M1,M2)</code>	$M1 - M2$
<code>__unm(M)</code>	$- M$
<code>__mul(M1,M2)</code>	$M1 * M2$
<code>__pow(M,n)</code>	$M ^ n$ n entier $>$ ou $<$ 0 ou 'T'
<code>__tostroing(M,n)</code>	<code>tex.print(tostring(M))</code> affiche la matrice
<code>__eq(M1,M2)</code>	vrai ou faux

19.4.1 Addition and subtraction of matrices

Pour simplifier les entrées, j'ai utilisé quelques fonctions pour simplifier les affichages.

```
\begin{tkzelements}
  A = matrix : new ({{1,2},{2,-1}})
  B = matrix : new ({{-1,0},{1,3}})
  S = A + B
  D = A - B
  dsp(A, 'A')
  nl() nl()
  dsp(B, 'B')
  nl() nl()
  dsp(S, 'S') sym(" = ") dsp(A) sym(' + ') dsp(B)
  nl() nl()
  dsp(D, 'D') sym(" = ") dsp(A) sym(' - ') dsp(B)
\end{tkzelements}
```

$$A = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix}$$

$$B = \begin{bmatrix} -1 & 0 \\ 1 & 3 \end{bmatrix}$$

$$S = \begin{bmatrix} 0 & 2 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 1 & 3 \end{bmatrix}$$

$$D = \begin{bmatrix} 2 & 2 \\ 1 & -4 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix} - \begin{bmatrix} -1 & 0 \\ 1 & 3 \end{bmatrix}$$

19.4.2 Multiplication et puissance des matrices

Pour simplifier les entrées, j'ai utilisé quelques fonctions. Vous trouverez leurs définitions dans la section sources de cette documentation.

```
\begin{tkzelements}
  A = matrix : new ({{1,2},{2,-1}})
  B = matrix : new ({{-1,0},{1,3}})
  P = A * B
  I = A^-1
  C = A^3
  K = 2 * A
  T = A^'T'
\end{tkzelements}
```

$$P = \begin{bmatrix} 1 & 6 \\ -3 & -3 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix} * \begin{bmatrix} -1 & 0 \\ 1 & 3 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} 0.20 & 0.40 \\ 0.40 & -0.20 \end{bmatrix}$$

$$K = \begin{bmatrix} 2 & 4 \\ 4 & -2 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix}$$

19.4.3 Métaméthode eq

Tester si deux matrices sont égales ou identiques.

19.5 Méthodes de la classe matrix

TABLE 28 – Méthodes de la classe matrix.

Functions	Comments
<code>new(...)</code>	<code>M = matrix : new ({1,2},{2,-1})</code>
<code>square()</code>	<code>M = matrix : square (2,1,2,2,-1)</code>
<code>vector()</code>	<code>M = matrix : vector (2,1)</code>
<code>htm()</code>	<code>M = matrix : htm (2,1,2,2,-1)</code>
Methods	Comments
<code>print(s,n)</code>	<code>M : print ()</code> s='matrix' or ... default 'bmatrix'
<code>htm_apply(...)</code>	<code>M : htm_apply (...)</code>
<code>get()</code>	<code>M : get (i,j)</code> i = rows , j = cols Refer to 19.5.10
<code>inverse()</code>	<code>M : inverse ()</code>
<code>adjugate()</code>	<code>M : adjugate ()</code>
<code>transpose()</code>	<code>M : transpose ()</code>
<code>is_diagonal()</code>	true or false result :boolean
<code>is_orthogonal()</code>	true or false
<code>homogenization()</code>	<code>M : homogenization ()</code>

19.5.1 Fonction new

C'est la principale méthode pour créer une matrice. Voici un exemple de matrice 2x3 à coefficients complexes :

```

\begin{tkzelements}
a = point : new (1,0)
b = point : new (1,1)
c = point : new (-1,1)
d = point : new (0,1)
e = point : new (1,-1)
f = point : new (0,-1)
M = matrix : new ({a,b,c},{d,e,f})
M : print ()
\end{tkzelements}

```

$$\begin{bmatrix} 1 & 1+i & -1+i \\ i & 1-i & i \end{bmatrix}$$

19.5.2 Fonction vector

Le cas particulier d'une matrice colonne, fréquemment utilisée pour représenter un vecteur, peut être traité comme suit :

```

\begin{tkzelements}
M = matrix : vector (1,2,3)
M : print ()
\end{tkzelements}

```

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

19.5.3 Méthode homogenization

homogenization de vecteur : l'objectif est de pouvoir utiliser une matrice de transformation homogène. Prenons un point A tel que `z.A = point : new (2,-1)`. Pour appliquer une matrice `htm`, nous devons effectuer quelques opérations sur ce point. La première consiste à déterminer le vecteur (matrice) associé au point. C'est simple, car il existe un attribut du point appelé `mtx` qui donne ce vecteur :

```

z.A = point : new (2,-1)
V = z.A.mtx : homogenization ()

```

which gives :

```

\begin{tkzelements}
  pi = math.pi
  M = matrix : htm (pi/4 , 3 , 1)
  z.A = point : new (2,-1)
  V = z.A.mtx : homogenization ()
  z.A.mtx : print ()
  tex.print ('then after homogenization: ')
  V : print ()
\end{tkzelements}

```

$$\begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} \text{ then after homogenization : } \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

19.5.4 Function htm: matrice de transformation homogène

Il y a plusieurs façons d'utiliser cette transformation. Tout d'abord, il faut créer une matrice permettant d'associer une rotation à une translation.

La principale méthode consiste à créer la matrice :

```

pi = math.pi
M = matrix : htm (pi/4 , 3 , 1)

```

Une matrice 3x3 est créée qui combine une rotation de $\pi/4$ et une translation de $\vec{t} = (3, 1)$.

$$\begin{bmatrix} 0.71 & -0.71 & 3 \\ 0.71 & 0.71 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Nous pouvons maintenant appliquer la matrice M. Soit A le point défini ici : 19.5.3. Par homogénéisation, on obtient la matrice colonne V.

```

W = A * V

```

$$\begin{bmatrix} 0.71 & -0.71 & 3 \\ 0.71 & 0.71 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5.12 \\ 1.71 \\ 1 \end{bmatrix}$$

Il ne reste plus qu'à extraire les coordonnées du nouveau point.

19.5.5 Méthode get_htm_point

Dans la section précédente, nous avons obtenu la matrice W. Il nous faut maintenant obtenir le point qu'elle définit.

La méthode get_htm_point permet d'extraire un point d'un vecteur obtenu après application d'une matrice htm.

```

\begin{tkzelements}
  W : print ()
  z.P = get_htm_point(W)
  tex.print("The affix of $$P$ is: ")
  tex.print(display(z.P))
\end{tkzelements}

```

$$\begin{bmatrix} 5.12 \\ 1.71 \\ 1 \end{bmatrix} \text{ The affix of } P \text{ is : } 5.12+1.71i$$

19.5.6 Méthode htm_apply

Les opérations ci-dessus peuvent être simplifiées en utilisant la méthode htm_apply directement au point A.

```

z.Ap = M: htm_apply (z.A)

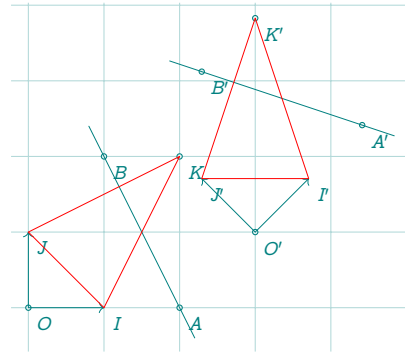
```

Ensuite, la méthode htm_apply transforme un point, une liste de points ou un objet.

```

\begin{tkzelements}
  pi      = math.pi
  M       = matrix : htm (pi/4 , 3 , 1 )
  z.O     = point : new (0,0)
  V.ori   = z.O.mtx : homogenization ()
  z.I     = point : new (1,0)
  z.J     = point : new (0,1)
  z.A     = point: new (2,0)
  z.B     = point: new (1,2)
  L.AB    = line : new (z.A,z.B)
  z.Op,z.Ip,z.Jp = M : htm_apply (z.O,z.I,z.J)
  L.ApBp  = M : htm_apply (L.AB)
  z.Ap    = L.ApBp.pa
  z.Bp    = L.ApBp.pb
  z.K     = point : new (2,2)
  T       = triangle : new ( z.I , z.J , z.K )
  Tp      = M : htm_apply (T)
  z.Kp    = Tp.pc
\end{tkzelements}

```



Nouveau système de coordonnées cartésiennes :

```

\begin{tkzelements}
  pi = math.pi
  tp = tex.print
  nl = '\\\\'
  a = point(1,0)
  b = point(0,1)
  R = matrix : htm (pi/5,2,1)
  R : print () tp(nl)
  v = matrix : vector (1,2)
  v : print ()
  v.h = v : homogenization ()
  v.h : print () tp(nl)
  V = R * v.h
  V : print ()
  z.N = get_htm_point(V)
  tex.print(display(z.N))
\end{tkzelements}

```

$$\begin{bmatrix} 0.81 & -0.59 & 2 \\ 0.59 & 0.81 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1.63 \\ 3.21 \\ 1 \end{bmatrix} 1.63+3.21i$$

19.5.7 Fonction square

Nous avons déjà vu cette méthode dans la présentation des matrices. Il faut d'abord donner l'ordre de la matrice, puis les coefficients, ligne par ligne.

```

\begin{tkzelements}
M = matrix : square (2,2,3,-5,4)
M : print ()
\end{tkzelements}

```

$$\begin{bmatrix} 2 & 3 \\ -5 & 4 \end{bmatrix}$$

19.5.8 Méthode print

Avec le paquetage amsmath chargé, cette méthode peut être utilisée. Par défaut, l'environnement bmatrix est sélectionné, mais vous pouvez choisir entre matrix, pmatrix, Bmatrix, "vmatrix", "Vmatrix". Une autre option permet de définir le nombre de chiffres après la virgule. La variable globale "tkz_dc" est utilisée pour définir le nombre de décimales. Voici un exemple :

```
\begin{tkzelements}
  M = matrix : new ({math.sqrt(2),math.sqrt(3)},{math.sqrt(4),math.sqrt(5)})
  M : print ('pmatrix')
\end{tkzelements}
```

$$\begin{pmatrix} 1.414 & 1.732 \\ 2 & 2.236 \end{pmatrix}$$

Vous pouvez également afficher la matrice sous la forme d'un simple tableau à l'aide de la fonction `print_array (M)`, voir l'exemple suivant.

Dans le cas d'une matrice carrée, il est possible de transmettre une liste de valeurs dont le premier élément est l'ordre de la matrice.

```
\begin{tkzelements}
  M = matrix : square (2,1,0,0,2)
  M : print ()
\end{tkzelements}
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

19.5.9 Afficher un tableau : fonction `print_array`

Nous aurons besoin d'afficher les résultats, donc examinons les différentes façons de les afficher et distinguons les différences entre les tableaux et les matrices.

Ci-dessous, A est un tableau. Il peut être affiché comme un simple tableau ou comme une matrice, mais nous ne pouvons pas utiliser les attributs et `A : print ()` n'est pas possible car A n'est pas un objet de la classe `matrix`. Si vous souhaitez afficher un tableau comme une matrice, vous pouvez utiliser la fonction `print_matrix` (référez-vous à l'exemple suivant).

```
\begin{tkzelements}
  A = {{1,2},{1,-1}}
  tex.print ('A = ') print_array (A)
  tex.print (' or ')
  print_matrix (A)
  M = matrix : new ({1,1},{0,2})
  tex.print ('\\\\\\')
  tex.print ('M = ') M : print ()
\end{tkzelements}
```

$$A = \{\{1, 2\}, \{1, -1\}\} \text{ or } \begin{bmatrix} 1 & 2 \\ 1 & -1 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$

19.5.10 Obtenir un élément d'une matrice : méthode `get`

```
\begin{tkzelements}
  M = matrix : new {{1,2},{2,-1}}
  S = M: get(1,1) + M: get(2,2)
  tex.print(S)
\end{tkzelements}
```

$$0$$

19.5.11 Matrice inverse : méthode `inverse`

```
\begin{tkzelements}
  A = matrix : new ({1,2},{2,-1})
  tex.print("Inverse of $A = $")
  B = A : inverse ()
  B : print ()
\end{tkzelements}
```

$$\text{Inverse of } A = \begin{bmatrix} 0.43 & 0.29 \\ 0.29 & -0.14 \end{bmatrix}$$

19.5.12 Matrice inverse avec syntaxe de puissance

```
\begin{tkzelements}
M = matrix : new ({1,0,1},{1,2, 1},{0,-1,2})
tex.print("$M = $") print_matrix (M)
tex.print('\\\\')
tex.print("Inverse of $M = M^{-1} = $")
print_matrix (M^{-1})
\end{tkzelements}
```

$$M = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 0 & -1 & 2 \end{bmatrix}$$

$$\text{Inverse of } M = M^{-1} = \begin{bmatrix} 1.25 & -0.25 & -0.50 \\ -0.50 & 0.50 & 0 \\ -0.25 & 0.25 & 0.50 \end{bmatrix}$$

19.5.13 Transposition de la matrice : méthode transpose

Une matrice transposée est accessible avec `A : transpose ()` ou avec `A^{T'}`.

```
\begin{tkzelements}
A = matrix : new ({1,2},{2,-1})
AT = A : transpose ()
tex.print("$A^{T'} = $")
AT : print ()
\end{tkzelements}
```

$$A^{T'} = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix}$$

Remark: $(A^{T'})^{T'} = A$

19.5.14 Méthode adjugate

```
\begin{tkzelements}
N = matrix : new {{1, 0, 3},{2, 1, 0},{-1, 2, 0}}
tex.print('N = ') print_matrix(N)
tex.print('\\\\')
N.a = N : adjugate ()
N.i = N * N.a
tex.print('adj(N) = ') N.a : print ()
tex.print('\\\\')
tex.print('N $\times$ adj(N) = ') print_matrix(N.i)
tex.print('\\\\')
tex.print('det(N) = ') tex.print(N.det)
\end{tkzelements}
```

$$N = \begin{bmatrix} 1 & 0 & 3 \\ 2 & 1 & 0 \\ -1 & 2 & 0 \end{bmatrix} \text{adj}(N) = \begin{bmatrix} 0 & 6 & -3 \\ 0 & 3 & 6 \\ 5 & -2 & 1 \end{bmatrix}$$

$$N \times \text{adj}(N) = \begin{bmatrix} 15 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 15 \end{bmatrix}$$

$$\det(N) = 15.0$$

19.5.15 Méthode identity

Création de la matrice identité d'ordre 3

```
\begin{tkzelements}
Id_3 = matrix : identity (3)
Id_3 : print ()
\end{tkzelements}
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

19.5.16 Diagonalisation : méthode diagonalize

Pour l'instant, cette méthode ne concerne que les matrices d'ordre 2.

```
\begin{tkzelements}
A = matrix : new {{5,-3}, {6,-4}}
tex.print('A = ') A : print ()
D,P = A : diagonalize ()
tex.print('D = ') D : print ()
tex.print('P = ') P : print ()
R = P^(-1)*A*P
tex.print('\\\\')
tex.print('Test: $D = P^{-1}AP = $ ')
R : print ()
tex.print('\\\\')
tex.print('Verification: $P^{-1}P = $ ')
T = P^(-1)*P
T : print ()
\end{tkzelements}
```

$$A = \begin{bmatrix} 5 & -3 \\ 6 & -4 \end{bmatrix} D = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix} P = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

$$\text{Test: } D = P^{-1}AP = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\text{Verification: } P^{-1}P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

19.5.17 Méthode is_orthogonal

La méthode renvoie vrai si la matrice est orthogonale et faux dans le cas contraire.

```
\begin{tkzelements}
local cos = math.cos
local sin = math.sin
local pi = math.pi
A = matrix : new ({{cos(pi/6),-sin(pi/6)}, {sin(pi/6),cos(pi/6)}})
A : print ()
bool = A : is_orthogonal ()
tex.print('\\\\')
if bool
then
tex.print("The matrix is orthogonal")
else
tex.print("The matrix is not orthogonal")
end
tex.print('\\\\')
tex.print('Test: $A^T = A^{-1} ?$')
print_matrix(transposeMatrix (A))
tex.print('=')
inv_matrix (A) : print ()
\end{tkzelements}
```

$$\begin{bmatrix} 0.87 & -0.50 \\ 0.50 & 0.87 \end{bmatrix}$$

The matrix is orthogonal

$$\text{Test: } A^T = A^{-1} ? \begin{bmatrix} 0.87 & 0.50 \\ -0.50 & 0.87 \end{bmatrix} = \begin{bmatrix} 0.87 & 0.50 \\ -0.50 & 0.87 \end{bmatrix}$$

19.5.18 Méthode is_diagonal

La méthode renvoie vrai si la matrice est diagonale et faux dans le cas contraire.

20 Constantes et fonctions mathématiques

TABLE 29 – Constantes et fonctions mathématiques.

Contants ou fonctions	Comments
tkzphi	constant $\varphi = (1 + \mathit{math.sqrt}(5))/2$
tkzinvpfi	constant $1/\varphi = 1/\mathit{tkzphi}$
tkzsqrtpfi	constant $\sqrt{\varphi} = \mathit{math.sqrt}(\mathit{tkzphi})$
length (a,b)	point.abs(a-b) Se référer à (8.2.2)
islinear (z1,z2,z3)	Les points sont-ils alignés? $(z2-z1) \parallel (z3-z1)$?
isortho (z1,z2,z3)	$(z2-z1) \perp (z3-z1)$? booléen
get_angle (z1,z2,z3)	le sommet est z1 Se référer à (20.8)
bisector (z1,z2,z3)	L.Aa = bisector (z.A,z.B,z.C) depuis A (20.8)
bisector_ext (z1,z2,z3)	L.Aa = bisector_ext (z.A,z.B,z.C) from A
altitude (z1,z2,z3)	altitude de z1
set_lua_to_tex (list)	set_lua_to_tex('a','n') définit \a et \n
tkzUseLua (variable)	<code>\textbackslash\tkzUseLua{a}</code> imprime la valeur de a
value (v)	appliquerscale * value
real (v)	applique value /scale
angle_normalize (an)	pour obtenir une valeur comprise entre 0 et 2π
barycenter ({z1,n1},{z2,n2}, ...)	barycentre d'une liste de points
solve_quadratic (a,b,c)	donne la solution de $ax^2 + bx + c = 0$ a,b,c réelle ou complexe (Se référer à 20.12.1)

20.1 Longueur d'un segment

`length(z.A,z.B)` is a shortcut for `point.abs(z.A-z.B)`. Il n'est donc pas nécessaire d'utiliser des complexes.

20.2 Division harmonique avec tkzphi

```
\begin{tkzelements}
  scale =.5
  z.a = point: new(0,0)
  z.b = point: new(8,0)
  L.ab = line: new (z.a,z.b)
  z.m,z.n = L.ab: harmonic_both (tkzphi)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLine[add= .2 and .2](a,n)
  \tkzDrawPoints(a,b,n,m)
  \tkzLabelPoints(a,b,n,m)
\end{tikzpicture}
```

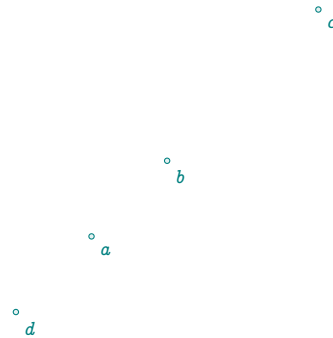


20.3 Fonction islinear

```

\begin{tkzelements}
  z.a = point: new (1, 1)
  z.b = point: new (2, 2)
  z.c = point: new (4, 4)
  if islinear (z.a,z.b,z.c) then
    z.d = point: new (0, 0)
  else
    z.d = point: new (-1, -1)
  end
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPoints(a,...,d)
  \tkzLabelPoints(a,...,d)
\end{tikzpicture}

```



20.4 Fonction value

valeur à appliquer pour la mise à l'échelle si nécessaire

Si $\text{scale} = 1.2$ avec $a = \text{valeur}(5)$ la valeur réelle de a sera $5 \times 1.2 = 6$.

20.5 Function real

If $\text{scale} = 1.2$ with $a = 6$ then $\text{real}(a) = 6/1.2 = 5$.

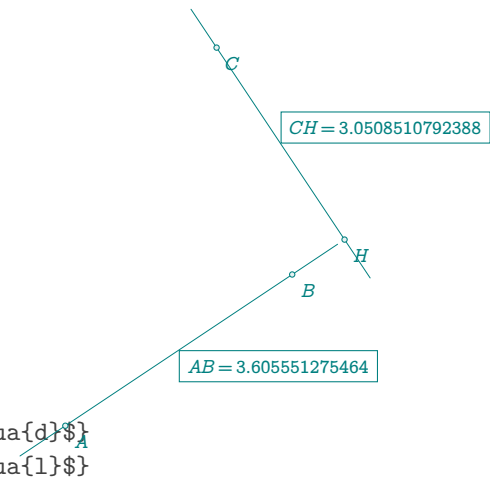
20.6 Transfer from lua to $\text{T}_\text{E}_\text{X}$

Il est possible de transférer une variable de Lua à $\text{T}_\text{E}_\text{X}$ avec `\tkzUseLua`.

```

\begin{tkzelements}
  z.A      = point : new (0 , 0)
  z.B      = point : new (3 , 2)
  z.C      = point : new (2 , 5)
  L.AB     = line  : new (z.A,z.B)
  d        = L.AB : distance (z.C)
  l        = L.AB : length
  z.H      = L.AB : projection (z.C)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(A,B C,H)
  \tkzDrawPoints(A,B,C,H)
  \tkzLabelPoints(A,B,C,H)
  \tkzLabelSegment[above right,draw] (C,H){\$CH = \tkzUseLua{d}\$}
  \tkzLabelSegment[below right,draw] (A,B){\$AB = \tkzUseLua{l}\$}
\end{tikzpicture}

```



20.7 Angles normalisés : Pente des droites (ab), (ac) and (ad)

```

\begin{tkzelements}
  z.a      = point: new(0, 0)
  z.b      = point: new(-3, -3)
  z.c      = point: new(0, 3)
  z.d      = point: new(2, -2)

```

```

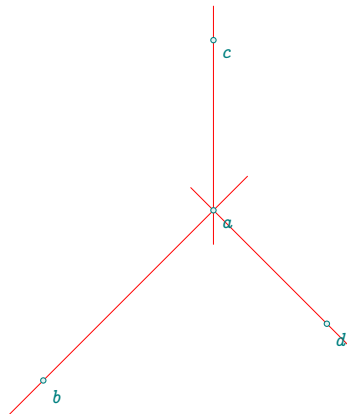
angle    = point.arg (z.b-z.a)
tex.print('slope of (ab) : '..tostring(angle)..'\\')
tex.print('slope normalized of (ab) : '..tostring(angle\_normalize(angle))..'\\')
angle    = point.arg (z.c-z.a)
tex.print('slope of (ac) : '..tostring(angle)..'\\')
tex.print('slope normalized of (ac) : '..tostring(angle\_normalize(angle))..'\\')
angle    = point.arg (z.d-z.a)
tex.print('slope of (ad) : '..tostring(angle)..'\\')
tex.print('slope normalized of (ad) : '..tostring(angle\_normalize(angle))..'\\')
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines[red](a,b a,c a,d)
  \tkzDrawPoints(a,b,c,d)
  \tkzLabelPoints(a,b,c,d)
\end{tikzpicture}

```

```

slope of (ab) : -2.3561944901923
slope normalized of (ab) : 3.9269908169872
slope of (ac) : 1.5707963267949
slope normalized of (ac) : 1.5707963267949
slope of (ad) : -0.78539816339745
slope normalized of (ad) : 5.4977871437821

```



20.8 Obtenir un angle

La fonction `get_angle (a,b,c)` donne l'angle normalisé de (\vec{ab}, \vec{ac}) .

```

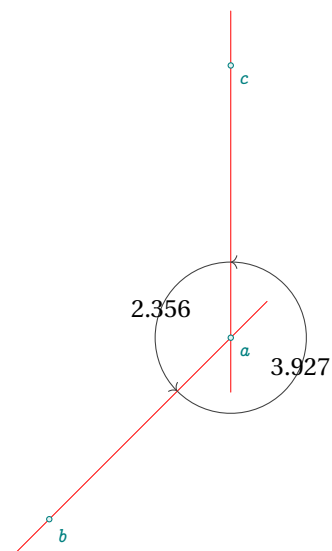
\begin{tkzelements}
  z.a  = point: new(0, 0)
  z.b  = point: new(-2, -2)
  z.c  = point: new(0, 3)
  angcb = tkzround ( get_angle (z.a,z.c,z.b),3)
  angbc = tkzround ( get_angle (z.a,z.b,z.c),3)
\end{tkzelements}

```

```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines[red](a,b a,c)
  \tkzDrawPoints(a,b,c)
  \tkzLabelPoints(a,b,c)
  \tkzMarkAngle[->](c,a,b)
  \tkzLabelAngle(c,a,b){\tkzUseLua{angcb}}
  \tkzMarkAngle[->](b,a,c)
  \tkzLabelAngle(b,a,c){\tkzUseLua{angbc}}
\end{tikzpicture}

```



20.9 Produit de point ou produit scalaire

```

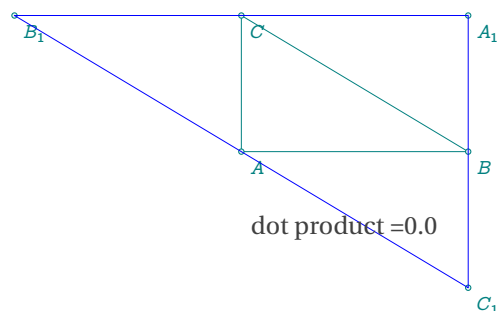
\begin{tkzelements}
  z.A  = point: new(0,0)
  z.B  = point: new(5,0)
  z.C  = point: new(0,3)
  T.ABC = triangle: new (z.A,z.B,z.C)
  z.A_1,
  z.B_1,
  z.C_1 = get_points (T.ABC: anti ())
  x = dot_product (z.A,z.B,z.C)
\end{tkzelements}

```

```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawPoints(A,B,C,A_1,B_1,C_1)
  \tkzLabelPoints(A,B,C,A_1,B_1,C_1)
  \tkzDrawPolygon[blue](A_1,B_1,C_1)
  \tkzText[right](0,-1){dot product =\tkzUseLua{x}}
\end{tikzpicture}

```



Le produit scalaire des vecteurs \overrightarrow{AC} et \overrightarrow{AB} est égal à 0.0, ces vecteurs sont donc orthogonaux.

20.10 Alignement ou orthogonalité

Avec les fonctions `islinear` et `isortho`. `islinear(z.a,z.b,z.c)` donne vrai si les points a , b et c sont alignés. `isortho(z.a,z.b,z.c)` donne vrai si la droite (ab) est orthogonale à la droite (ac) .

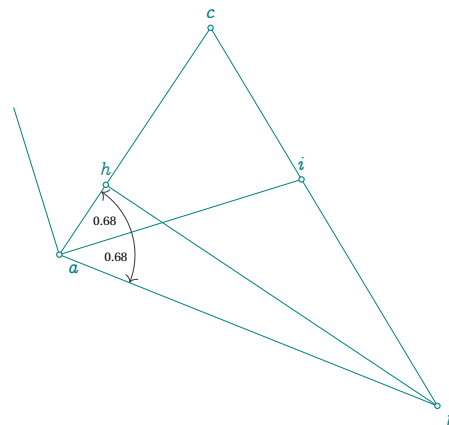
20.11 Bissectrice et hauteur

Ces fonctions sont utiles si vous n'avez pas besoin de créer un objet triangulaire utile pour le reste de votre code.

```

\begin{tkzelements}
  z.a = point: new (0, 0)
  z.b = point: new (5, -2)
  z.c = point: new (2, 3)
  z.i = bisector (z.a,z.c,z.b).pb
  z.h = altitude (z.b,z.a,z.c).pb
  angic = tkzround ( get_angle (z.a,z.i,z.c),2)
  angci = tkzround ( get_angle (z.a,z.b,z.i),2)
  z.e = bisector_ext (z.a,z.b,z.c).pb
\end{tkzelements}

```



```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(a,b,c)
  \tkzDrawSegments(a,i b,h a,e)
  \tkzDrawPoints(a,b,c,i,h)
  \tkzLabelPoints(a,b)
  \tkzLabelPoints[above](c,i,h)
  \tkzMarkAngle[->](i,a,c)
  \tkzLabelAngle[font=\tiny,pos=.75](i,a,c){\tkzUseLua{angci}}
  \tkzMarkAngle[<-](b,a,i)
  \tkzLabelAngle[font=\tiny,pos=.75](b,a,i){\tkzUseLua{angic}}
\end{tikzpicture}

```

20.12 Autres fonctions

Non documenté car encore en version beta : parabole, Cramer22, Cramer33.

20.12.1 Fonction solve_quadratic

Cette fonction résout l'équation $ax^2 + bx + c = 0$ avec des nombres réels ou complexes.

```

\begin{tkzelements}
  tex.sprint('Solve : $x^2+1=0$ The solution set is ')
  r1,r2 = solve_quadratic(1,0,1)
  tex.print('\{\}..\tostring(r1).. ' , '..tostring(r2)..'\}')
  tex.print('\{\}\}')
  tex.sprint('Solve : $x^2+2x-3=0$ The solution set is ')
  r1,r2 = solve_quadratic(1,2,-3)
  tex.print('\{\}..\tostring(r1).. ' , '..tostring(r2)..'\}')
  tex.print('\{\}\}')
  a = point (0,1)
  b = point (1,1)
  c = point (-1,1)
  tex.sprint('Solve : $ix^2+(1+i)x+(-1+i)=0$ The solution set is ')
  r1,r2 = solve_quadratic(a,b,c)
  tex.print('\{\}..\tostring(r1).. ' , '..tostring(r2)..'\}')
\end{tkzelements}

```

Solve : $x^2 + 1 = 0$ The solution set is {i, -i}

Solve : $x^2 + 2x - 3 = 0$ The solution set is {1.0, -3.0}

Solve : $ix^2 + (1+i)x + (-1+i) = 0$ The solution set is {0.13-0.68i, -1.13+1.68i}

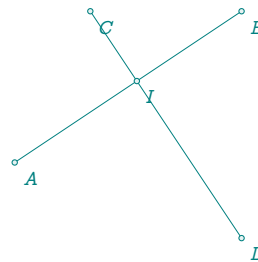
21 Intersections

C'est un outil essentiel. Pour l'instant, les classes concernées sont les droites, les cercles et les ellipses, avec les combinaisons suivantes : ligne-ligne ; ligne-cercle ; cercle-cercle et ligne-ellipse. L'argument est une paire d'objets, dans n'importe quel ordre. Les résultats consistent en une ou deux valeurs, soit des points, soit des booléens **false**, soit des traits de soulignement `_`.

21.1 Droite-droite

Le résultat est de la forme : point ou false.

```
\begin{tkzelements}
  z.A = point : new (1,-1)
  z.B = point : new (4,1)
  z.C = point : new (2,1)
  z.D = point : new (4,-2)
  z.I = point : new (0,0)
  L.AB = line : new (z.A,z.B)
  L.CD = line : new (z.C,z.D)
  x = intersection (L.AB,L.CD)
  if x == false then
    tex.print('error')
  else
    z.I = x
  end
\end{tkzelements}
```



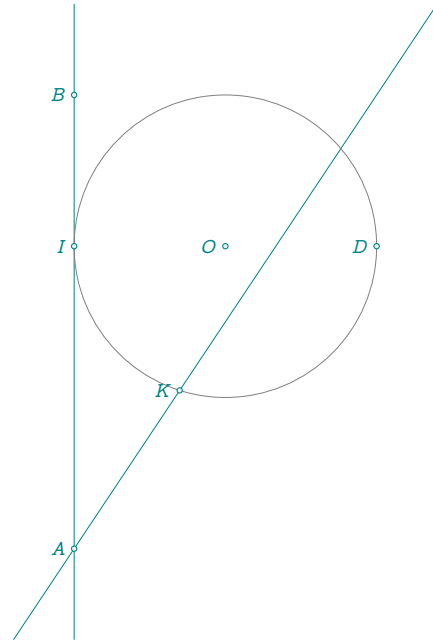
```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegments(A,B C,D)
  \tkzDrawPoints(A,B,C,D,I)
  \tkzLabelPoints(A,B,C,D,I)
\end{tikzpicture}
Autres exemples : 10.2.1, 10.2.2, 23.3
```

21.2 Droite-cercle

Le résultat est de la forme : point ,point ou faux ,faux. Si la droite est tangente au cercle, les deux points sont identiques. Vous pouvez ignorer l'un des points en utilisant le trait de soulignement : `_`,point ou point,`_`. Lorsque l'intersection donne deux solutions, l'ordre des points est déterminé par l'argument de $(z.p - z.c)$ avec c centre du cercle et p point d'intersection. La première solution correspond à l'argument le plus petit (les arguments sont compris entre 0 et 2π).

```
\begin{tkzelements}
  z.A = point : new (1,-1)
  z.B = point : new (1,2)
  L.AB = line : new (z.A,z.B)
  z.O = point : new (2,1)
  z.D = point : new (3,1)
  z.E = point : new (3,2)
  L.AE = line : new (z.A,z.E)
  C.OD = circle : new (z.O,z.D)
  z.I,_ = intersection (L.AB,C.OD)
  _,z.K = intersection (C.OD,L.AE)
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
  \tkzDrawLines(A,B A,E)
  \tkzDrawCircle(O,D)
  \tkzDrawPoints(A,B,O,D,I,K)
  \tkzLabelPoints[left] (A,B,O,D,I,K)
\end{tikzpicture}
Autres exemples : 10.2.1
```



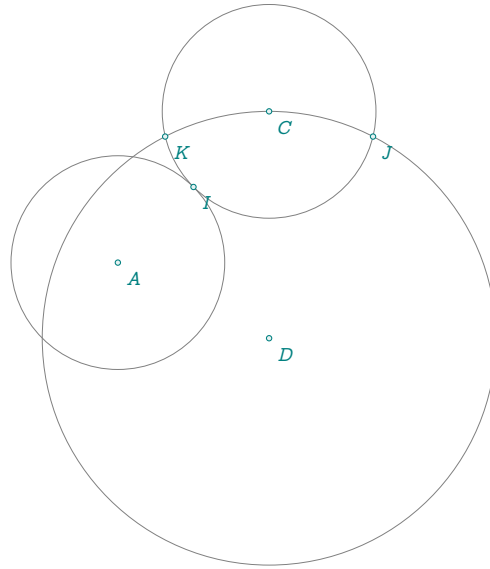
21.3 Cercle-cercle

Le résultat est de la forme : point , point ou faux , faux. Si les cercles sont tangents, les deux points sont identiques. Vous pouvez ignorer l'un des points en utilisant le trait de soulignement : `_` , point ou point `,` `_`. Comme pour l'intersection d'une droite et d'un cercle, on considère l'argument de `z.p-z.c` avec `c` centre du premier cercle et `p` point d'intersection. La première solution correspond au plus petit argument (les arguments sont compris entre 0 et 2π).

```

\begin{tkzelements}
  z.A      = point : new (1,1)
  z.B      = point : new (2,2)
  z.C      = point : new (3,3)
  z.D      = point : new (3,0)
  C.AB     = circle : new (z.A,z.B)
  C.CB     = circle : new (z.C,z.B)
  z.I,_    = intersection (C.AB,C.CB)
  C.DC     = circle : new (z.D,z.C)
  z.J,z.K  = intersection (C.DC,C.CB)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles(A,B C,B D,C)
  \tkzDrawPoints(A,I,C,D,J,K)
  \tkzLabelPoints(A,I,C,D,J,K)
\end{tikzpicture}
Autres exemples : 10.2.1, 3.3

```



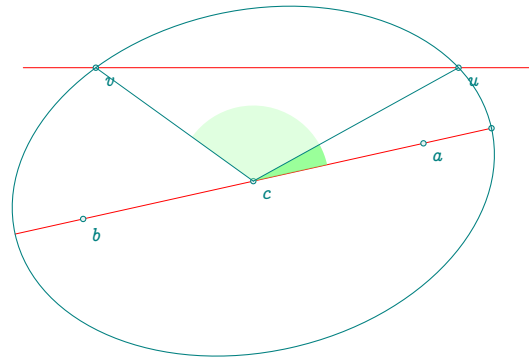
21.4 Droite-ellipse

L'exemple suivant est complexe, mais il montre les possibilités de Lua. La désignation des points d'intersection est un peu plus compliquée que la précédente, car il faut tenir compte de l'argument caractérisant le grand axe. Le principe est le même, mais il faut soustraire cet argument. Concrètement, il faut considérer les pentes des droites formées par le centre de l'ellipse et les points d'intersection, et la pente du grand axe.

```

\begin{tkzelements}
  scale      = .5
  z.a        = point: new (5 , 2)
  z.b        = point: new (-4 , 0)
  z.m        = point: new (2 , 4)
  z.n        = point: new (4 , 4)
  L.ab       = line : new (z.a,z.b)
  L.mn       = line : new (z.m,z.n)
  z.c        = L.ab. mid
  z.e        = L.ab: point (-.2)
  E          = ellipse: foci (z.a,z.b,z.e)
  z.u,z.v    = intersection (E,L.mn)
  -- transfer to tex
  a          = E.Rx
  b          = E.Ry
  ang        = math.deg(E.slope)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines[red](a,b u,v) % p,s p,t
  \tkzDrawPoints(a,b,c,e,u,v) %
  \tkzLabelPoints(a,b,c,u,v)
  \tkzDrawEllipse[teal](c,\tkzUseLua{a},\tkzUseLua{b},\tkzUseLua{ang})
  \tkzDrawSegments(c,u c,v)
  \tkzFillAngles[green!30,opacity=.4](e,c,v)
  \tkzFillAngles[green!80,opacity=.4](e,c,u)
\end{tikzpicture}
Autres exemples : 12.2.2, 23.32

```



22 Étude approfondie

22.1 Les tables

22.1.1 General tables

Les tableaux sont le seul "conteneur" de structure de données intégré à Lua. Ce sont des tableaux associatifs qui associent une clé (référence ou index) à une valeur sous la forme d'un champ (ensemble) de paires clé/valeur. De plus, les tableaux n'ont pas de taille fixe et peuvent croître dynamiquement en fonction de nos besoins.

Les tableaux sont créés à l'aide de constructeurs de tableaux, dont le plus simple est l'utilisation d'accolades, par exemple `{}`. Cela définit un tableau vide.

```
F = {"banane", "pomme", "cerise"}
```

`print(F[2])` -> pomme

qui peut être également défini par

```
FR = {[1] = "banane", [3] = "cerise", [2] = "pomme"}
```

`print(FR[3])` -> cerise

`FR[4] = "orange"`

```
print(#FR)
-- I for Index
for I,V in ipairs(FR) do
    print(I,V)
end
```

1 banane

2 pommes

3 cerise

4 orange

```
C = {"banane" = "jaune" , ["pomme"] = "verte" , ["cerise"] = "rouge" }
C.orange = "orange"
```

```
for K,V in pairs (C) do
    print(K,V)
end
```

banane = jaune cerise = rouge orange = orange pomme = vert

Une autre fonction utile est la possibilité de créer un tableau pour stocker un nombre inconnu de paramètres de fonctions, par exemple :

```
function ReturnTable (...)
    return table.pack (...)
end
```

```
function ParamToTable (...)
  mytab = ReturnTable(...)
  for i=1,mytab.n do
    print(mytab[i])
  end
end
ParamToTable("cerise","pomme","orange")
```

Utilisation de tableaux avec la syntaxe `table[key]` :

```
C["banana"] and F[1]
```

But with string constants as keys we have the sugar syntax : `C.banana` but this syntax does not accept numbers. It's possible to erase a key/value pair from a table, with :

```
C.banane = nil
```

22.1.2 Table `z`

Il s'agit du tableau le plus important du paquet. Elle stocke tous les points et permet de les transférer dans TikZ. Elle est définie avec `z = {}`, puis chaque fois que nous écrivons

```
z.name = point : new (a , b)
```

un objet point est stocké dans la table. La clé est `nom`, la valeur est un objet. Nous avons vu que `z.name.re = a` et que `z.name.im = b`.

Cependant, les éléments de ce tableau ont des propriétés essentielles.

Par exemple, si l'on souhaite afficher un élément, alors `tex.print(tostring(z.name)) = a+ib` l'opération `tostring` affiche l'affixe correspondant au point.

De plus, nous verrons qu'il est possible d'effectuer des opérations avec les éléments du tableau `z`.

22.2 Transferts

Nous avons vu (sous-section 6.1.1) que la macro transfère les coordonnées des points vers TikZ. Regardons de plus près cette macro :

```
\def\tkzGetNodes{\directlua{%
  for K,V in pairs(z) do
    local K,n,sd,ft
    n = string.len(KS)
    if n > 1 then
      _,_,ft, sd = string.find( K , "(.+)(") )
      if sd == "p" then K=ft.." end
    end
    tex.print("\coordinate ("..K..) at ("..V.re..","..V.im..) ;\\")
  end}
}
```

Il s'agit principalement d'une boucle. Les variables utilisées sont `K` (pour les clés) et `V` (pour les valeurs). Pour extraire des paires (clé/valeur) de la table `z`, utilisez la fonction `pairs`. `K` devient le nom d'un nœud dont les coordonnées sont `V.re` et `V.im`. Parallèlement, nous recherchons les clés comportant plus d'un symbole se terminant par `p`, afin de les associer au symbole `""` valable dans TikZ.

22.3 Bibliothèque des nombres complexes et point

À moins que vous ne souhaitiez créer vos propres fonctions, vous n'aurez pas besoin de connaître et d'utiliser les nombres complexes. Toutefois, dans certains cas, il peut être utile de mettre en œuvre certaines de leurs propriétés.

`z.A = point : nouveau (1,2)` et `z.B = point : nouveau (1,-1)` définissent deux affixes qui sont $z_A = 1 + 2i$ et $z_B = 1 - i$. Notez la différence de notation entre `z.A` et z_A pour deux entités distinctes : un objet Lua et un affixe.

Si vous souhaitez utiliser uniquement des nombres complexes, vous devez choisir la syntaxe suivante : `za = point (1,2)`. La différence entre `z.A = point : new (1,2)` et `za = point (1,2)` est que la première fonction prend en compte l'échelle. Si `scale = 2` alors $z_A = 2 + 4i$. De plus, l'objet référencé par A est stocké dans la table `z` et non dans `za`.

La notation peut surprendre, car j'ai utilisé le terme "point". L'objectif ici n'était pas de créer une bibliothèque complète sur les nombres complexes, mais de pouvoir utiliser leurs principales propriétés par rapport aux points. Je ne voulais pas avoir deux niveaux différents, et puisqu'une connexion unique peut être établie entre les points du plan et les complexes, j'ai décidé de ne pas mentionner les nombres complexes! Mais ils sont là.

TABLE 30 – Métaméthodes des points ou des complexes.

Métaméthodes	Application	
<code>__add(z1,z2)</code>	<code>z.a + z.b</code>	affixe
<code>__sub(z1,z2)</code>	<code>z.a - z.b</code>	affixe
<code>__unm(z)</code>	<code>- z.a</code>	affixe
<code>__mul(z1,z2)</code>	<code>z.a * z.b</code>	affixe
<code>__concat(z1,z2)</code>	<code>z.a .. z.b</code>	produit scalaire = nombre réel ^a
<code>__pow(z1,z2)</code>	<code>z.a ^ z.b</code>	déterminant = nombre réel
<code>__div(z1,z2)</code>	<code>z.a / z.b</code>	produit scalaire
<code>__tostring(z)</code>	<code>tex.print(tostring(z))</code>	affiche l'affixe
<code>__tonumber(z)</code>	<code>tonumber(z)</code>	produit scalaire ou nil
<code>__eq(z1,z2)</code>	<code>eq(z.a,z.b)</code>	booléen

a. Si O est l'origine du plan complexe, on obtient le produit des vecteurs \vec{Oa} et \vec{Ob}

TABLE 31 – Méthodes de la classe de point (complexe).

Méthodes	Application	
<code>conj(z)</code>	<code>z.a : conj()</code>	affixe (conjugate)
<code>mod(z)</code>	<code>z.a : mod()</code>	nombre réel = module <code>z.a</code>
<code>abs(z)</code>	<code>z.a : abs()</code>	nombre réel = module
<code>norm(z)</code>	<code>z.a : norm()</code>	norme (nombre réel)
<code>arg(z)</code>	<code>z.a : arg()</code>	nombre réel = argument de <code>z.a</code> (en rad)
<code>get(z)</code>	<code>z.a : get()</code>	re et im (deux nombres réels)
<code>sqrt(z)</code>	<code>z.a : sqrt()</code>	affixe

La classe est dotée de deux métaméthodes spécifiques.

- La concaténation n'ayant guère de sens ici, l'opération associée à `..` est le produit scalaire ou produit de points. Si $z_1 = a+ib$ et $z_2 = c+id$ alors

$$z_1..z_2 = (a+ib) .. (c+id) = (a+ib) (c-id) = ac+bd + i(bc-ad)$$

Il existe également une fonction mathématique, `dot_product`, qui prend trois arguments. Voir l'exemple 20.9

- Avec la même idée, l'opération associée à `^` est le déterminant, c'est-à-dire

$$z_1 ^ z_2 = (a+ib) ^ (c+id) = ad - bc$$

De $(a-ib) (c+id) = ac+bd + i(ad - bc)$ nous prenons la partie imaginaire.

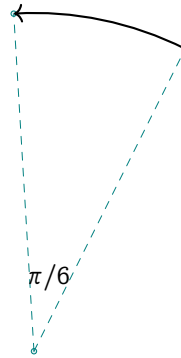
22.3.1 Exemple d'utilisation complexe

Soit $z_a = \text{math.cos}(a) + i \text{math.sin}(a)$. On l'obtient à partir de la bibliothèque en écrivant

```
za = point(math.cos(a),math.sin(a)).
```

Alors $z.B = z.A * z_a$ décrit une rotation du point A d'un angle a .

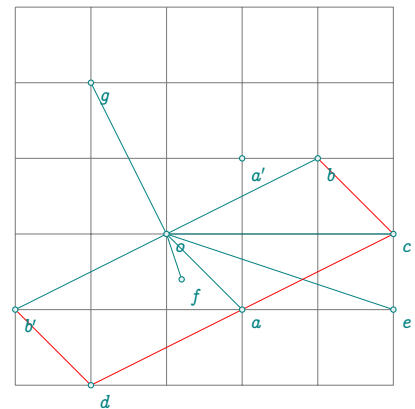
```
\begin{tkzelements}
  z.O = point : new (0,0)
  z.A = point : new (1,2)
  a = math.pi/6
  za = point(math.cos(a),math.sin(a))
  z.B = z.A * za
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(O,A,B)
\tkzDrawArc[->,delta=0](O,A)(B)
\tkzDrawSegments[dashed](O,A O,B)
\tkzLabelAngle(A,O,B){$\pi/6$}
\end{tikzpicture}
```



22.3.2 Opérations avec les points (complexes)

```
\begin{tkzelements}
  z.o = point: new(0,0)
  z.a = point: new(1,-1)
  z.b = point: new(2,1)
  z.bp = -z.b
  z.c = z.a + z.b
  z.d = z.a - z.b
  z.e = z.a * z.b
  z.f = z.a / z.b
  z.ap = point.conj (z.a)
  -- = z.a : conj ()
  z.g = z.b* point(math.cos(math.pi/2),
                  math.sin(math.pi/2))
\end{tkzelements}
```

```
\hspace*{\fill}
\begin{tikzpicture}
\tkzGetNodes
\tkzInit[xmin=-2,xmax=3,ymin=-2,ymax=3]
\tkzGrid
\tkzDrawSegments(o,a o,b o,c o,e o,b' o,f o,g)
\tkzDrawSegments[red](a,c b,c b',d a,d)
\tkzDrawPoints(a,...,g,o,a',b')
\tkzLabelPoints(o,a,b,c,d,e,f,g,a',b')
\end{tikzpicture}
```



22.4 Barycentre

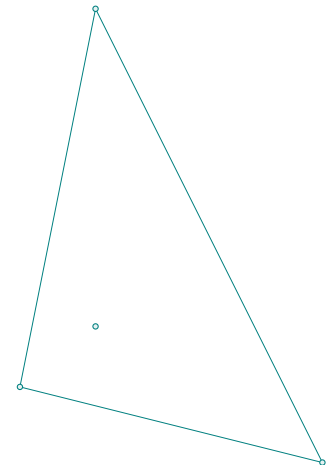
Voici la définition du barycentre, qui est utilisé une quarantaine de fois dans le paquet. `table.pack` construit un tableau à partir d'une liste. `tp.n` donne le nombre de paires. `tp[i][1]` est un affixe et `tp[i][2]` le poids associé (valeur réelle). Voir l'exemple.

```
function barycenter_ (...)
local tp = table.pack(...)
local i
local sum = 0
local weight=0
for i=1,tp.n do
    sum = sum + tp[i][1]*tp[i][2]
    weight = weight + tp[i][2]
end
return sum/weight
end
```

22.4.1 Utilisation du barycentre

```
\begin{tkzelements}
z.A = point: new (1,0)
z.B = point: new (5,-1)
z.C = point: new (2,5)
z.G = barycenter ({z.A,3},{z.B,1},{z.C,1})
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawPoints(A,B,C,G)
\end{tikzpicture}
```



22.4.2 Centre du cercle inscrit d'un triangle

Le calcul des poids k_a , k_b et k_c est précis et le résultat obtenu avec le barycentre est excellent. Notez la présence du trait de soulignement `_` pour certaines fonctions. Ces fonctions sont internes (développeur). Chaque fonction externe (utilisateur) est associée à son équivalent interne.

Voici comment déterminer le centre du cercle inscrit d'un triangle :

```
function in_center_ ( a,b,c )
    local ka = point.abs (b-c)
    local kc = point.abs (b-a)
    local kb = point.abs (c-a)
    return    barycenter_ ( {a,ka} , {b,kb} , {c,kc} )
```

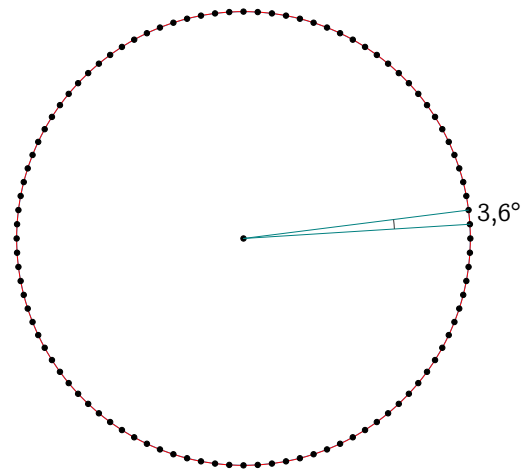
22.5 Notation des boucles et des tableaux

Le problème rencontré dans cet exemple provient de la notation des noms de points. Comme il n'est pas possible d'écrire sous forme simplifiée, nous devons recourir à la notation `table[key]`.

```

\begin{tkzelements}
  local r = 3
  z.0 = point : new (0,0)
  max = 100
  for i = 1,max
  do
    z["A"..i] = point : polar(r,2*i*math.pi/max)
  end
  a = math.deg(get_angle (z.0,z.A_1,z.A_2))
\end{tkzelements}

```



22.6 Méthode in_out

Cette fonction peut être utilisée pour les objets suivants

- droite
- cercle
- triangle
- ellipse

L'objet disque n'existe pas, donc avec `in_out_disk` il est possible de déterminer si un point est dans un disque.

22.6.1 In_out pour une droite

```

function line: in_out (pt)
  local sc,epsilon
  epsilon = 10-12
  sc = math.abs ((pt-self.pa)^(pt-self.pb))
  if sc <= epsilon
  then
    return true
  else
    return false
  end
end

```

Le paquet `ifthen` est nécessaire pour le code ci-dessous.

```

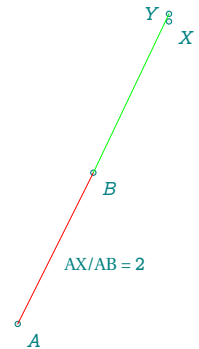
\begin{tkzelements}
z.A = point: new (0,0)
z.B = point: new (1,2)
z.X = point: new (2,4.000)
z.Y = point: new (2,4.1)
L.AB = line : new (z.A,z.B)
if L.AB : in_out (z.X)
  then
    inline = true k = (z.X-z.A)/(z.B-z.A)
  else
    inline = false
  end
  inline_bis = L.AB : in_out (z.Y)
\end{tkzelements}

```

```

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(A,B,X,Y)
\tkzLabelPoints(A,B,X)
\tkzLabelPoints[left](Y)
\ifthenelse{\equal{\tkzUseLua{inline}}{true}}{
  \tkzDrawSegment[red](A,B)
  \tkzLabelSegment(A,B){AX/AB = $\tkzUseLua{k}$}{%
  \tkzDrawSegment[blue](A,B)}
\ifthenelse{\equal{\tkzUseLua{inline_bis}}{false}}{
  \tkzDrawSegment[green](B,Y)}{}
\end{tikzpicture}

```



22.7 Déterminant et produit scalaire

22.7.1 Déterminant

Nous venons de voir comment utiliser \wedge pour obtenir le déterminant associé à deux vecteurs.

`in_out` est simplement une copie de `islinear`.

Voici la définition et la transformation de la puissance d'un nombre complexe.

```

-- déterminant is '^' ad - bc
function point.__pow(z1,z2)
  local z
  z = point.conj(z1) * z2 -- (a-ib) (c+id) = ac+bd + i(ad - bc)
  return z.im
end

```

22.7.2 Produit scalaire

Voici la définition du produit de points entre deux affixes et la transformation de concaténation.

```

-- produit scalaire is '..' result ac + bd
function point.__concat(z1,z2)
  local z
  z = z1 * point.conj(z2) -- (a+ib) (c-id) = ac+bd + i(bc-ad)
  return z.re
end

```

22.7.3 produit scalaire: test d'orthogonalité

Voici une fonction `isortho` qui permet de tester l'orthogonalité entre deux vecteurs.

```
function isortho (z1,z2,z3)
  local epsilon
  local dp
  epsilon = 10(-8)
  dp = (z2-z1) .. (z3-z1)
  if math.abs(dp) < epsilon
    then
      return true
    else
      return false
    end
  end
end
```

22.7.4 produit scalaire: projection

La projection d'un point sur une droite est une fonction fondamentale, dont la définition est la suivante :

```
function projection_ ( pa,pb,pt )
  local v
  local z
  if aligned ( pa,pb,pt ) then
    return pt
  else
    v = pb - pa
    z = ((pt - pa)..v)/(point.norm(v)) -- .. dot product
    return pa + z * v
  end
end
```

La fonction `aligned` est équivalente à `islinear` mais n'utilise pas de déterminant. Elle sera remplacée dans une prochaine version.

22.8 Méthode point

La méthode du point est une méthode pour de nombreux objets :

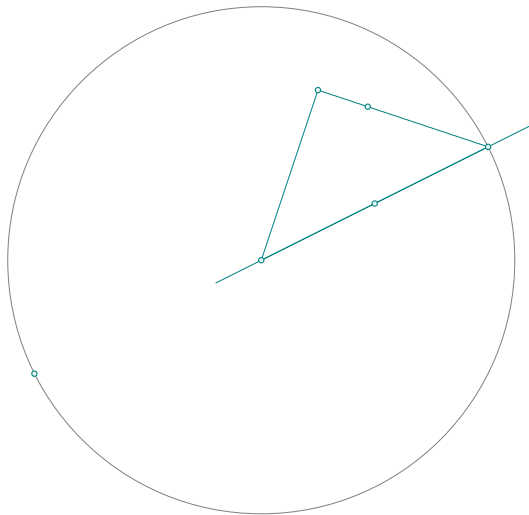
- droite ,
- cercle,
- ellipse,
- triangle.

Vous obtenez un point sur l'objet en entrant un nombre réel entre 0 et 1.


```

\begin{tkzelements}
  z.A = point : new ( 0 , 0 )
  z.B = point : new ( 4 , 2 )
  z.C = point : new ( 1 , 3 )
  L.AB = line : new (z.A,z.B)
  C.AB = circle : new (z.A,z.B)
  T.ABC = triangle : new (z.A,z.B,z.C)
  z.I = L.AB : point (0.5)
  z.J = C.AB : point (0.5)
  z.K = T.ABC : point (0.5)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLine(A,B)
  \tkzDrawCircle(A,B)
  \tkzDrawPolygon(A,B,C)
  \tkzDrawPoints(A,B,C,I,J,K)
\end{tikzpicture}

```



22.9 Derrière les objets

Avant d'introduire les objets, j'utilisais uniquement des fonctions dont les paramètres étaient des points (nombres complexes).

Par exemple, $z.m = \text{midpoint}_ (z.a, z.b)$ définit le milieu des points a et b . Avec les objets, on définit d'abord la ligne/segment $L.ab$ puis on obtient le milieu avec $z.m = L.ab.\text{mid}$.

J'ai conservé les fonctions (que je vais appeler "primaires") dont les seuls arguments sont des points. Elles se distinguent des autres par un trait de soulignement terminal. En fait, toutes (ou presque) les fonctions liées aux objets dépendent d'une fonction primaire.

Nous venons de voir le cas du milieu d'un point, donc examinons deux autres cas :

- Rotation autour d'un point. c est le centre de rotation, a est l'angle et pt est le point à affecter. Par exemple :
 $z.Mp = \text{rotation} (z.A, \text{math.pi}/6, z.M)$

```

function rotation_ (c,a,pt)
  local z = point( math.cos(a) , math.sin(a) )
  return z*(pt-c)+c
end

```

Avec les objets, cela donne $z.Mp = z.A : \text{rotation} (\text{math.pi}/6, z.M)$

- L'intersection d'une ligne et d'un cercle est obtenue en utilisant $\text{intersection_lc}_ (z.A, z.B, z.O, z.T)$. en utilisant la ligne droite (A, B) et le cercle $C(O, T)$.

Cela donnera les objets : $\text{intersection} (L.AB, C.OT)$

La différence est que la programmation est plus directe avec les fonctions primaires et un peu plus efficace, mais perd en visibilité.

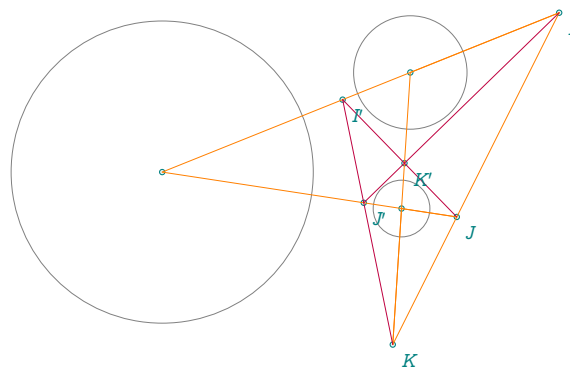
23 Exemples

23.1 D'Alembert 1

```

\begin{tkzelements}
  z.A = point : new (0,0)
  z.a = point : new (4,0)
  z.B = point : new (7,-1)
  z.b = point : new (5.5,-1)
  z.C = point : new (5,-4)
  z.c = point : new (4.25,-4)
  C.Aa = circle : new (z.A,z.a)
  C.Bb = circle : new (z.B,z.b)
  C.Cc = circle : new (z.C,z.c)
  z.I = C.Aa : external_similitude (C.Bb)
  z.J = C.Aa : external_similitude (C.Cc)
  z.K = C.Cc : external_similitude (C.Bb)
  z.Ip = C.Aa : internal_similitude (C.Bb)
  z.Jp = C.Aa : internal_similitude (C.Cc)
  z.Kp = C.Cc : internal_similitude (C.Bb)
\end{tkzelements}
\begin{tikzpicture}[rotate=-60]
  \tkzGetNodes
  \tkzDrawCircles(A,a B,b C,c)
  \tkzDrawPoints(A,B,C,I,J,K,I',J',K')
  \tkzDrawSegments[new](I,K A,I A,J B,I B,K C,J C,K)
  \tkzDrawSegments[purple](I,J' I',J I',K)
  \tkzLabelPoints(I,J,K,I',J',K')
\end{tikzpicture}

```

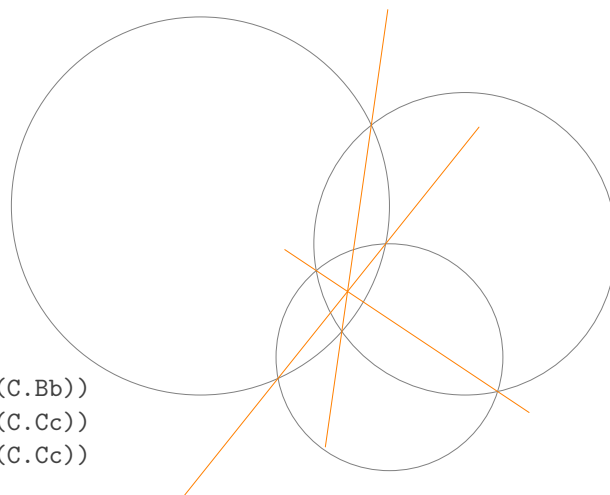


23.2 D'Alembert 2

```

\begin{tkzelements}
  scale = .75
  z.A = point : new (0,0)
  z.a = point : new (5,0)
  z.B = point : new (7,-1)
  z.b = point : new (3,-1)
  z.C = point : new (5,-4)
  z.c = point : new (2,-4)
  C.Aa = circle : new (z.A,z.a)
  C.Bb = circle : new (z.B,z.b)
  C.Cc = circle : new (z.C,z.c)
  z.i,z.j = get_points (C.Aa : radical_axis (C.Bb))
  z.k,z.l = get_points (C.Aa : radical_axis (C.Cc))
  z.m,z.n = get_points (C.Bb : radical_axis (C.Cc))
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles(A,a B,b C,c)
  \tkzDrawLines[new](i,j k,l m,n)
\end{tikzpicture}

```

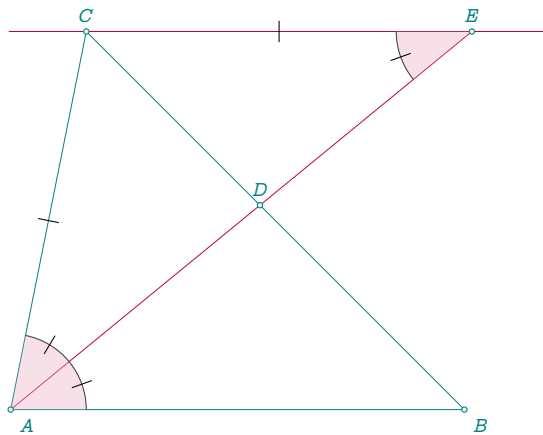


23.3 Alternance

```

\begin{tkzelements}
z.A = point: new (0 , 0)
z.B = point: new (6 , 0)
z.C = point: new (1 , 5)
T = triangle: new (z.A,z.B,z.C)
z.I = T.incenter
L.AI = line: new (z.A,z.I)
z.D = intersection (L.AI,T.bc)
L.LLC = T.ab: ll_from (z.C)
z.E = intersection (L.AI,L.LLC)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawLine[purple](C,E)
\tkzDrawSegment[purple](A,E)
\tkzFillAngles[purple!30,opacity=.4](B,A,C C,E,D)
\tkzMarkAngles[mark=|](B,A,D D,A,C C,E,D)
\tkzDrawPoints(A,...,E)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D,E)
\tkzMarkSegments(A,C C,E)
\end{tikzpicture}

```



23.4 Cercle d'Apollonius

```

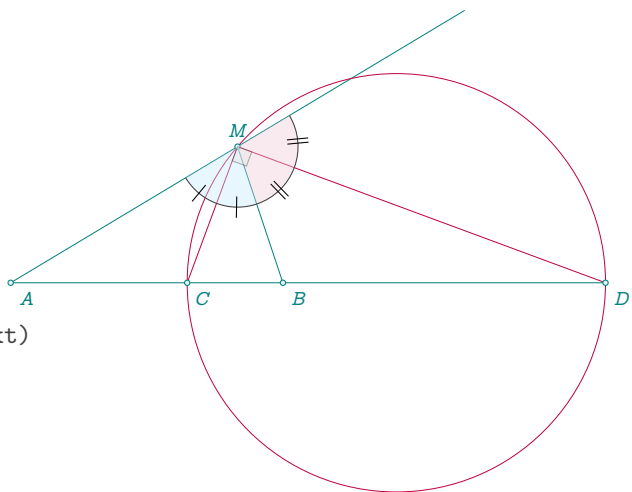
\begin{tkzelements}
scale=.75
z.A = point: new (0 , 0)
z.B = point: new (6 , 0)
z.M = point: new (5 , 3)
T.MAB = triangle : new (z.M,z.A,z.B)
L.bis = T.MAB : bisector ()
z.C = L.bis.pb
L.bisext = T.MAB : bisector_ext ()
z.D = intersection (T.MAB.bc, L.bisext)
L.CD = line: new (z.C,z.D)
z.O = L.CD.mid
L.AM = T.MAB.ab
z.E = z.M : symmetry (z.A)
\end{tkzelements}

```

```

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawSegment[add=0 and 1](A,M)
\tkzDrawSegments[purple](M,C M,D)
\tkzDrawCircle[purple](O,C)
\tkzDrawSegments(A,B B,M D,B)
\tkzDrawPoints(A,B,M,C,D)
\tkzLabelPoints[below right](A,B,C,D)
\tkzLabelPoints[above](M)
\tkzFillAngles[opacity=.4,cyan!20](A,M,B)
\end{tikzpicture}

```



```

\tkzFillAngles[opacity=.4,purple!20](B,M,E)
\tkzMarkRightAngle[opacity=.4,fill=gray!20](C,M,D)
\tkzMarkAngles[mark=|](A,M,C C,M,B)
\tkzMarkAngles[mark=||](B,M,D D,M,E)
\end{tikzpicture}

```

Remarque : Le cercle peut être obtenu avec :

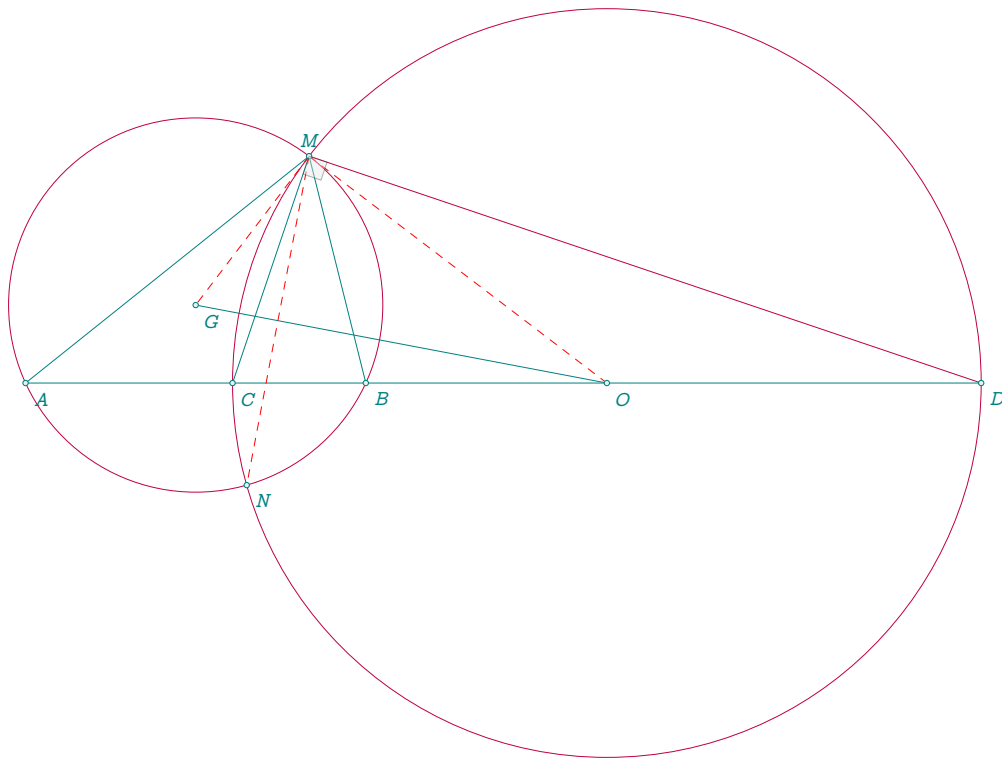
$C.AB = T.MAB.bc$: apollonius ($\text{length}(z.M,z.A)/\text{length}(z.M,z.B)$)

23.5 Apollonius et le cercle circonscrit

```

\begin{tkzelements}
  scale =.75
  z.A = point: new (0 , 0)
  z.B = point: new (6 , 0)
  z.M = point: new (5 , 4)
  T.AMB = triangle: new (z.A,z.M,z.B)
  L.AB = T.AMB.ca
  z.I = T.AMB.incenter
  L.MI = line: new (z.M,z.I)
  z.C = intersection (L.AB , L.MI)
  L.MJ = L.MI: ortho_from (z.M)
  z.D = intersection (L.AB , L.MJ)
  L.CD = line: new (z.C,z.D)
  z.O = L.CD.mid
  z.G = T.AMB.circumcenter
  C.GA = circle: new (z.G,z.A)
  C.OC = circle: new (z.O,z.C)
  _,z.N = intersection (C.GA , C.OC)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,M)
  \tkzDrawCircles[purple](O,C G,A)
  \tkzDrawSegments[purple](M,D)
  \tkzDrawSegments(D,B O,G M,C)
  \tkzDrawSegments[red,dashed](M,N M,O M,G)
  \tkzDrawPoints(A,B,M,C,D,N,O,G)
  \tkzLabelPoints[below right](A,B,C,D,N,O,G)
  \tkzLabelPoints[above](M)
  \tkzMarkRightAngle[opacity=.4,fill=gray!20](C,M,D)
\end{tikzpicture}

```



23.6 Cercles d'Apollonius dans un triangle

```

\begin{tkzelements}
  z.A = point: new (0 , 0)
  z.B = point: new (6 , 0)
  z.C = point: new (4.5 , 1)
  T.ABC = triangle: new (z.A,z.B,z.C)
  z.I = T.ABC.incenter
  z.O = T.ABC.circumcenter
  L.CI = line: new (z.C,z.I)
  z.Cp = intersection (T.ABC.ab , L.CI)
  z.x = L.CI.north_pa
  L.Cx = line: new (z.C,z.x)
  z.R = intersection (L.Cx,T.ABC.ab)
  L.CpR = line: new (z.Cp,z.R)
  z.O1 = L.CpR.mid
  L.AI = line: new (z.A,z.I)
  z.Ap = intersection (T.ABC.bc , L.AI)
  z.y = L.AI.north_pa
  L.Ay = line: new (z.A,z.y)
  z.S = intersection (L.Ay,T.ABC.bc)
  L.ApS = line: new (z.Ap,z.S)
  z.O2 = L.ApS.mid
  L.BI = line: new (z.B,z.I)
  z.Bp = intersection (T.ABC.ca , L.BI)
  z.z = L.BI.north_pa
  L.Bz = line: new (z.B,z.z)
  z.T = intersection (L.Bz,T.ABC.ca)
  L.Bpt = line: new (z.Bp,z.T)
  z.O3 = L.Bpt.mid

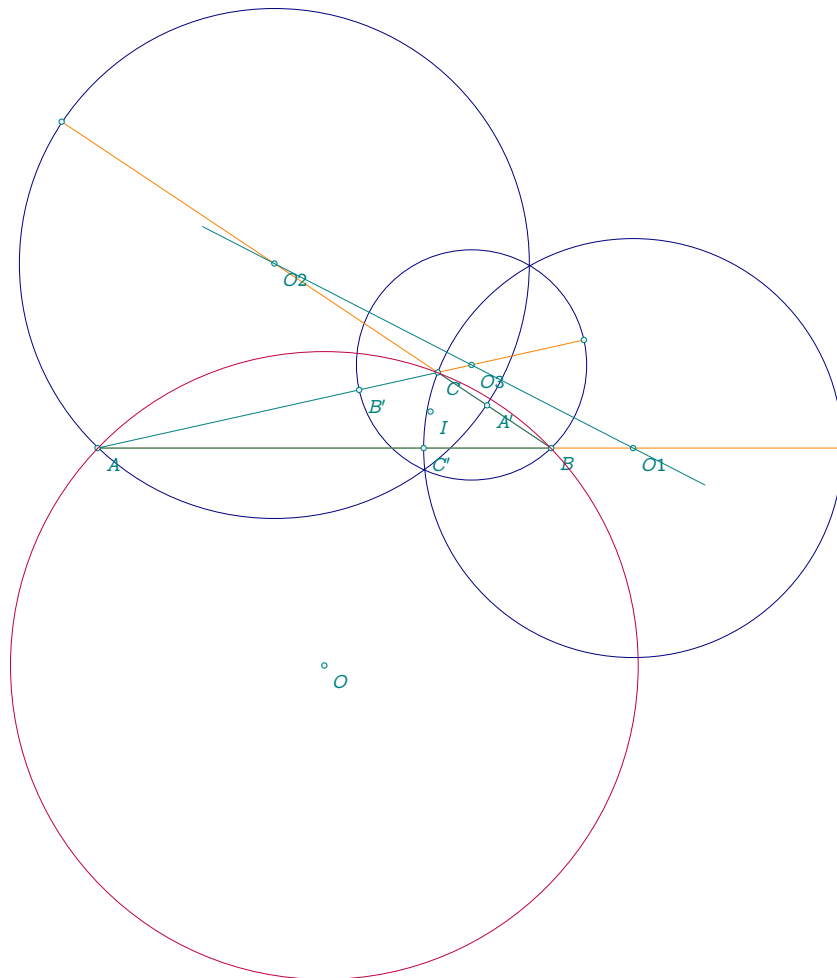
```

```

\end{tkzelements}

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles[blue!50!black](O1,C' O2,A' O3,B')
  \tkzDrawSegments[new](B,S C,T A,R)
  \tkzDrawPolygon(A,B,C)
  \tkzDrawPoints(A,B,C,A',B',C',O,I,R,S,T,O1,O2,O3)
  \tkzLabelPoints(A,B,C,A',B',C',O,I)
  \tkzLabelPoints(O1,O2,O3)
  \tkzDrawCircle[purple](O,A)
  \tkzDrawLine(O1,O2)
\end{tikzpicture}

```



Même résultat en utilisant la fonction T.ABC.ab : apollonius (k)

```

\begin{tkzelements}
  scale      = .75
  z.A        = point: new (0 , 0)
  z.B        = point: new (6 , 0)
  z.C        = point: new (4.5 , 1)
  T.ABC      = triangle : new (z.A,z.B,z.C)
  z.O        = T.ABC.circumcenter
  C.AB       = T.ABC.ab : apollonius (length(z.C,z.A)/length(z.C,z.B))
  z.w1,z.t1  = get_points ( C.AB )

```

```

C.AC      = T.ABC.ca : apollonius (length(z.B,z.C)/length(z.B,z.A))
z.w2,z.t2 = get_points ( C.AC )
C.BC      = T.ABC.bc : apollonius (length(z.A,z.B)/length(z.A,z.C))
z.w3,z.t3 = get_points ( C.BC )
\end{tkzelements}

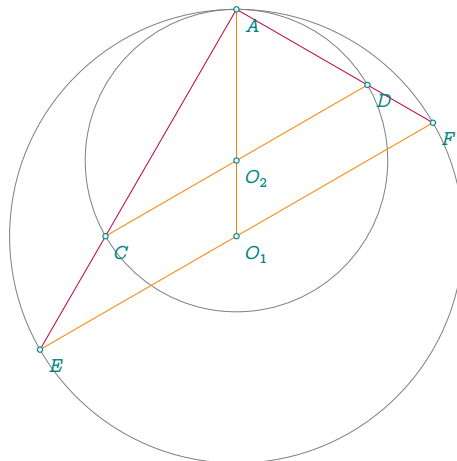
```

23.7 Archimède

```

\begin{tkzelements}
z.O_1 = point: new (0, 0)
z.O_2 = point: new (0, 1)
z.A = point: new (0, 3)
z.F = point: polar (3, math.pi/6)
L = line: new (z.F,z.O_1)
C = circle: new (z.O_1,z.A)
z.E = intersection (L,C)
T = triangle: new (z.F,z.E,z.O_2)
z.x = T: parallelogram ()
L = line: new (z.x,z.O_2)
C = circle: new (z.O_2,z.A)
z.C,z.D = intersection (L ,C)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(O_1,A O_2,A)
\tkzDrawSegments[new](O_1,A E,F C,D)
\tkzDrawSegments[purple](A,E A,F)
\tkzDrawPoints(A,O_1,O_2,E,F,C,D)
\tkzLabelPoints(A,O_1,O_2,E,F,C,D)
\end{tikzpicture}

```



23.8 Le cercle de Bankoff

```

\begin{tkzelements}
z.A = point: new (0 , 0)
z.B = point: new (10 , 0)
L.AB = line : new (z.A,z.B)
z.C = L.AB: gold_ratio ()
L.AC = line : new (z.A,z.C)
L.CB = line : new (z.C,z.B)
z.O_0 = L.AB.mid
z.O_1 = L.AC.mid
z.O_2 = L.CB.mid
C.O0B = circle : new (z.O_0,z.B)
C.O1C = circle : new (z.O_1,z.C)
C.O2C = circle : new (z.O_2,z.B)
z.Pp = C.O0B : midarc (z.B,z.A)
z.P = C.O1C : midarc (z.C,z.A)
z.Q = C.O2C : midarc (z.B,z.C)
L.O102 = line : new (z.O_1,z.O_2)
L.O001 = line : new (z.O_0,z.O_1)
L.O002 = line : new (z.O_0,z.O_2)
z.M_0 = L.O102 : harmonic_ext (z.C)

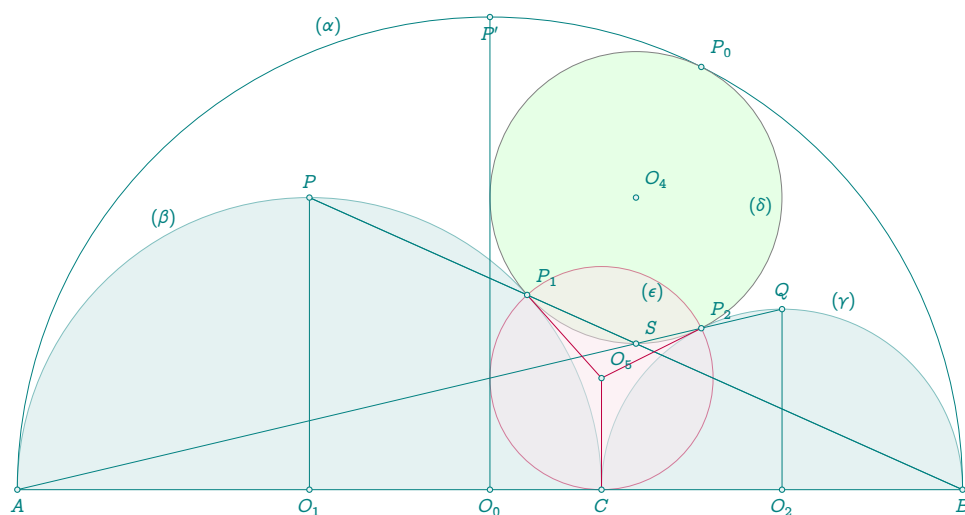
```

```

z.M_1 = L.O001 : harmonic_int (z.A)
z.M_2 = L.O002 : harmonic_int (z.B)
L.BP = line : new (z.B,z.P)
L.AQ = line : new (z.A,z.Q)
z.S = intersection (L.BP,L.AQ)
L.Pp00 = line : new (z.Pp,z.O_0)
L.PC = line : new (z.P,z.C)
z.Ap = intersection (L.Pp00,L.PC)
L.CS = line : new (z.C,z.S)
C.M1A = circle : new (z.M_1,z.A)
C.M2B = circle : new (z.M_2,z.B)
z.P_0 = intersection (L.CS,C.O0B)
z.P_1 = intersection (C.M2B,C.O1C)
z.P_2 = intersection (C.M1A,C.O2C)
T.P0P1P2 = triangle : new (z.P_0,z.P_1,z.P_2)
z.O_4 = T.P0P1P2.circumcenter
T.CP1P2 = triangle : new (z.C,z.P_1,z.P_2)
z.O_5 = T.CP1P2.circumcenter
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawSemiCircles[teal](O_0,B)
\tkzDrawSemiCircles[teal,fill=teal!20,opacity=.5](O_1,C O_2,B)
\tkzDrawCircle[fill=green!10](O_4,P_0)
\tkzDrawCircle[purple,fill=purple!10,opacity=.5](O_5,C)
\tkzDrawSegments(A,B O_0,P' B,P A,Q)
\tkzDrawSegments(P,B Q,O_2 P,O_1)
\tkzDrawSegments[purple](O_5,P_2 O_5,P_1 O_5,C)
\tkzDrawPoints(A,B,C,P_0,P_2,P_1,O_0,O_1,O_2,O_4,O_5,Q,P,P',S)
\tkzLabelPoints[below](A,B,C,O_0,O_1,O_2,P')
\tkzLabelPoints[above](Q,P)
\tkzLabelPoints[above right](P_0,P_2,P_1,O_5,O_4,S)
\begin{scope}[font=\scriptsize]
\tkzLabelCircle[above](O_1,C)(120){\beta}
\tkzLabelCircle[above](O_2,B)(70){\gamma}
\tkzLabelCircle[above](O_0,B)(110){\alpha}
\tkzLabelCircle[left](O_4,P_2)(60){\delta}
\tkzLabelCircle[left](O_5,C)(140){\epsilon}
\end{scope}
\end{tikzpicture}

```

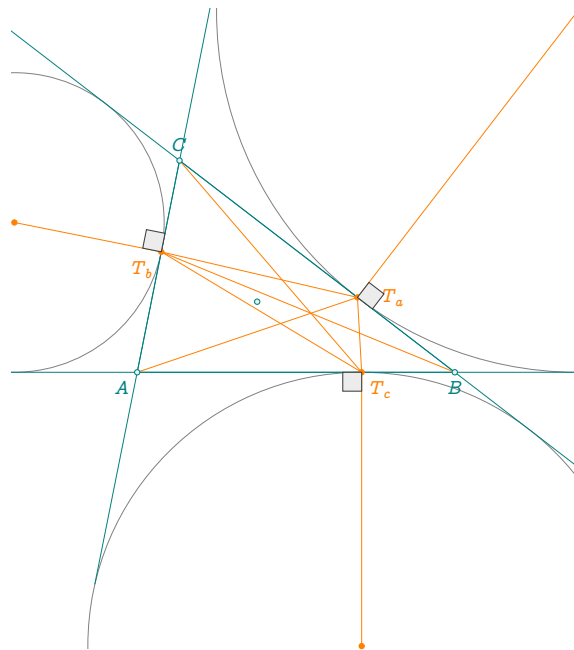



23.9 Cercles exinscrits

```

\begin{tkzelements}
  scale = 0.7
  z.A = point: new (0,0)
  z.B = point: new (6,0)
  z.C = point: new (.8,4)
  T = triangle: new ( z.A, z.B, z.C)
  z.K = T.centroid
  z.J_a,z.J_b,z.J_c = get_points (T: excentral())
  z.T_a,z.T_b,z.T_c = get_points (T: extouch())
  la = line: new ( z.A, z.T_a)
  lb = line: new ( z.B, z.T_b)
  z.G = intersection (la,lb)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPoints[new] (J_a,J_b,J_c)
  \tkzClipBB
  \tkzDrawCircles[gray] (J_a,T_a J_b,T_b J_c,T_c)
  \tkzDrawLines[add=1 and 1] (A,B B,C C,A)
  \tkzDrawSegments[new] (A,T_a B,T_b C,T_c)
  \tkzDrawSegments[new] (J_a,T_a J_b,T_b J_c,T_c)
  \tkzDrawPolygon(A,B,C)
  \tkzDrawPolygon[new] (T_a,T_b,T_c)
  \tkzDrawPoints(A,B,C,K)
  \tkzDrawPoints[new] (T_a,T_b,T_c)
  \tkzLabelPoints[below left] (A)
  \tkzLabelPoints[below] (B)
  \tkzLabelPoints[above] (C)
  \tkzLabelPoints[new,below left] (T_b)
  \tkzLabelPoints[new,below right] (T_c)
  \tkzLabelPoints[new,right=6pt] (T_a)
  \tkzMarkRightAngles[fill=gray!15] (J_a,T_a,B J_b,T_b,C J_c,T_c,A)
\end{tikzpicture}

```

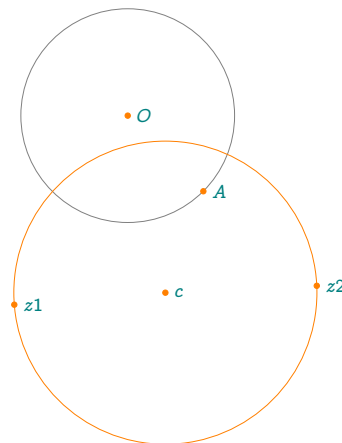


23.10 Orthogonal_through

```

\begin{tkzelements}
  z.O = point: new (0,1)
  z.A = point: new (1,0)
  z.z1 = point: new (-1.5,-1.5)
  z.z2 = point: new (2.5,-1.25)
  C.OA = circle: new (z.O,z.A)
  C = C.OA: orthogonal_through (z.z1,z.z2)
  z.c = C.center
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircle(O,A)
  \tkzDrawCircle[new](c,z1)
  \tkzDrawPoints[new](O,A,z1,z2,c)
  \tkzLabelPoints[right](O,A,z1,z2,c)
\end{tikzpicture}

```



23.11 Rapport divin

```

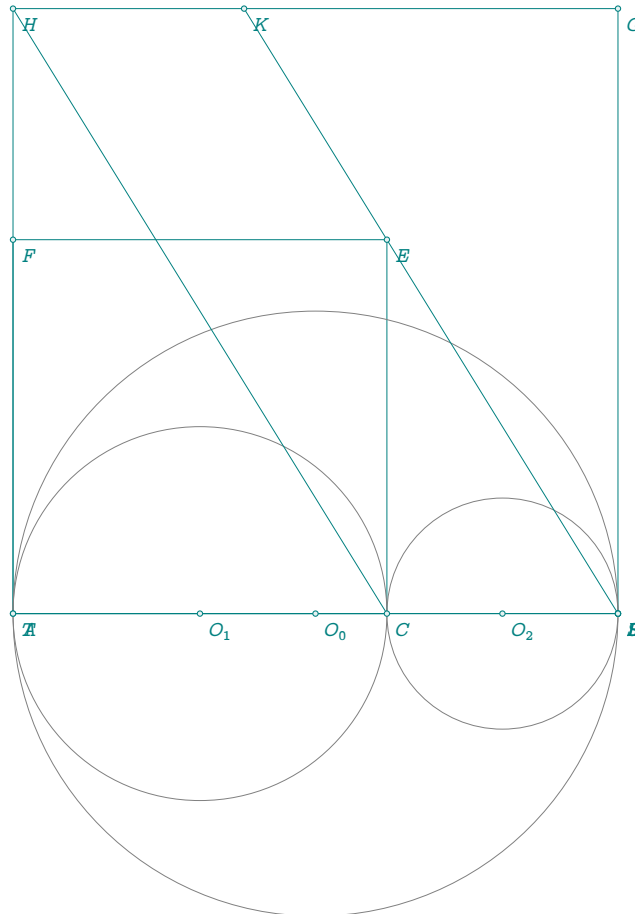
\begin{tkzelements}
z.A = point: new (0 , 0)
z.B = point: new (8 , 0)
L.AB = line: new (z.A,z.B)
z.C = L.AB: gold_ratio ()
L.AC = line: new (z.A,z.C)
z.O_1 = L.AC.mid
_,_,z.G,z.H = get_points(L.AB: square ())
_,_,z.E,z.F = get_points(L.AC: square ())
L.CB = line: new (z.C,z.B)
z.O_2 = L.CB.mid
z.O_0 = L.AB.mid
L.BE = line: new (z.B,z.E)

```

```

L.GH      = line: new (z.G,z.H)
z.K       = intersection (L.BE,L.GH)
C0        = circle: new (z.O_0,z.B)
z.R,_     = intersection (L.BE,C0)
C2        = circle: new (z.O_2,z.B)
z.S,_     = intersection (L.BE,C2)
L.AR      = line: new (z.A,z.R)
C1        = circle: new (z.O_1,z.C)
_,z.T     = intersection (L.AR,C1)
L.BG      = line: new (z.B,z.G)
z.L       = intersection (L.AR,L.BG)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygons(A,C,E,F A,B,G,H)
\tkzDrawCircles(O_1,C O_2,B O_0,B)
\tkzDrawSegments(H,C B,K A,L)
\tkzDrawPoints(A,B,C,K,E,F,G,H,O_0,O_1,O_2,R,S,T,L)
\tkzLabelPoints(A,B,C,K,E,F,G,H,O_0,O_1,O_2,R,S,T,L)
\end{tikzpicture}

```

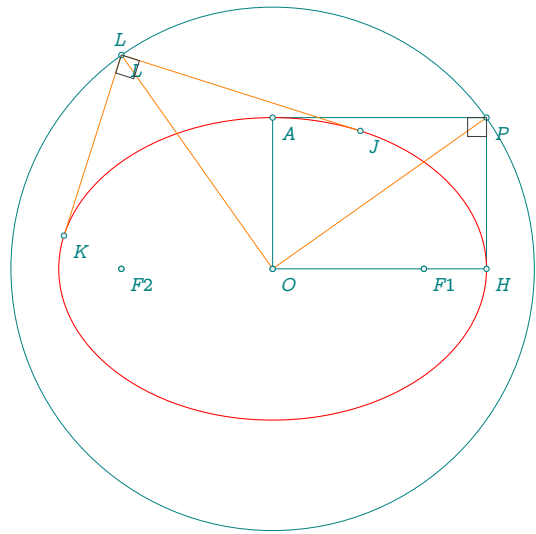


23.12 Cercle directeur

```

\begin{tkzelements}
  scale      = .5
  z.O        = point: new (0 , 0)
  z.F1       = point: new (4 , 0)
  z.F2       = point: new (-4 , 0)
  z.H        = point: new (4* $\sqrt{2}$  , 0)
  E          = ellipse: foci (z.F2,z.F1,z.H)
  a,b       = E.Rx, E.Ry
  z.A        = E.covertex
  T          = triangle: new (z.H,z.O,z.A)
  z.P        = T: parallelogram ()
  C          = circle: new (z.O,z.P)
  z.L        = C: point (0.25)
  L.J,L.K    = E: tangent_from (z.L)
  z.J        = L.J.pb
  z.K        = L.K.pb
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPoints(F1,F2,O)
  \tkzDrawCircles[teal](O,P)
  \tkzDrawPolygon(H,O,A,P)
  \tkzDrawEllipse[red](O,\tkzUseLua{a},\tkzUseLua{b},0)
  \tkzDrawSegments[orange](O,P O,L L,J L,K)
  \tkzDrawPoints(F1,F2,O,H,A,P,L,J,K)
  \tkzLabelPoints(F1,F2,O,H,A,P,L,J,K)
  \tkzLabelPoints[above](L)
  \tkzMarkRightAngles(A,P,H J,L,K)
\end{tikzpicture}

```



23.13 Division or

```

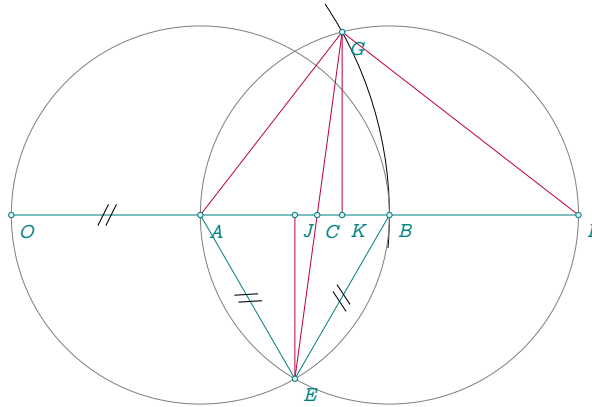
\begin{tkzelements}
  z.A        = point: new (0,0)
  z.B        = point: new (2.5,0)
  L.AB       = line: new (z.A,z.B)
  C.AB       = circle: new (z.A,z.B)
  C.BA       = circle: new (z.B,z.A)
  z.J        = L.AB: midpoint ()
  L.JB       = line:new (z.J,z.B)
  z.F,z.E    = intersection (C.AB , C.BA)
  z.I,_      = intersection (L.AB , C.BA)
  z.K        = L.JB : midpoint ()
  L.mediator = L.JB: mediator ()
  z.G        = intersection (L.mediator,C.BA)
  L.EG       = line:new (z.E,z.G)
  z.C        = intersection (L.EG,L.AB)
  z.O        = C.AB: antipode (z.B)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes

```

```

\tkzDrawArc[delta=5](O,B)(G)
\tkzDrawCircles(A,B B,A)
\tkzDrawSegments(A,E B,E O,I)
\tkzDrawSegments[purple](J,E A,G G,I K,G E,G)
\tkzMarkSegments[mark=s||](A,E B,E O,A)
\tkzDrawPoints(A,B,C,E,I,J,G,O,K)
\tkzLabelPoints(A,B,C,E,I,J,G,O,K)
\end{tikzpicture}

```

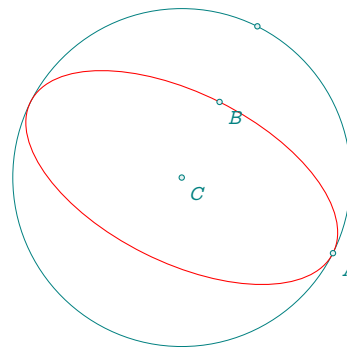


23.14 Ellipse

```

\begin{tkzelements}
  z.C      = point: new (3 , 2)
  z.A      = point: new (5 , 1)
  L.CA     = line : new (z.C,z.A)
  z.b      = L.CA.north_pa
  L        = line : new (z.C,z.b)
  z.B      = L : point (0.5)
  E        = ellipse: new (z.C,z.A,z.B)
  a        = E.Rx
  b        = E.Ry
  slope    = math.deg(E.slope)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles[teal](C,A)
  \tkzDrawEllipse[red](C,\tkzUseLua{a},\tkzUseLua{b},\tkzUseLua{slope})
  \tkzDrawPoints(C,A,B,b)
  \tkzLabelPoints(C,A,B)
\end{tikzpicture}

```

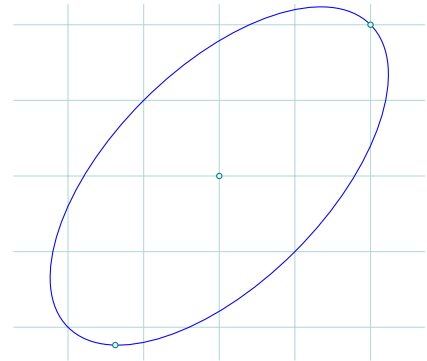


23.15 Ellipse avec rayons

```

\begin{tkzelements}
scale=.5
z.C = point: new (0 , 4)
b = value(math.sqrt(8))
a = value(math.sqrt(32))
ang = math.deg(math.pi/4)
E = ellipse: radii (z.C,a,b,math.pi/4)
z.V = E : point (0)
z.CoV = E : point (math.pi/2)
\end{tkzelements}
\begin{tikzpicture}[gridded]
\tkzGetNodes
\tkzDrawEllipse[blue](C,\tkzUseLua{a},
\tkzUseLua{b},\tkzUseLua{ang})
\tkzDrawPoints(C,V,CoV)
\end{tikzpicture}

```

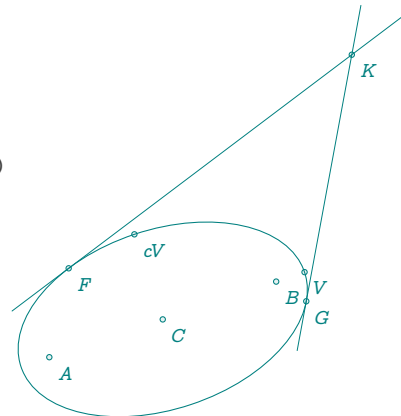


23.16 Ellipse with foci

```

\begin{tkzelements}
local e
e = .8
z.A = point: new (2 , 3)
z.B = point: new (5 , 4)
z.K = point: new (6, 7)
L.AB = line: new (z.A,z.B)
z.C = L.AB.mid
c = point.abs(z.B-z.C)
a = c/e
b = math.sqrt (a^2-c^2)
z.V = z.C + a*(z.B-z.C)/point.abs(z.B-z.C)
E = ellipse: foci (z.A,z.B,z.V)
z.cV = E.covertex
ang = math.deg(E.slope)
L.ta,L.tb = E: tangent_from (z.K)
z.F = L.ta.pb
z.G = L.tb.pb
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(A,B,C,K,F,G,V,cV)
\tkzLabelPoints(A,B,C,K,F,G,V,cV)
\tkzDrawEllipse[teal](C,\tkzUseLua{a},\tkzUseLua{b},\tkzUseLua{ang})
\tkzDrawLines(K,F K,G)
\end{tikzpicture}

```



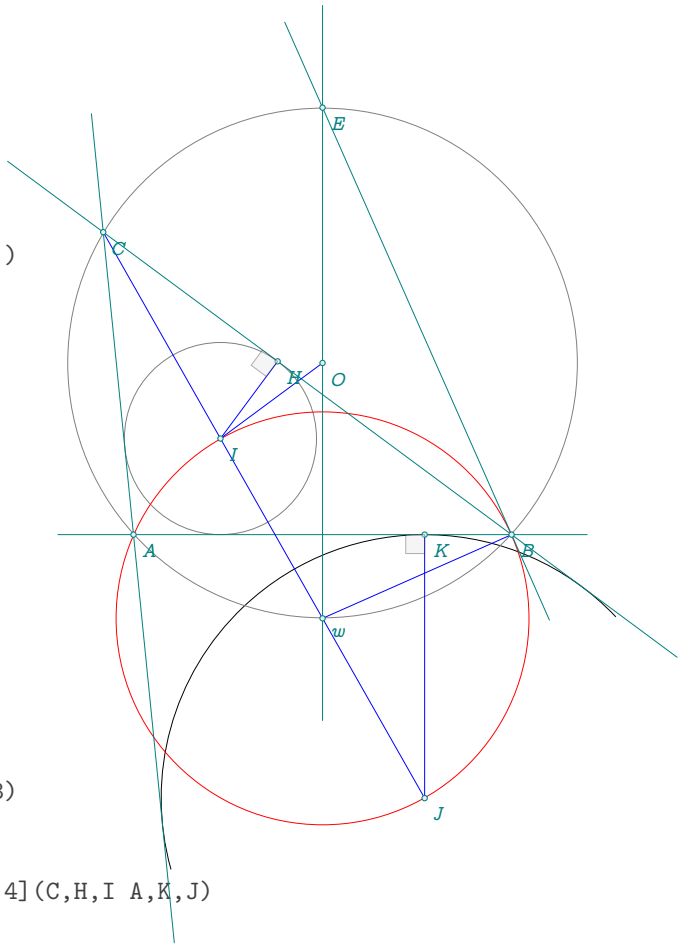
23.17 Relation d'Euler

```

\begin{tkzelements}
  scale      = .75
  z.A        = point: new (0 , 0)
  z.B        = point: new (5 , 0)
  z.C        = point: new (-.4 , 4)
  T.ABC      = triangle: new (z.A,z.B,z.C)
  z.J,z.K    = get_points(T.ABC: ex_circle (2))
  z.X,z.Y,z.K= T.ABC : projection (z.J)
  z.I,z.H    = get_points(T.ABC : in_circle())
  z.O        = T.ABC.circumcenter
  C.OA       = circle : new (z.O,z.A)
  T.IBA      = triangle: new (z.I,z.B,z.A)
  z.w        = T.IBA.circumcenter
  L.Ow       = line : new (z.O,z.w)
  _,z.E      = intersection (L.Ow, C.OA)
\end{tkzelements}

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawArc(J,X)(Y)
  \tkzDrawCircles(I,H O,A)
  \tkzDrawCircle[red](w,I)
  \tkzDrawLines(Y,C A,B X,C E,w E,B)
  \tkzDrawSegments[blue](J,C J,K I,H I,O w,B)
  \tkzDrawPoints(A,B,C,I,J,E,w,H,K,O)
  \tkzLabelPoints(A,B,C,J,I,w,H,K,E,O)
  \tkzMarkRightAngles[fill=gray!20,opacity=.4](C,H,I A,K,J)
\end{tikzpicture}

```



23.18 Angle externe

```

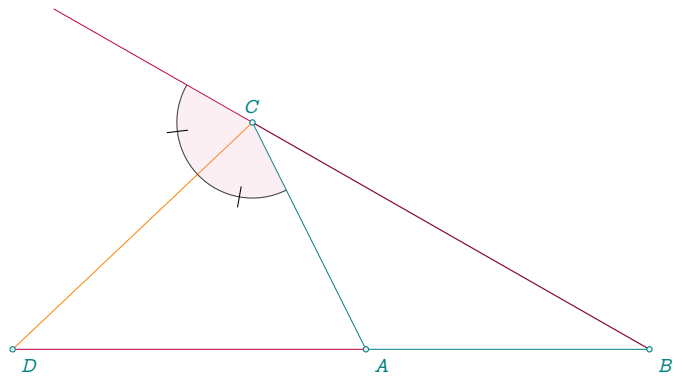
\begin{tkzelements}
  scale = .75
  z.A = point: new (0 , 0)
  z.B = point: new (5 , 0)
  z.C = point: new (-2 , 4)
  T.ABC = triangle: new (z.A,z.B,z.C)
  T.ext = T.ABC: excentral ()
  z.O = T.ABC.circumcenter
  z.D = intersection (T.ext.ab,T.ABC.ab)
  z.E = z.C: symmetry (z.B)
\end{tkzelements}

```

```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawLine[purple,add=0 and .5](B,C)
  \tkzDrawSegment[purple](A,D)
  \tkzDrawSegment[orange](C,D)
  \tkzFillAngles[purple!30,opacity=.2](D,C,A E,C,D)
  \tkzMarkAngles[mark=|](D,C,A E,C,D)
  \tkzDrawPoints(A,...,D)
  \tkzLabelPoints[above](C)
  \tkzLabelPoints(A,B,D)
\end{tikzpicture}

```



23.19 Angle interne

```

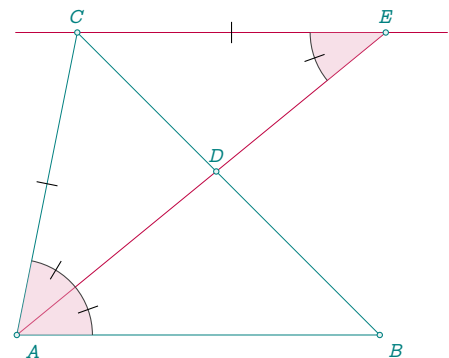
\begin{tkzelements}
  scale = .8
  z.A = point: new (0 , 0)
  z.B = point: new (6 , 0)
  z.C = point: new (1 , 5)
  T = triangle: new (z.A,z.B,z.C)
  z.I = T.incenter
  L.AI = line: new (z.A,z.I)
  z.D = intersection (L.AI, T.bc)
  L.LL = T.ab: ll_from (z.C)
  L.AD = line: new (z.A,z.D)
  z.E = intersection (L.LL,L.AD)
\end{tkzelements}

```

```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawLine[purple](C,E)
  \tkzDrawSegment[purple](A,E)
  \tkzFillAngles[purple!30,opacity=.4](B,A,C C,E,D)
  \tkzMarkAngles[mark=|](B,A,D D,A,C C,E,D)
  \tkzDrawPoints(A,...,E)
  \tkzLabelPoints(A,B)
  \tkzLabelPoints[above](C,D,E)
  \tkzMarkSegments(A,C C,E)
\end{tikzpicture}

```

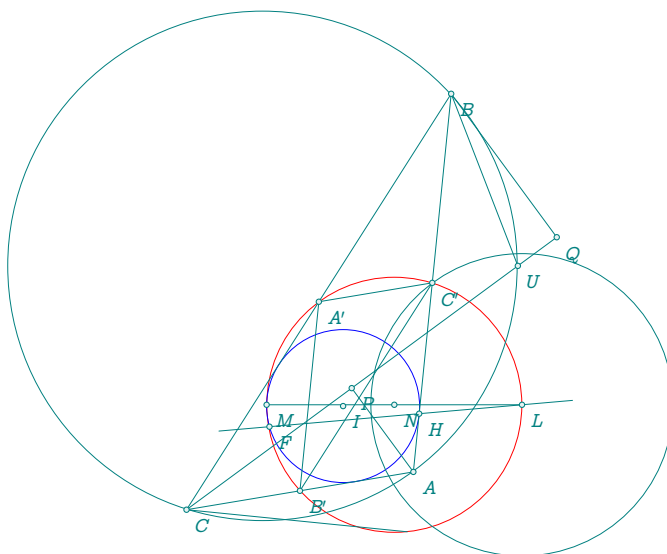


23.20 Théorème de Feuerbach

```

\begin{tkzelements}
  scale      = 1.5
  z.A        = point: new (0 , 0)
  z.B        = point: new (5 , -.5)
  z.C        = point: new (-.5 , 3)
  T.ABC      = triangle: new (z.A,z.B,z.C)
  z.O        = T.ABC.circumcenter
  z.N        = T.ABC.eulercenter
  z.I,z.K    = get_points(T.ABC: in_circle())
  z.H        = T.ABC.ab : projection (z.I)
  z.Ap,
  z.Bp,
  z.Cp      = get_points (T.ABC : medial ())
  C.IH      = circle:new (z.I,z.H)
  C.NAp     = circle:new (z.N,z.Ap)
  C.OA      = circle:new (z.O,z.A)
  z.U        = C.OA.south
  z.L        = C.NAp.south
  z.M        = C.NAp.north
  z.X        = T.ABC.ab: projection (z.C)
  L.CU      = line: new (z.C,z.U)
  L.ML      = line: new (z.M,z.L)
  z.P        = L.CU: projection (z.A)
  z.Q        = L.CU: projection (z.B)
  L.LH      = line: new (z.L,z.H)
  z.F        = intersection (L.LH,C.IH) -- feuerbach
\end{tkzelements}

```



```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLine(L,F)
  \tkzDrawCircle[red] (N,A')
  \tkzDrawCircle[blue] (I,H)
  \tkzDrawCircles[teal] (O,A L,C')
  \tkzDrawSegments(M,L B,U Q,C C,X A,P B,Q)
  \tkzDrawPolygons(A,B,C A',B',C')
  \tkzDrawPoints(A,B,C,N,H,A',B',C',U,L,M,P,Q,F,I)
  \tkzLabelPoints(A,B,C,N,H,A',B',C',U,L,M,P,Q,F,I)
\end{tikzpicture}

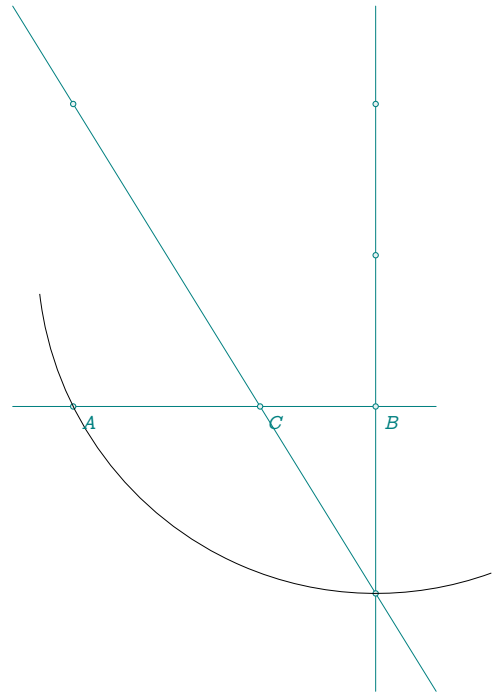
```

23.21 Ratio d'or avec le segment

```

\begin{tkzelements}
  z.A      = point: new (0 , 0)
  z.B      = point: new (8 , 0)
  L.AB     = line: new (z.A,z.B)
  _,_,z.X,z.Y = get_points(L.AB: square ())
  L.BX     = line: new (z.B,z.X)
  z.M      = L.BX.mid
  C.MA     = circle: new (z.M,z.A)
  _,z.K    = intersection (L.BX,C.MA)
  L.AK     = line: new (z.Y,z.K)
  z.C      = intersection (L.AK,L.AB)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(A,B X,K)
  \tkzDrawLine[teal](Y,K)
  \tkzDrawPoints(A,B,C,X,Y,M,K)
  \tkzDrawArc[delta=20](M,A)(K)
  \tkzLabelPoints(A,B,C)
\end{tikzpicture}

```

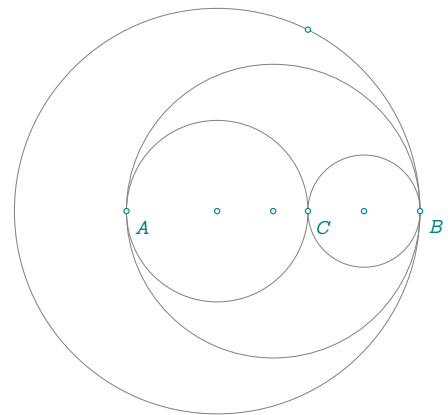


23.22 Gold Arbelos

```

\begin{tkzelements}
  scale    = .6
  z.A      = point: new (0 , 0)
  z.C      = point: new (6 , 0)
  L.AC     = line: new (z.A,z.C)
  _,_,z.x,z.y = get_points(L.AC: square ())
  z.O_1    = L.AC . mid
  C        = circle: new (z.O_1,z.x)
  z.B      = intersection (L.AC,C)
  L.CB     = line: new (z.C,z.B)
  z.O_2    = L.CB.mid
  L.AB     = line: new (z.A,z.B)
  z.O_0    = L.AB.mid
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles(O_1,C O_2,B O_0,B)
  \tkzDrawPoints(A,C,B,O_1,O_2,O_0)
  \tkzLabelPoints(A,C,B)
\end{tikzpicture}

```

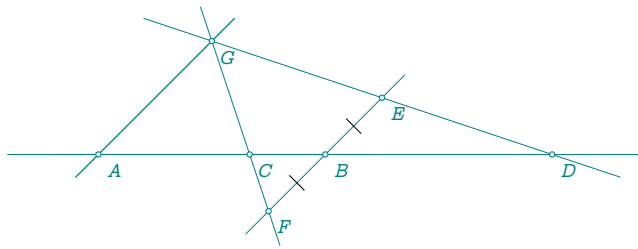


23.23 Division harmonique v1

```

\begin{tkzelements}
  scale=.75
  z.A = point: new (0 , 0)
  z.B = point: new (4 , 0)
  z.G = point: new (2,2)
  L.AG = line : new (z.A,z.G)
  L.AB = line : new (z.A,z.B)
  z.E = L.AG : colinear_at (z.B,.5)
  L.GE = line : new (z.G,z.E)
  z.D = intersection (L.GE,L.AB)
  z.F = z.B : symmetry (z.E)
  L.GF = line :new (z.G,z.F)
  z.C = intersection (L.GF,L.AB)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(A,B A,G A,D A,G F,E G,F G,D)
  \tkzDrawPoints(A,B,G,E,F,C,D)
  \tkzLabelPoints(A,B,G,E,F,C,D)
  \tkzMarkSegments(F,B B,E)
\end{tikzpicture}

```

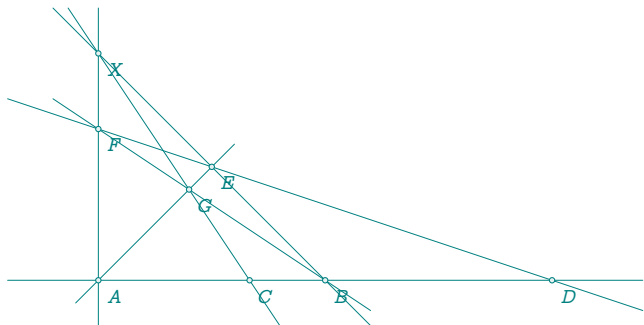


23.24 Division harmonique v2

```

\begin{tkzelements}
  scale = .5
  z.A = point: new (0 , 0)
  z.B = point: new (6 , 0)
  z.D = point: new (12 , 0)
  L.AB = line: new (z.A,z.B)
  z.X = L.AB.north_pa
  L.XB = line: new (z.X,z.B)
  z.E = L.XB.mid
  L.ED = line: new (z.E,z.D)
  L.AX = line: new (z.A,z.X)
  L.AE = line: new (z.A,z.E)
  z.F = intersection (L.ED,L.AX)
  L.BF = line: new (z.B,z.F)
  z.G = intersection (L.AE,L.BF)
  L.GX = line: new (z.G,z.X)
  z.C = intersection (L.GX,L.AB)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(A,D A,E B,F D,F X,A X,B X,C)
  \tkzDrawPoints(A,...,G,X)
  \tkzLabelPoints(A,...,G,X)
\end{tikzpicture}

```

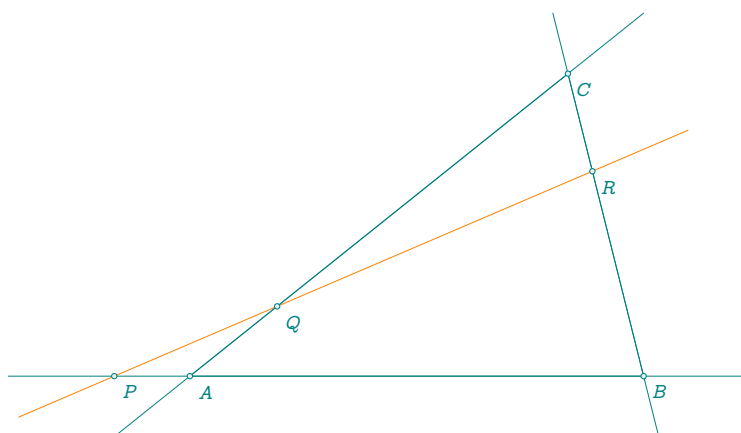


23.25 Menelaus

```

\begin{tkzelements}
  z.A = point: new (0 , 0)
  z.B = point: new (6 , 0)
  z.C = point: new (5 , 4)
  z.P = point: new (-1 , 0)
  z.X = point: new (6 , 3)
  L.AC = line: new (z.A,z.C)
  L.PX = line: new (z.P,z.X)
  L.BC = line: new (z.B,z.C)
  z.Q = intersection (L.AC,L.PX)
  z.R = intersection (L.BC,L.PX)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawLine[new](P,R)
  \tkzDrawLines(P,B A,C B,C)
  \tkzDrawPoints(P,Q,R,A,B,C)
  \tkzLabelPoints(A,B,C,P,Q,R)
\end{tikzpicture}

```

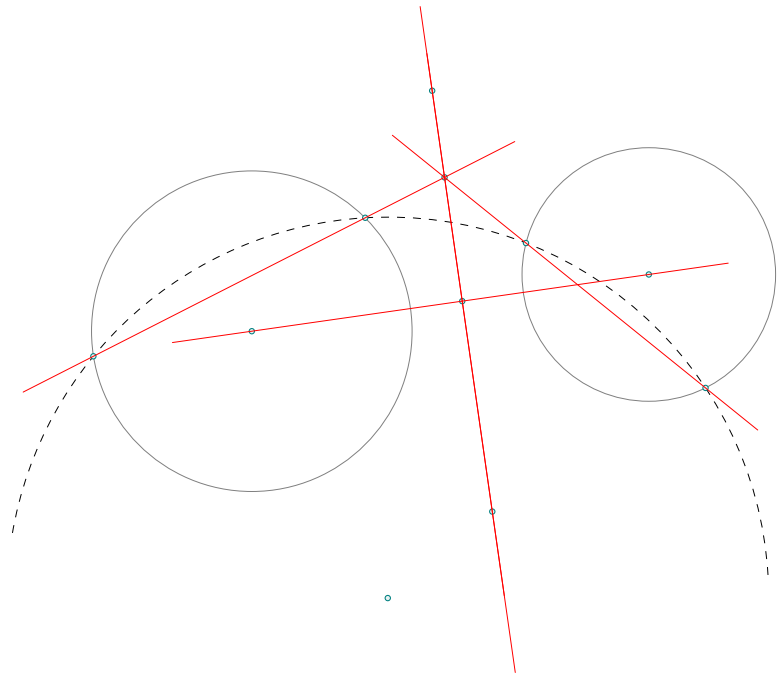


23.26 Axe radical v1

```

\begin{tkzelements}
  scale = .75
  z.X = point : new (0,0)
  z.B = point : new (2,2)
  z.Y = point : new (7,1)
  z.Ap = point : new (8,-1)
  L.XY = line : new (z.X,z.Y)
  C.XB = circle : new (z.X,z.B)
  C.YAp = circle : new (z.Y,z.Ap)
  z.E,z.F = get_points (C.XB : radical_axis (C.YAp))
  z.A = C.XB : point (0.4)
  T.ABAp = triangle: new (z.A,z.B,z.Ap)
  z.O = T.ABAp.circumcenter
  C.OAp = circle : new (z.O,z.Ap)
  _,z.Bp = intersection (C.OAp,C.YAp)
  L.AB = line : new (z.A,z.B)
  L.ApBp = line : new (z.Ap,z.Bp)
  z.M = intersection (L.AB,L.ApBp)
  z.H = L.XY : projection (z.M)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles(X,B Y,A')
  \tkzDrawArc[dashed,delta=30](O,A')(A)
  \tkzDrawPoints(A,B,A',B',M,H,X,Y,O,E,F)
  \tkzDrawLines[red](A,M A',M X,Y E,F)
  \tkzDrawLines[red,add=1 and 3](M,H)
\end{tikzpicture}

```

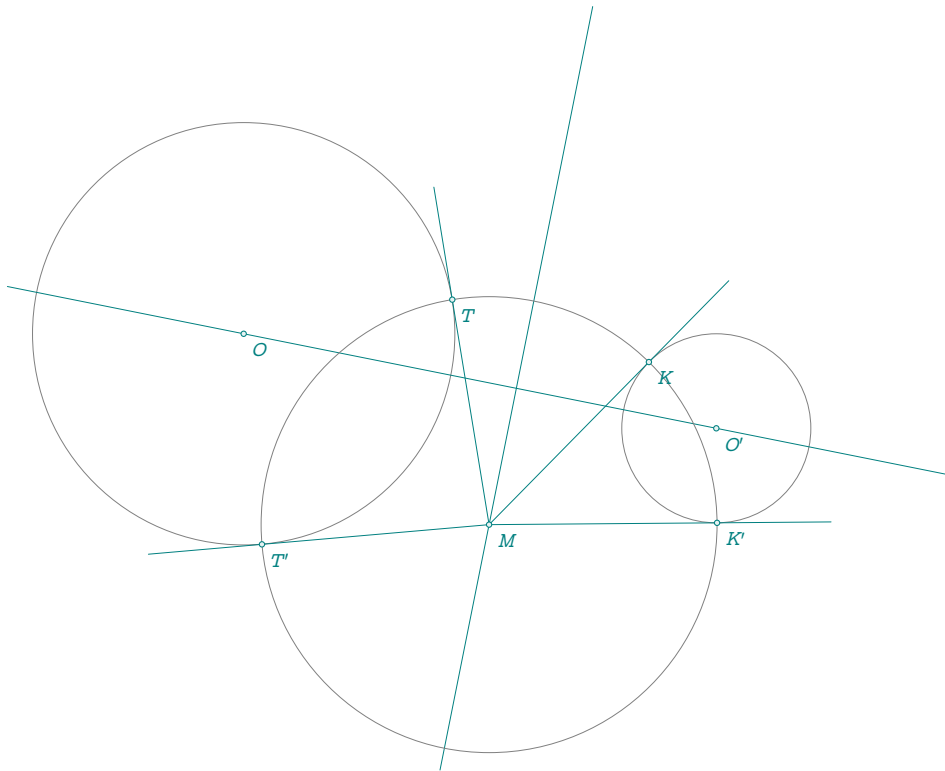


23.27 Axe radical v2

```

\begin{tkzelements}
scale      = 1.25
z.O        = point : new (-1,0)
z.Op       = point : new (4,-1)
z.B        = point : new (0,2)
z.D        = point : new (4,0)
C.OB       = circle :   new (z.O,z.B)
C.OpD      = circle :   new (z.Op,z.D)
L.EF       = C.OB : radical_axis (C.OpD)
z.E,z.F    = get_points (L.EF)
z.M        = L.EF : point (.75)
L.MT,L.MTp = C.OB : tangent_from (z.M)
_,z.T      = get_points (L.MT)
_,z.Tp     = get_points (L.MTp)
L.MK,L.MKp = C.OpD : tangent_from (z.M)
_,z.K      = get_points (L.MK)
_,z.Kp     = get_points (L.MKp)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles(O,B O',D)
  \tkzDrawLine(E,F)
  \tkzDrawLine[add=.5 and .5](O,O')
  \tkzDrawLines[add = 0 and .5](M,T M,T' M,K M,K')
  \tkzDrawCircle(M,T)
  \tkzDrawPoints(O,O',T,M,T',K,K')
  \tkzLabelPoints(O,O',T,T',K,K',M)
\end{tikzpicture}

```

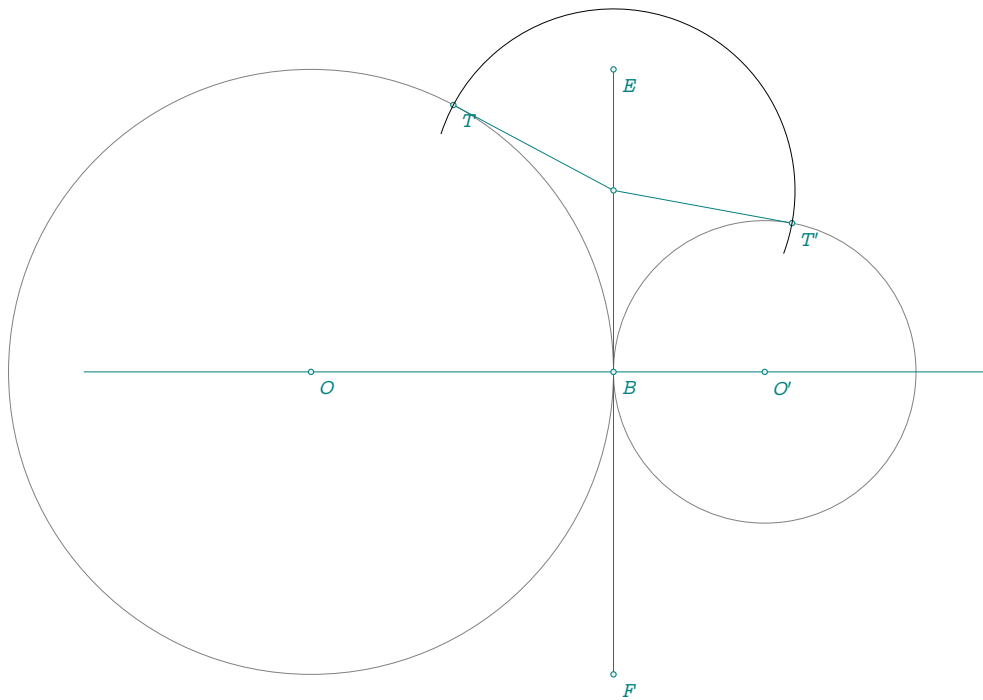


23.28 Axe radical v3

```

\begin{tkzelements}
z.O      = point : new (0,0)
z.B      = point : new (4,0)
z.Op     = point : new (6,0)
C.OB     = circle :   new (z.O,z.B)
C.OpB    = circle :   new (z.Op,z.B)
L.EF     = C.OB : radical_axis (C.OpB)
z.E,z.F  = get_points(L.EF)
z.M      = L.EF : point (0.2)
L        = C.OB : tangent_from (z.M)
_,z.T    = get_points (L)
L        = C.OpB : tangent_from (z.M)
_,z.Tp   = get_points (L)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(O,B O',B)
\tkzDrawSegments(M,T M,T')
\tkzDrawSegments(E,F)
\tkzDrawLine[add=.5 and .5](O,O')
\tkzDrawPoints(O,B,O',E,F,M,T,T')
\tkzLabelPoints(O,O',B,E,F,T,T')
\tkzDrawArc(M,T')(T)
\end{tikzpicture}

```

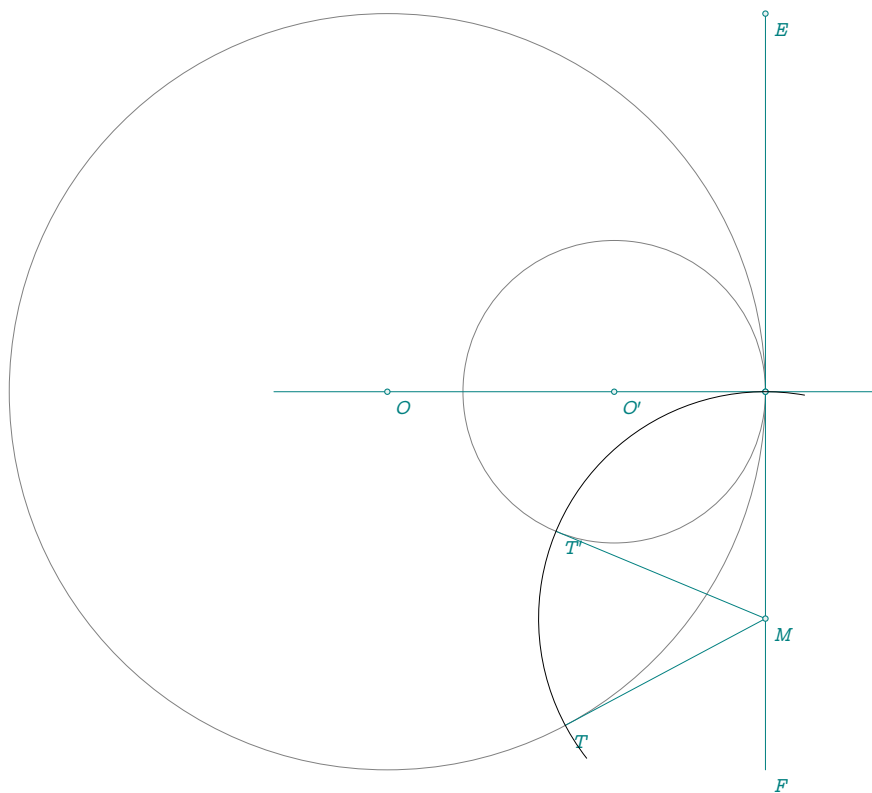


23.29 Axe radical v4

```

\begin{tkzelements}
z.O      = point : new (0,0)
z.B      = point : new (5,0)
z.Op     = point : new (3,0)
C.OB     = circle : new (z.O,z.B)
C.OpB    = circle : new (z.Op,z.B)
L.EF     = C.OB : radical_axis (C.OpB)
z.E,z.F  = get_points(L.EF)
z.H      = L.EF.mid
z.M      = L.EF : point (.8)
_,L      = C.OB : tangent_from (z.M)
_,z.T    = get_points (L)
_,L      = C.OpB : tangent_from (z.M)
_,z.Tp   = get_points (L)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(O,B O',B)
\tkzDrawSegments(M,T M,T')
\tkzDrawSegments(E,F)
\tkzDrawLine[add=.3 and .3](O,H)
\tkzDrawPoints(O,O',B,E,H,M)
\tkzLabelPoints[below right](O,O',E,F,M,T,T')
\tkzDrawArc(M,B)(T)
\end{tikzpicture}

```

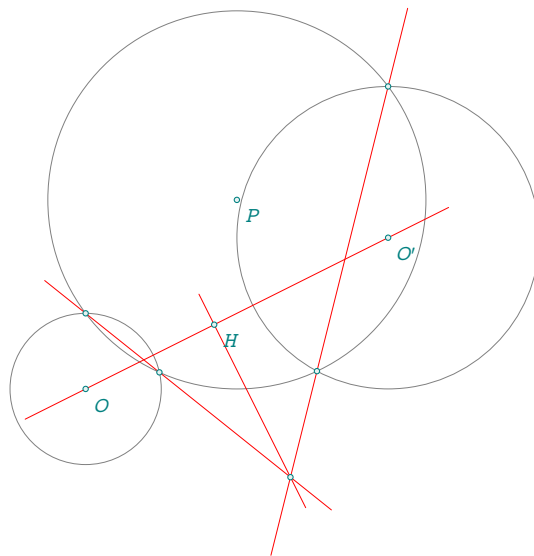


23.30 Centre radical

```

\begin{tkzelements}
  z.O      = point : new (0,0)
  z.x      = point : new (1,0)
  z.y      = point : new (4,0)
  z.z      = point : new (2,0)
  z.Op     = point : new (4,2)
  z.P      = point : new (2,2.5)
  C.Ox     = circle : new (z.O,z.x)
  C.Pz     = circle : new (z.P,z.z)
  C.Opy    = circle : new (z.Op,z.y)
  z.ap,z.a = intersection (C.Ox,C.Pz)
  z.bp,z.b = intersection (C.Opy,C.Pz)
  L.aap    = line : new (z.a,z.ap)
  L.bbp    = line : new (z.b,z.bp)
  z.X      = intersection (L.aap,L.bbp)
  -- or z.X = radical_center(C.Ox,C.Pz,C.Opy)
  L.OOp    = line : new (z.O,z.Op)
  z.H      = L.OOp : projection (z.X)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles(O,a O',b P,z)
  \tkzDrawLines[red](a,X b',X H,X O,O')
  \tkzDrawPoints(O,O',P,a,a',b,b',X,H)
  \tkzLabelPoints[below right](O,O',P,H)
\end{tikzpicture}

```

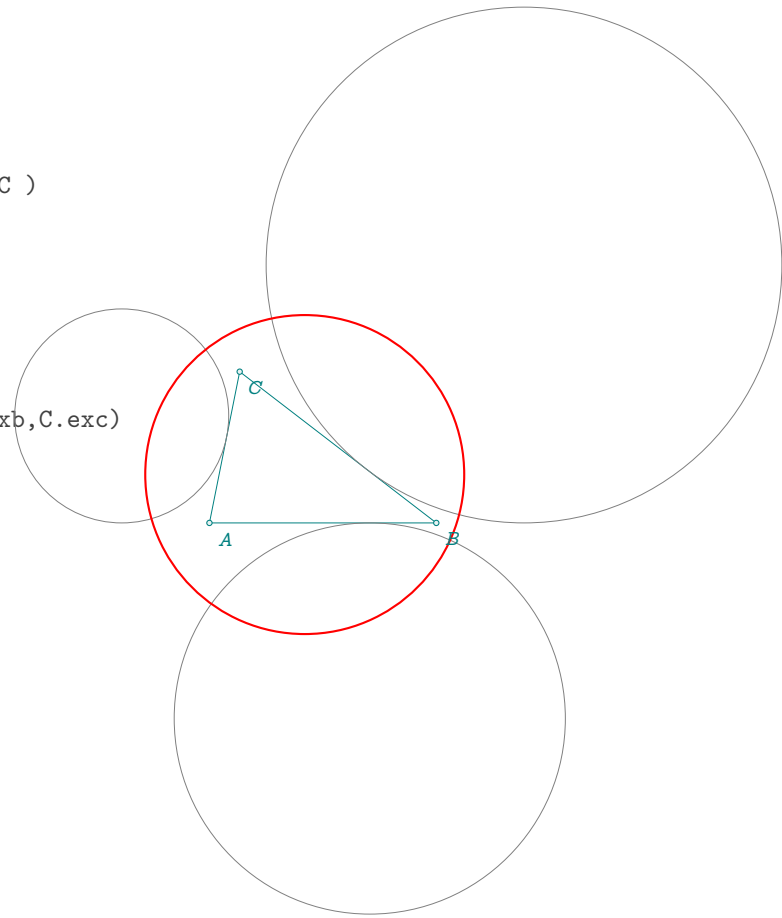


23.31 Cercle radical

```

\begin{tkzelements}
  scale      = .25
  z.A        = point: new (0,0)
  z.B        = point: new (6,0)
  z.C        = point: new (0.8,4)
  T.ABC      = triangle : new ( z.A,z.B,z.C )
  C.exa      = T.ABC : ex_circle ( )
  z.I_a,z.Xa = get_points (C.exa)
  C.exb      = T.ABC : ex_circle (1)
  z.I_b,z.Xb = get_points (C.exb)
  C.exc      = T.ABC : ex_circle (2)
  z.I_c,z.Xc = get_points (C.exc)
  C.ortho    = C.exa : radical_circle (C.exb,C.exc)
  z.w,z.a    = get_points (C.ortho)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawCircles(I_a,Xa I_b,Xb I_c,Xc)
  \tkzDrawCircles[red,thick](w,a)
  \tkzDrawPoints(A,B,C)
  \tkzLabelPoints(A,B,C)
\end{tikzpicture}

```



23.32 Ellipse d'Euler

```

\begin{tkzelements}
  scale          = 1.3
  z.A            = point: new (0 , 0)
  z.B            = point: new (5 , 1)
  L.AB           = line : new (z.A,z.B)
  z.C            = point: new (.8 , 3)
  T.ABC          = triangle: new (z.A,z.B,z.C)
  z.N            = T.ABC.eulercenter
  z.G            = T.ABC.centroid
  z.O            = T.ABC.circumcenter
  z.H            = T.ABC.orthocenter
  z.Ma,z.Mb,
  z.Mc           = get_points (T.ABC : medial ())
  z.Ha,z.Hb,
  z.Hc           = get_points (T.ABC : orthic ())
  z.Ea,z.Eb,
  z.Ec           = get_points (T.ABC: extouch())
  L.euler        = T.ABC : euler_line ()
  C.circum       = T.ABC : circum_circle ()
  C.euler        = T.ABC : euler_circle ()
  z.I,z.J        = intersection (L.euler,C.euler)
  E              = ellipse: foci (z.H,z.O,z.I)
  a              = E.Rx
  b              = E.Ry
  ang            = math.deg(E.slope)
  L.AH           = line: new (z.A,z.H)
  L.BH           = line: new (z.B,z.H)
  L.CH           = line: new (z.C,z.H)
  z.X            = intersection (L.AH,C.circum)
  _,z.Y          = intersection (L.BH,C.circum)
  _,z.Z          = intersection (L.CH,C.circum)
  L.BC           = line: new (z.B,z.C)
  L.XO           = line: new (z.X,z.O)
  L.YO           = line: new (z.Y,z.O)
  L.ZO           = line: new (z.Z,z.O)
  z.x            = intersection (L.BC,L.XO)
  z.U            = intersection (L.XO,E)
  _,z.V          = intersection (L.YO,E)
  _,z.W          = intersection (L.ZO,E)
\end{tkzelements}

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawCircles[red](N,Ma O,A)
  \tkzDrawSegments(A,X B,Y C,Z B,Hb C,Hc X,O Y,O Z,O)
  \tkzDrawPolygon[red](U,V,W)
  \tkzLabelPoints[red](U,V,W)
  \tkzLabelPoints(A,B,C,X,Y,Z)
  \tkzDrawLine[blue](I,J)
  \tkzLabelPoints[blue,right](O,N,G,H,I,J)
  \tkzDrawPoints(I,J,U,V,W)

```

```

\tkzDrawPoints(A,B,C,N,G,H,O,X,Y,Z,Ma,Mb,Mc,Ha,Hb,Hc)
\tkzDrawEllipse[blue](N,\tkzUseLua{a},\tkzUseLua{b},\tkzUseLua{ang})
\end{tikzpicture}

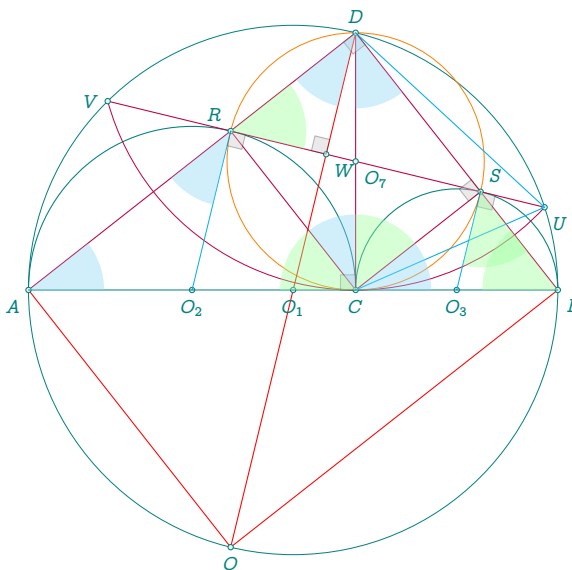
```

23.33 Gold Arbelos propriétés

```

\begin{tkzelements}
z.A      = point : new(0,0)
z.B      = point : new(10,0)
z.C      = gold_segment_ (z.A,z.B)
L.AB     = line:new (z.A,z.B)
z.O_1    = L.AB.mid
L.AC     = line:new (z.A,z.C)
z.O_2    = L.AC.mid
L.CB     = line:new (z.C,z.B)
z.O_3    = L.CB.mid
C1       = circle:new (z.O_1,z.B)
C2       = circle:new (z.O_2,z.C)
C3       = circle:new (z.O_3,z.B)
z.Q      = C2.north
z.P      = C3.north
L1       = line:new (z.O_2,z.O_3)
z.M_0    = L1:harmonic_ext (z.C)
L2       = line:new (z.O_1,z.O_2)
z.M_1    = L2:harmonic_int (z.A)
L3       = line:new (z.O_1,z.O_3)
z.M_2    = L3:harmonic_int (z.B)
Lbq      = line:new (z.B,z.Q)
Lap      = line:new (z.A,z.P)
z.S      = intersection (Lbq,Lap)
z.x      = z.C: north ()
L        = line : new (z.C,z.x)
z.D,_    = intersection (L,C1)
L.CD     = line :new (z.C,z.D)
z.O_7    = L.CD.mid
C.DC     = circle: new (z.D,z.C)
z.U,z.V  = intersection (C.DC,C1)
L.UV     = line :new (z.U,z.V)
z.R ,z.S = L.UV : projection (z.O_2,z.O_3)
L.O1D    = line : new (z.O_1,z.D)
z.W      = intersection (L.UV,L.O1D)
z.O      = C.DC : inversion (z.W)
\end{tkzelements}

```



```

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles[teal](O_1,B)
\tkzDrawSemiCircles[thin,teal](O_2,C O_3,B)
\tkzDrawArc[purple,delta=0](D,V)(U)
\tkzDrawCircle[new](O_7,C)
\tkzDrawSegments[thin,purple](A,D D,B C,R C,S C,D U,V)
\tkzDrawSegments[thin,red](O,D A,O O,B)
\tkzDrawPoints(A,B,C,D,O_7) %,
\tkzDrawPoints(O_1,O_2,O_3,U,V,R,S,W,O)

```

```

\tkzDrawSegments[cyan](O_3,S O_2,R)
\tkzDrawSegments[very thin](A,B)
\tkzDrawSegments[cyan,thin](C,U U,D)
\tkzMarkRightAngles[size=.2,fill=gray!40,opacity=.4](D,C,A A,D,B
  D,S,C D,W,V O_3,S,U O_2,R,U)
\tkzFillAngles[cyan!40,opacity=.4](B,A,D A,D,O_1
  C,D,B D,C,R B,C,S A,R,O_2)
\tkzFillAngles[green!40,opacity=.4](S,C,D W,R,D
  D,B,C R,C,A O_3,S,B)
\tkzLabelPoints[below](C,O_2,O_3,O_1)
\tkzLabelPoints[above](D)
\tkzLabelPoints[below](O)
\tkzLabelPoints[below left](A)
\tkzLabelPoints[above left](R)
\tkzLabelPoints[above right](S)
\tkzLabelPoints[left](V)
\tkzLabelPoints[below right](B,U,W,O_7)
\end{tikzpicture}

```

23.34 Cercle d'Apollonius v1 avec inversion

```

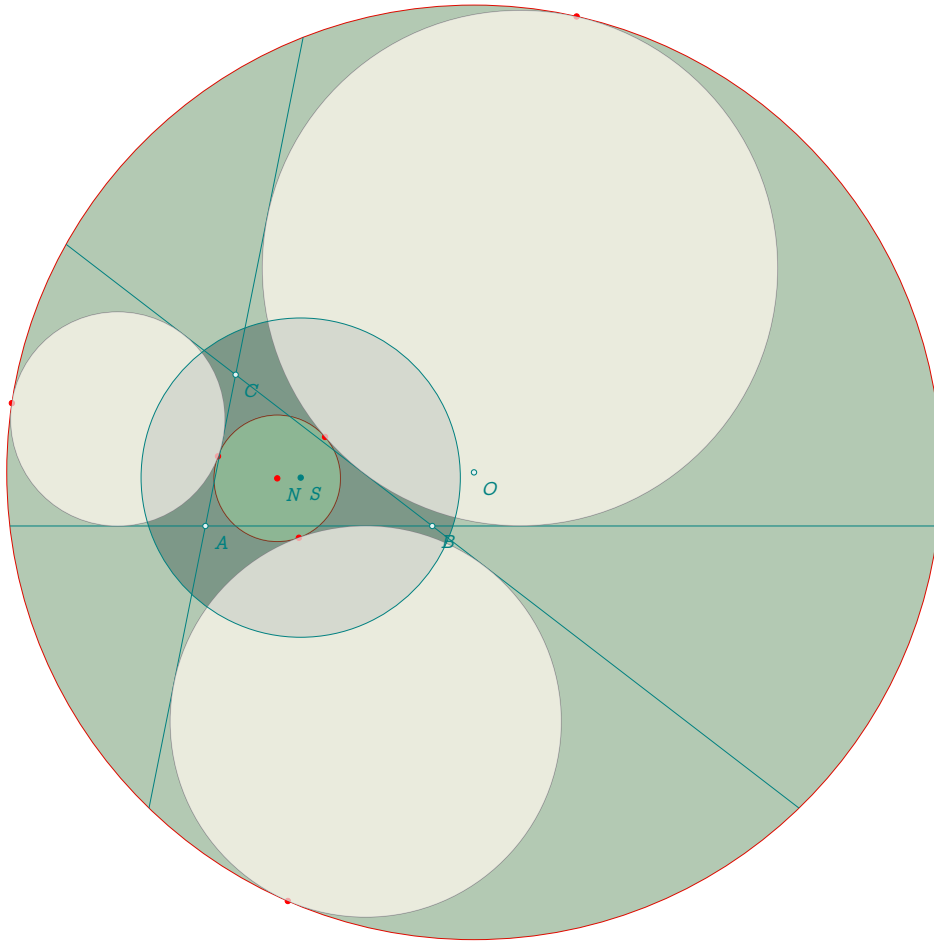
\begin{tkzelements}
  scale          = .7
  z.A            = point: new (0,0)
  z.B            = point: new (6,0)
  z.C            = point: new (0.8,4)
  T.ABC          = triangle : new ( z.A,z.B,z.C )
  z.N            = T.ABC.eulercenter
  z.Ea,z.Eb,z.Ec = get_points ( T.ABC : feuerbach () )
  z.Ja,z.Jb,z.Jc = get_points ( T.ABC : excentral () )
  z.S            = T.ABC : spieker_center ()
  C.JaEa         = circle : new (z.Ja,z.Ea)
  C.ortho        = circle : radius (z.S,math.sqrt(C.JaEa : power (z.S) ))
  z.a            = C.ortho.south
  C.euler        = T.ABC: euler_circle ()
  C.apo          = C.ortho : inversion (C.euler)
  z.O            = C.apo.center
  z.xa,z.xb,z.xc = C.ortho : inversion (z.Ea,z.Eb,z.Ec)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles[red](O,xa N,Ea)
  \tkzFillCircles[green!30!black,opacity=.3](O,xa)
  \tkzFillCircles[yellow!30,opacity=.7](Ja,Ea Jb,Eb Jc,Ec)
  \tkzFillCircles[teal!30!black,opacity=.3](S,a)
  \tkzFillCircles[green!30,opacity=.3](N,Ea)
  \tkzDrawPoints[red](Ea,Eb,Ec,xa,xb,xc,N)
  \tkzClipCircle(O,xa)
  \tkzDrawLines[add=3 and 3](A,B A,C B,C)
  \tkzDrawCircles(Ja,Ea Jb,Eb Jc,Ec)
  \tkzFillCircles[lightgray!30,opacity=.7](Ja,Ea Jb,Eb Jc,Ec)
  \tkzDrawCircles[teal](S,a)
  \tkzDrawPoints(A,B,C,O)

```

```

\tkzDrawPoints[teal](S)
\tkzLabelPoints(A,B,C,O,S,N)
\end{tikzpicture}

```



23.35 Cercle d'Apollonius v2

```

\begin{tkzelements}
  scale      = .5
  z.A        = point: new (0,0)
  z.B        = point: new (6,0)
  z.C        = point: new (0.8,4)
  T.ABC      = triangle: new(z.A,z.B,z.C)
  z.O        = T.ABC.circumcenter
  z.H        = T.ABC.orthocenter
  z.G        = T.ABC.centroid
  z.L        = T.ABC: lemoine_point ()
  z.S        = T.ABC: spieker_center ()
  C.euler    = T.ABC: euler_circle ()
  z.N,z.Ma   = get_points (C.euler)
  C.exA      = T.ABC : ex_circle ()
  z.Ja,z.Xa  = get_points (C.exA)
  C.exB      = T.ABC : ex_circle (1)
  z.Jb,z.Xb  = get_points (C.exB)
  C.exC      = T.ABC : ex_circle (2)
  z.Jc,z.Xc  = get_points (C.exC)

```

```

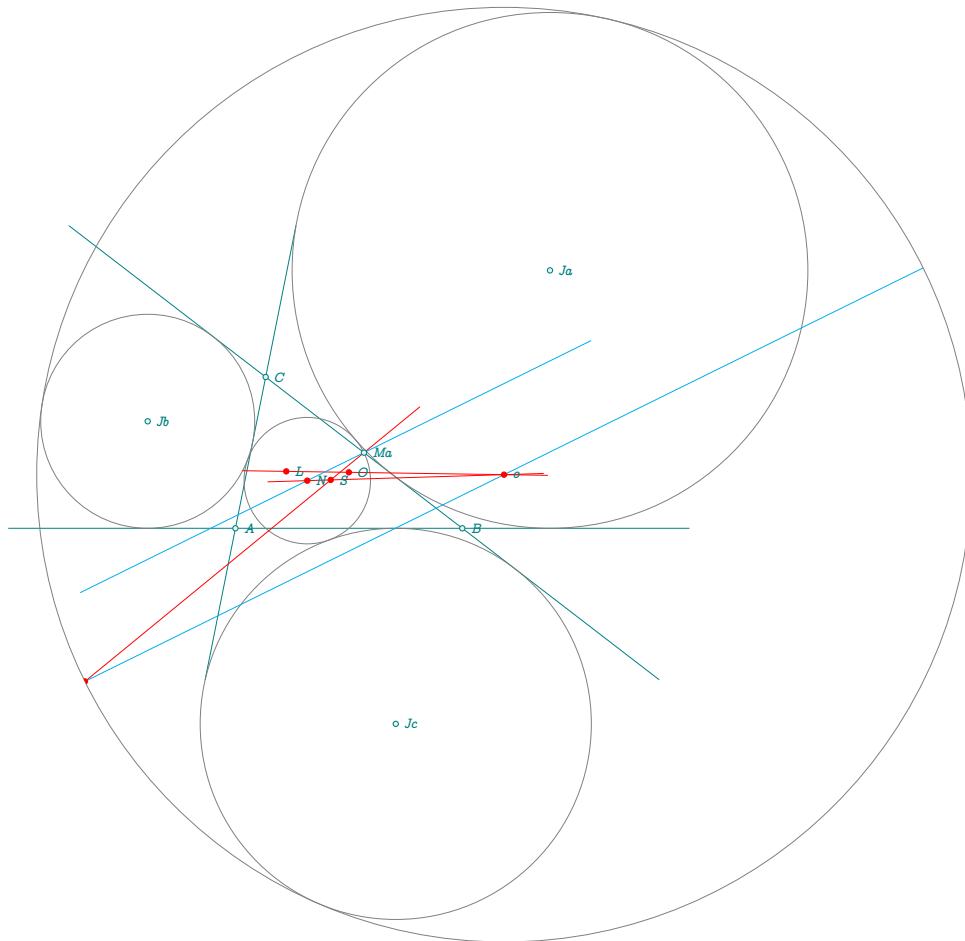
L.OL      = line: new (z.O,z.L)
L.NS      = line: new (z.N,z.S)
z.o       = intersection (L.OL,L.NS) -- center of Apollonius circle
L.NMa     = line: new (z.N,z.Ma)
L.ox      = L.NMa: ll_from (z.o)
L.MaS     = line: new (z.Ma,z.S)
z.t       = intersection (L.ox,L.MaS) -- through
\end{tkzelements}

```

```

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawLines[add=1 and 1](A,B A,C B,C)
\tkzDrawCircles(Ja,Xa Jb,Xb Jc,Xc o,t N,Ma) %
\tkzClipCircle(o,t)
\tkzDrawLines[red](o,L N,o Ma,t)
\tkzDrawLines[cyan,add=4 and 4](Ma,N o,t)
\tkzDrawPoints(A,B,C,Ma,Ja,Jb,Jc)
\tkzDrawPoints[red](N,O,L,S,o,t)
\tkzLabelPoints[right,font=\tiny](A,B,C,Ja,Jb,Jc,O,N,L,S,Ma,o)
\end{tikzpicture}

```



23.36 Cercles orthogonaux v1

```

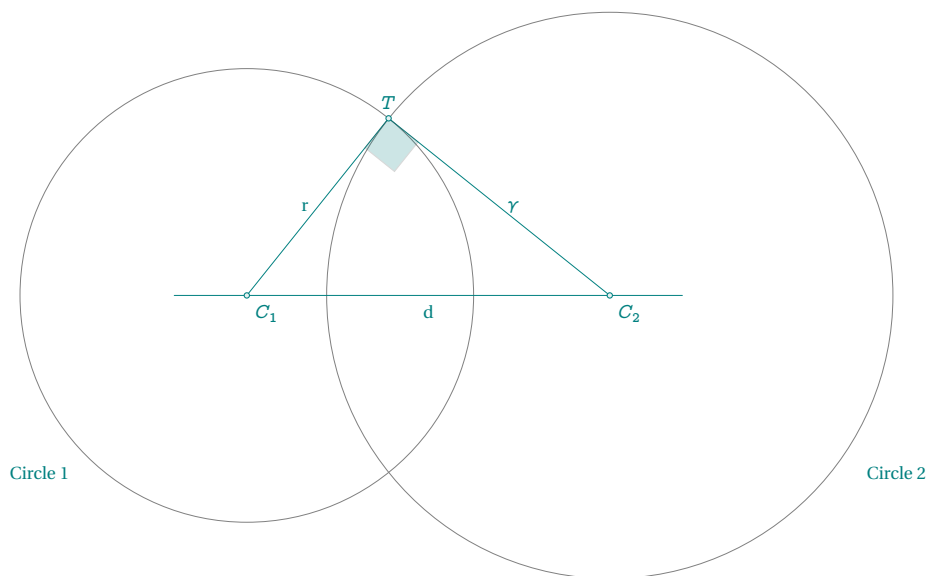
\begin{tkzelements}
scale    = .6

```

```

z.C_1 = point: new (0,0)
z.C_2 = point: new (8,0)
z.A   = point: new (5,0)
C     = circle: new (z.C_1,z.A)
z.S,z.T = get_points (C: orthogonal_from (z.C_2))
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(C_1,T C_2,T)
\tkzDrawSegments(C_1,T C_2,T)
\tkzDrawLine(C_1,C_2)
\tkzMarkRightAngle[fill=teal,%
opacity=.2,size=1](C_1,T,C_2)
\tkzDrawPoints(C_1,C_2,T)
\tkzLabelPoints(C_1,C_2)
\tkzLabelPoints[above](T)
\tkzLabelSegment[left](C_1,T){r}
\tkzLabelSegments[right](C_2,T){$\gamma$}
\tkzLabelSegment[below](C_1,C_2){d}
\tkzLabelCircle[left=10pt](C_1,T)(180){Circle 1}
\tkzLabelCircle[right=10pt](C_2,T)(180){Circle 2}
\end{tikzpicture}

```



23.37 Cercles orthogonaux v2

```

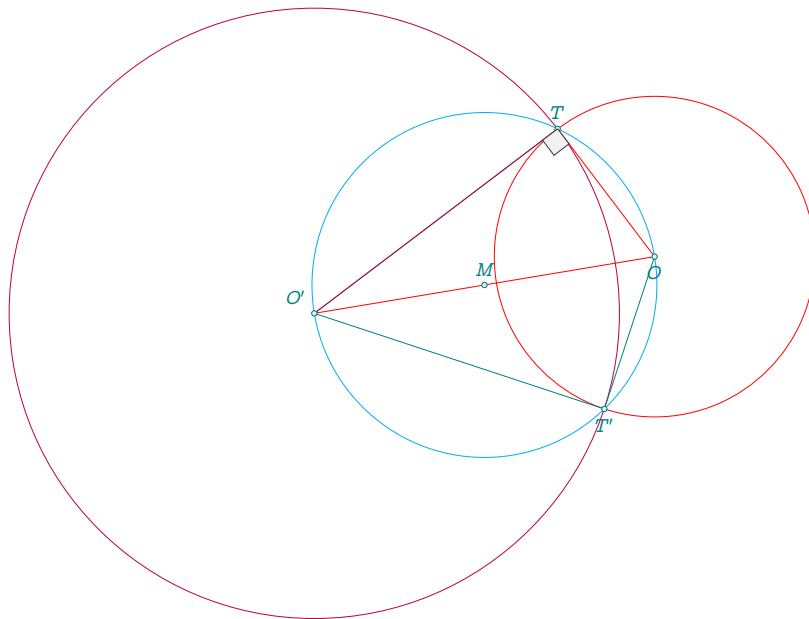
\begin{tkzelements}
scale = .75
z.O = point: new (2,2)
z.Op = point: new (-4,1)
z.P = point: polar (4,0)
C.OP = circle: new (z.O,z.P)
C.Oz1 = C.OP : orthogonal_from (z.Op)
z.z1 = C.Oz1.through
L.OP = line : new (z.O,z.P)
C.Opz1 = circle: new (z.Op,z.z1)
L.T,L.Tp = C.Opz1 : tangent_from (z.O)

```

```

z.T      = L.T.pb
z.Tp     = L.Tp.pb
L.OOp    = line : new (z.O,z.Op)
z.M      = L.OOp.mid
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircle[red](O,P)
  \tkzDrawCircle[purple](O',z1)
  \tkzDrawCircle[cyan](M,T)
  \tkzDrawSegments(O',T O,T' O',T')
  \tkzDrawSegment[purple](O',T)
  \tkzDrawSegments[red](O,T O,O')
  \tkzDrawPoints(O,O',T,T',M)
  \tkzMarkRightAngle[fill=gray!10](O',T,O)
  \tkzLabelPoint[below](O){$O$}
  \tkzLabelPoint[above](T){$T$}
  \tkzLabelPoint[above](M){$M$}
  \tkzLabelPoint[below](T'){$T'$}
  \tkzLabelPoint[above left](O'){$O'$}
\end{tikzpicture}

```



23.38 Cercle orthogonal à deux cercles

```

\begin{tkzelements}
  z.O      = point : new (-1,0)
  z.B      = point : new (0,2)
  z.Op     = point : new (4,-1)
  z.D      = point : new (4,0)
  C.OB     = circle : new (z.O,z.B)
  C.OpD    = circle : new (z.Op,z.D)
  z.E,z.F  = get_points (C.OB : radical_axis (C.OpD))
  L.EF     = line : new (z.E,z.F)
  z.M      = L.EF : point (.25)
\end{tkzelements}

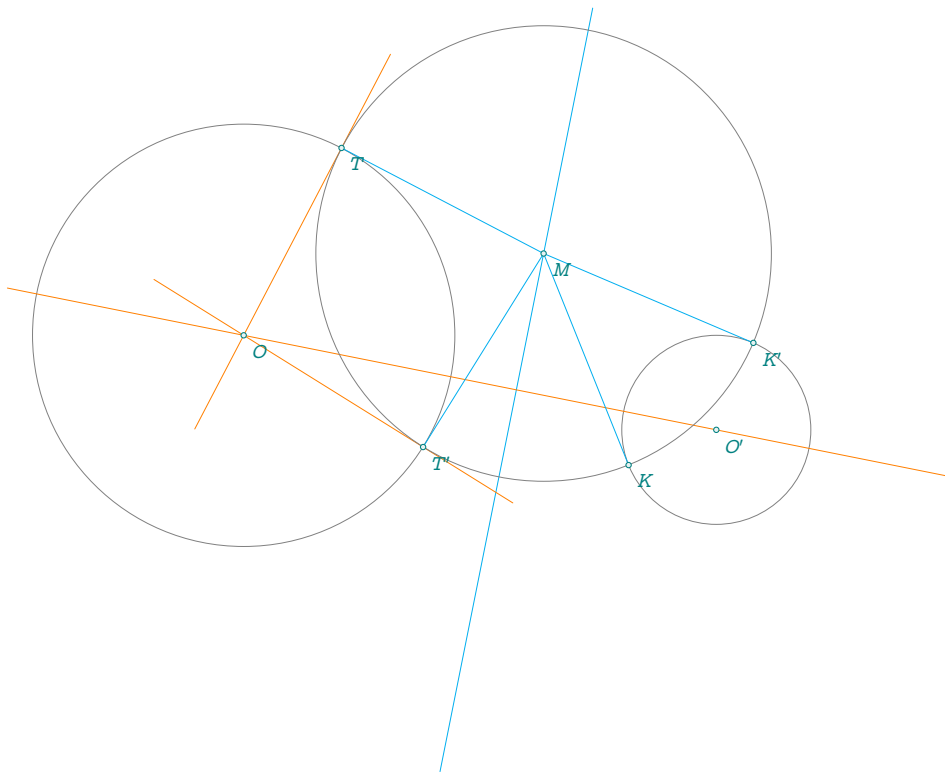
```



```

L.T,L.Tp    = C.OB : tangent_from (z.M)
L.K,L.Kp    = C.OpD : tangent_from (z.M)
z.T         = L.T.pb
z.K         = L.K.pb
z.Tp        = L.Tp.pb
z.Kp        = L.Kp.pb
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(O,B O',D)
\tkzDrawLine[cyan](E,F)
\tkzDrawLines[add=.5 and .5,orange](O,O' O,T O,T')
\tkzDrawSegments[cyan](M,T M,T' M,K M,K')
\tkzDrawCircle(M,T)
\tkzDrawPoints(O,O',T,M,T',K,K')
\tkzLabelPoints(O,O',T,T',M,K,K')
\end{tikzpicture}

```




```

z.J      = intersection (L.BBN,L.M1P2)
L.AP0    = line : new (z.A,z.P_0)
L.BP0    = line : new (z.B,z.P_0)
C.O4P0   = circle : new (z.O_4,z.P_0)
_,z.G    = intersection (L.AP0,C.O4P0)
z.H      = intersection (L.BP0,C.O4P0)
z.Ap     = z.M_1: symmetry (z.A)
z.H_4,z.F,z.E,z.H_0 = L.AB : projection (z.O_4,z.G,z.H,z.P_0)
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircle[thin,fill=green!10](O_4,P_0)
\tkzDrawCircle[purple,fill=purple!10,opacity=.5](O_5,C)
\tkzDrawSemiCircles[teal](O_0,B)
\tkzDrawSemiCircles[thin,teal,fill=teal!20,opacity=.5](O_1,C O_2,B)
\tkzDrawSemiCircles[color = orange](M_2,B)
\tkzDrawSemiCircles[color = orange](M_1,A')
\tkzDrawArc[purple,delta=0](M_0,P_0)(C)
\tkzDrawSegments[very thin](A,B A,P B,Q)
\tkzDrawSegments[color=cyan](O_0,P_0 B,J G,J G,O_0 H,O_2)
\tkzDrawSegments[ultra thin,purple](M_1,P_0 M_2,P_0 M_1,M_0 M_0,P_1 M_0,P_0 M_1,J)
\tkzDrawPoints(A,B,C,P_0,P_2,P_1,M_0,M_1,M_2,J,P,Q,S)
\tkzDrawPoints(O_0,O_1,O_2,O_4,O_5,G,H)
\tkzMarkRightAngle[size=.2,fill=gray!20,opacity=.4](O_0,P_0,M_0)
\tkzLabelPoints[below](A,B,C,M_0,M_1,M_2,O_1,O_2,O_0)
\tkzLabelPoints[above](P_0,O_5,O_4)
\tkzLabelPoints[above](P_1,J)
\tkzLabelPoints[above](P_2,P,Q,S)
\tkzLabelPoints[above right](H,E)
\tkzLabelPoints[above left](F,G)
\tkzLabelPoints[below right](H_0)
\tkzLabelCircle[below=4pt,font=\scriptsize](O_1,C)(80){\beta}
\tkzLabelCircle[below=4pt,font=\scriptsize](O_2,B)(80){\gamma}
\tkzLabelCircle[below=4pt,font=\scriptsize](O_0,B)(110){\alpha}
\tkzLabelCircle[left,font=\scriptsize](O_4,P_2)(60){\delta}
\tkzLabelCircle[above left,font=\scriptsize](O_5,C)(40){\epsilon}
\end{tikzpicture}

```

23.40 Faisceau: v1

```

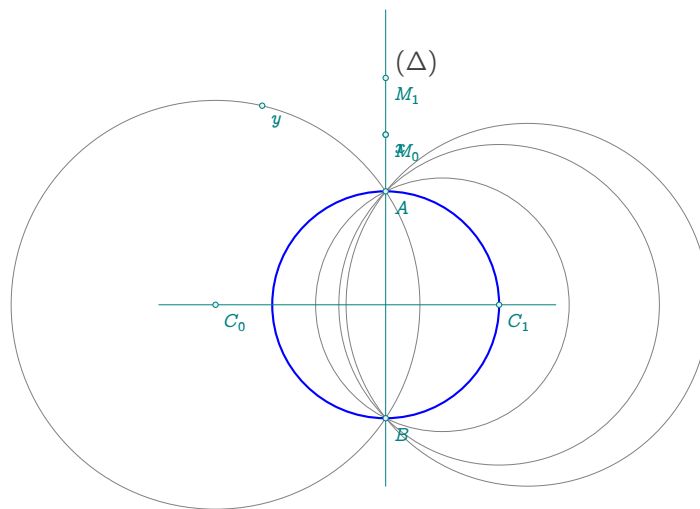
\begin{tkzelements}
scale    = .75
z.A      = point : new (0,2)
z.B      = point : new (0,-2)
z.C_0    = point : new (-3,0)
z.C_1    = point : new (2,0)
z.C_3    = point : new (2.5,0)
z.C_5    = point : new (1,0)
L.BA     = line : new (z.B,z.A)
z.M_0    = L.BA : point (1.25)
z.M_1    = L.BA : point (1.5)
C.C0A    = circle : new (z.C_0,z.A)
z.x,z.y  = get_points (C.C0A : orthogonal_from (z.M_0))
\end{tkzelements}

```

```

z.xp,z.yp = get_points (C.C0A : orthogonal_from (z.M_1))
z.O       = L.BA.mid
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(C_0,A C_1,A C_3,A C_5,A)
\tkzDrawCircles[thick,color=red](M_0,x M_1,x')
\tkzDrawCircles[thick,color=blue](O,A)
\tkzDrawLines(C_0,C_1 B,M_1)
\tkzDrawPoints(A,B,C_0,C_1,M_0,M_1,x,y)
\tkzLabelPoints[below right](A,B,C_0,C_1,M_0,M_1,x,y)
\tkzLabelLine[pos=1.25,right]( M_0,M_1){$(\Delta)$}
\end{tikzpicture}

```



23.41 Faisceau v2

```

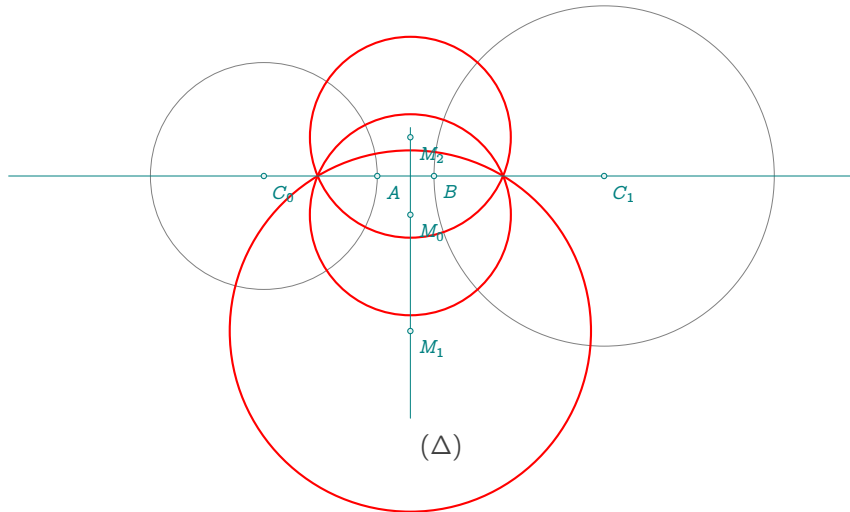
\begin{tkzelements}
scale=.75
z.A = point : new (0,0)
z.B = point : new (1,0)
z.C_0 = point : new (-2,0)
z.C_1 = point : new (4,0)
C.C0A = circle : new (z.C_0,z.A)
C.C1B = circle : new (z.C_1,z.B)
L.EF = C.C0A : radical_axis (C.C1B)
z.M_0 = L.EF : point (.4)
z.M_1 = L.EF : point (.1)
z.M_2 = L.EF : point (.6)
C.orth0 = C.C0A : orthogonal_from (z.M_0)
C.orth1 = C.C0A : orthogonal_from (z.M_1)
C.orth2 = C.C0A : orthogonal_from (z.M_2)
z.u = C.orth0.through
z.v = C.orth1.through
z.t = C.orth2.through
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(C_0,A C_1,B)

```

```

\tkzDrawCircles[thick,color=red](M_0,u M_1,v M_2,t)
\tkzDrawLines[add= .75 and .75](C_0,C_1 M_0,M_1)
\tkzDrawPoints(A,B,C_0,C_1,M_0,M_1,M_2)
\tkzLabelPoints[below right](A,B,C_0,C_1,M_0,M_1,M_2)
\tkzLabelLine[pos=2,right]( M_0,M_1){$(\Delta)$}
\end{tikzpicture}

```

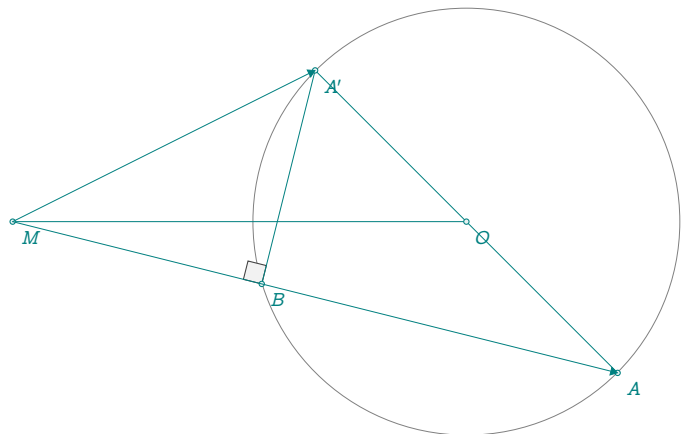


23.42 Puissance v1

```

\begin{tkzelements}
  z.O      = point : new (0,0)
  z.A      = point : new (2,-2)
  z.M      = point : new (-6,0)
  L.AM     = line : new (z.A,z.M)
  C.OA     = circle : new (z.O,z.A)
  z.Ap     = C.OA : antipode (z.A)
  z.B      = intersection (L.AM, C.OA)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircle(O,A)
  \tkzMarkRightAngle[fill=gray!10](A',B,M)
  \tkzDrawSegments(M,O A,A' A',B)
  \tkzDrawPoints(O,A,A',M,B)
  \tkzLabelPoints(O,A,A',M,B)
  \tkzDrawSegments[-Triangle](M,A M,A')
\end{tikzpicture}

```

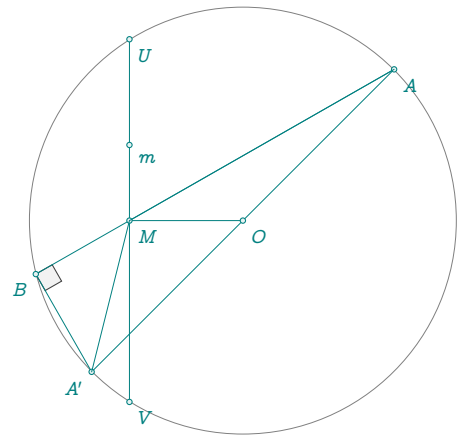


23.43 Puissance v2

```

\begin{tkzelements}
z.O      = point : new (0,0)
z.A      = point : new (2,2)
z.M      = point : new (-1.5,0)
L.AM     = line : new (z.A,z.M)
C.OA     = circle : new (z.O,z.A)
z.Ap     = C.OA : antipode (z.A)
_,z.B    = intersection (L.AM, C.OA)
z.m      = z.M : north(1)
L.mM     = line : new (z.m,z.M)
z.U,z.V  = intersection (L.mM,C.OA)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircle(O,A)
\tkzMarkRightAngle[fill=gray!10](A',B,M)
\tkzDrawSegments(M,O A,A' A',B A,B U,V)
\tkzDrawPoints(O,A,A',M,B,U,V,m)
\tkzLabelPoints(O,A,M,U,V,m)
\tkzLabelPoints[below left](A',B)
\tkzDrawSegments(M,A M,A')
\end{tikzpicture}

```



23.44 Reim v1

```

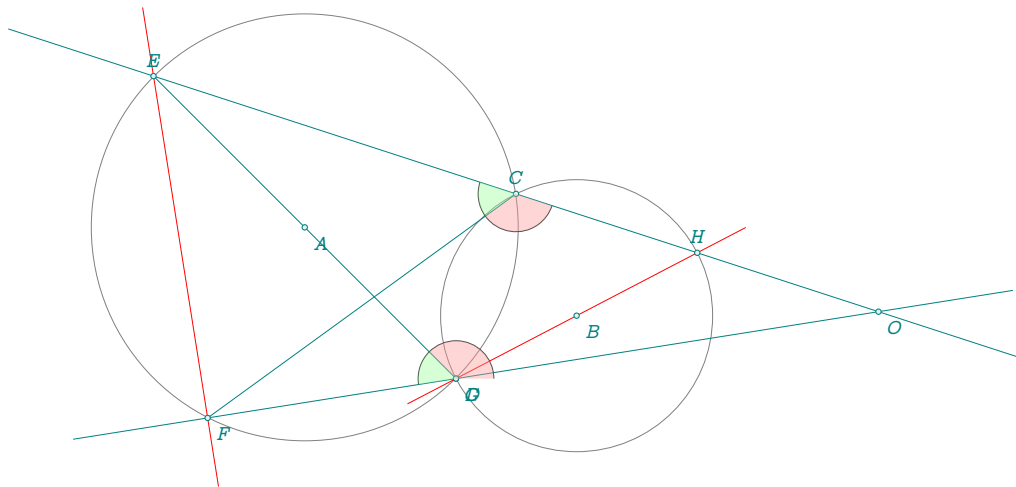
\begin{tkzelements}
z.A      = point: new (0,0)
z.E      = point: new (-2,2)
C.AE     = circle : new (z.A,z.E)
z.C      = C.AE : point (0.65)
z.D      = C.AE : point (0.5)
z.F      = C.AE : point (0.30)
L.EC     = line: new (z.E,z.C)
z.H      = L.EC : point (1.5)
T.CDH    = triangle : new (z.C,z.D,z.H)
z.B      = T.CDH.circumcenter
C.BD     = circle : new (z.B,z.D)
L.FD     = line: new (z.F,z.D)
z.G      = intersection (L.FD,C.BD)
z.O      = intersection (L.EC,L.FD)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(A,E B,H)
\tkzDrawSegments(E,D C,F)
\tkzDrawLines(E,O F,O)
\tkzDrawLines[red](E,F H,G)
\tkzDrawPoints(A,...,H,O)
\tkzLabelPoints(A,B,D,F,G,O)
\tkzLabelPoints[above](E,C,H)
\tkzMarkAngles[size=.5](E,C,F E,D,F)
\end{tikzpicture}

```

```

\tkzFillAngles[green!40,opacity=.4,size=.5](E,C,F E,D,F)
\tkzMarkAngles[size=.5](F,C,H G,D,E)
\tkzFillAngles[red!40,opacity=.4,size=.5](F,C,H G,D,E)
\end{tikzpicture}

```

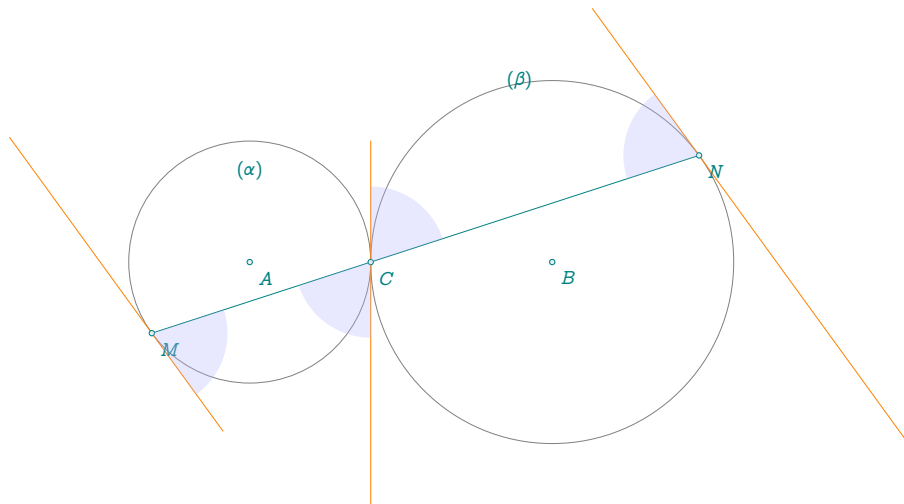


23.45 Reim v2

```

\begin{tkzelements}
  scale = .6
  z.A = point: new (0,0)
  z.B = point: new (10,0)
  z.C = point: new (4,0)
  C.AC = circle: new (z.A,z.C)
  z.c,z.cp = get_points (C.AC: tangent_at (z.C))
  z.M = C.AC: point (0.6)
  L.MC = line: new (z.M,z.C)
  C.BC = circle: new (z.B,z.C)
  z.N = intersection (L.MC,C.BC)
  z.m,z.mp = get_points (C.AC: tangent_at (z.M))
  z.n,z.np = get_points (C.BC: tangent_at (z.N))
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles(A,C B,C)
  \tkzDrawLines[new,add=1 and 1](M,m N,n C,c)
  \tkzDrawSegment(M,N)
  \tkzDrawPoints(A,B,C,M,N)
  \tkzLabelPoints[below right](A,B,C,M,N)
  \tkzFillAngles[blue!30,opacity=.3](m',M,C N,C,c' M,C,c n',N,C)
  \tkzLabelCircle[below=4pt,font=\scriptsize](A,C)(90){$(\alpha)$}
  \tkzLabelCircle[left=4pt,font=\scriptsize](B,C)(-90){$(\beta)$}
\end{tikzpicture}

```



23.46 Reim v3

```

\begin{tkzelements}
  z.A      = point: new (0,0)
  z.B      = point: new (8,0)
  z.C      = point: new (2,6)
  L.AB     = line : new (z.A,z.B)
  L.AC     = line : new (z.A,z.C)
  L.BC     = line : new (z.B,z.C)
  z.I      = L.BC : point (0.75)
  z.J      = L.AC : point (0.4)
  z.K      = L.AB : point (0.5)
  T.AKJ    = triangle : new (z.A,z.K,z.J)
  T.BIK    = triangle : new (z.B,z.I,z.K)
  T.CIJ    = triangle : new (z.C,z.I,z.J)
  z.x      = T.AKJ.circumcenter
  z.y      = T.BIK.circumcenter
  z.z      = T.CIJ.circumcenter
  C.xK     = circle: new (z.x,z.K)
  C.yK     = circle: new (z.y,z.K)
  z.O,_    = intersection (C.xK,C.yK)
  C.zO     = circle: new (z.z,z.O)
  L.KO     = line: new (z.K,z.O)
  z.D      = intersection (L.KO,C.zO)
\end{tkzelements}

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegments(K,D D,C)
  \tkzDrawPolygon[teal] (A,B,C)
  \tkzDrawCircles[orange] (x,A y,B z,C)
  \tkzDrawPoints[fill=white] (A,B,C,I,J,K,D)
  \tkzLabelPoints[below] (A,B,J,K,O)
  \tkzLabelPoints[above] (C,D,I)
  \tkzDrawPoints[fill=black] (O)
  \tkzLabelCircle[below=4pt,font=\scriptsize] (x,A) (20){$(\alpha)$}
  \tkzLabelCircle[left=4pt,font=\scriptsize] (y,B) (60){$(\beta)$}

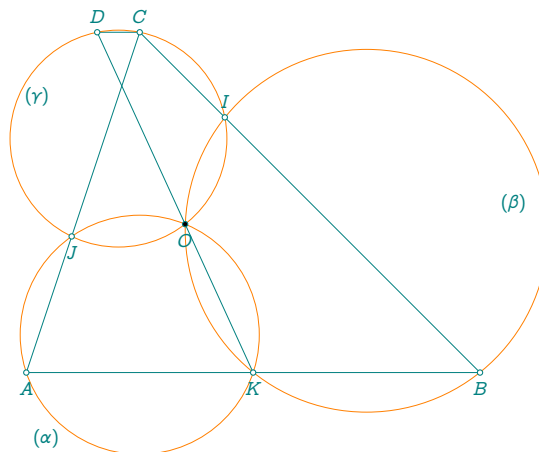
```



```

\tkzLabelCircle[below=4pt,font=\scriptsize](z,C)(60){$(\gamma)$}
\end{tikzpicture}

```

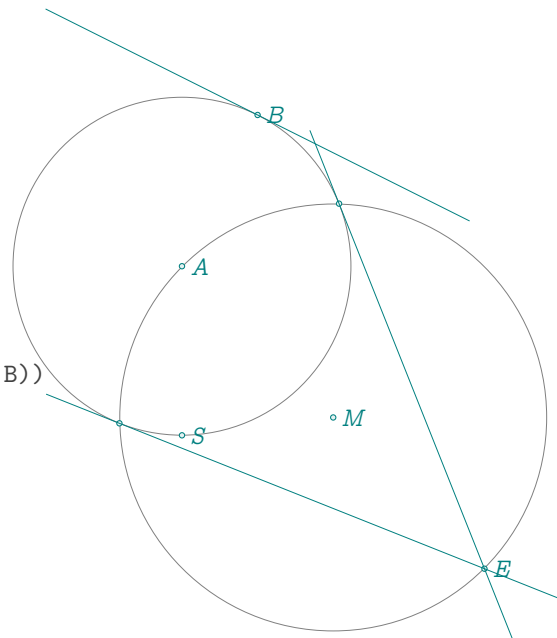


23.47 Tangente et cercle

```

\begin{tkzelements}
z.A      = point:    new (1,0)
z.B      = point:    new (2,2)
z.E      = point:    new (5,-4)
L.AE     = line :    new (z.A,z.E)
C.AB     = circle:   new (z.A ,z.B)
z.S      = C.AB.south
z.M      = L.AE.mid
L.Ti,L.Tj = C.AB:    tangent_from (z.E)
z.i      = L.Ti.pb
z.j      = L.Tj.pb
z.k,z.l  = get_points (C.AB: tangent_at (z.B))
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(A,B M,A)
\tkzDrawPoints(A,B,E,i,j,M,S)
\tkzDrawLines(E,i E,j k,l)
\tkzLabelPoints[right,font=\small](A,B,E,S,M)
\end{tikzpicture}

```

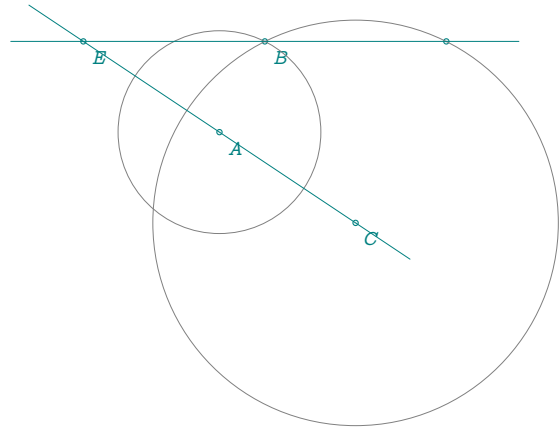


23.48 Homothétie

```

\begin{tkzelements}
z.A      = point: new (0,0)
z.B      = point: new (1,2)
z.E      = point: new (-3,2)
z.C,z.D = z.E : homothety(2,z.A,z.B)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(A,B,C,E,D)
\tkzLabelPoints(A,B,C,E)
\tkzDrawCircles(A,B C,D)
\tkzDrawLines(E,C E,D)
\end{tikzpicture}

```

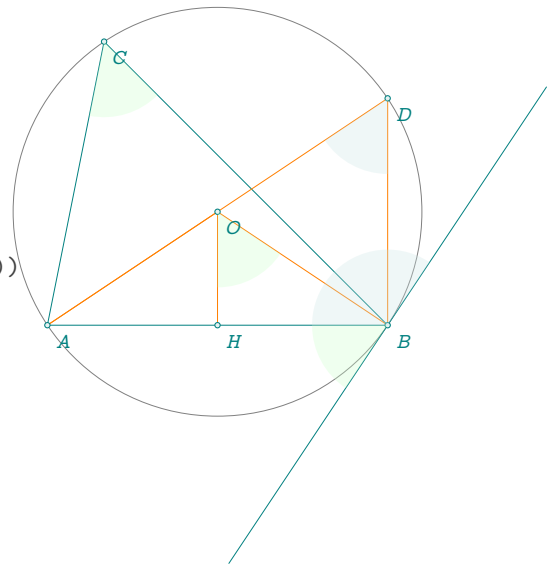


23.49 Tangente et corde

```

\begin{tkzelements}
scale    = .8
z.A      = point: new (0 , 0)
z.B      = point: new (6 , 0)
z.C      = point: new (1 , 5)
z.Bp     = point: new (2 , 0)
T.ABC    = triangle: new (z.A,z.B,z.C)
L.AB     = line: new (z.A,z.B)
z.O      = T.ABC.circumcenter
C.OA     = circle: new (z.O,z.A)
z.D      = C.OA: point (4.5)
L.AO     = line: new (z.A,z.O)
z.b1,z.b2 = get_points (C.OA: tangent_at (z.B))
z.H      = L.AB: projection (z.O)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircle(O,A)
\tkzDrawPolygon(A,B,C)
\tkzDrawSegments[new] (A,O B,O O,H A,D D,B)
\tkzDrawLine(b1,b2)
\tkzDrawPoints(A,B,C,D,H,O)
\tkzFillAngles[green!20,opacity=.3] (H,O,B A,C,B A,B,b1)
\tkzFillAngles[teal!20,opacity=.3] (A,D,B b2,B,A)
\tkzLabelPoints(A,B,C,D,H,O)
\end{tikzpicture}

```



23.50 Trois cordes

```

\begin{tkzelements}
z.O = point: new (0 , 0)
z.B = point: new (0 , 2)
z.P = point: new (1 , -.5)
C.OB = circle : new (z.O,z.B)
C.PB = circle : new (z.P,z.B)
_,z.A = intersection (C.OB,C.PB)
z.D = C.PB: point(0.85)

```

```

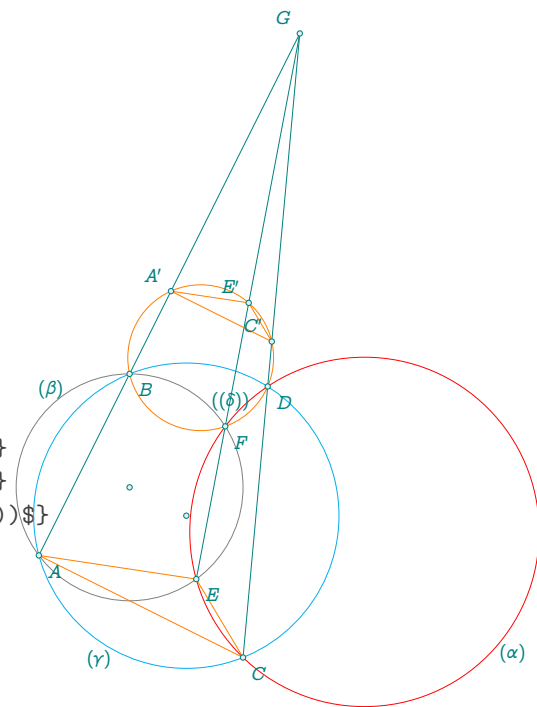
z.C = C.PB: point(0.5)
z.E = C.OB: point(0.6)
L.AB = line : new (z.A,z.B)
L.CD = line : new (z.C,z.D)
z.G = intersection (L.AB,L.CD)
L.GE = line : new (z.G,z.E)
z.F,_ = intersection (L.GE,C.OB)
T.CDE = triangle: new (z.C,z.D,z.E)
T.BFD = triangle: new (z.B,z.F,z.D)
z.w = T.CDE.circumcenter
z.x = T.BFD.circumcenter
L.GB = line : new (z.G,z.B)
L.GE = line : new (z.G,z.E)
L.GD = line : new (z.G,z.D)
C.xB = circle : new (z.x,z.B)
C.xF = circle : new (z.x,z.F)
C.xD = circle : new (z.x,z.D)
z.Ap = intersection (L.GB,C.xB)
z.Ep,_ = intersection (L.GE,C.xF)
z.Cp,_ = intersection (L.GD,C.xD)
\end{tkzelements}

```

```

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(O,B)
\tkzDrawCircles[cyan](P,B)
\tkzDrawCircles[red](w,E)
\tkzDrawCircles[new](x,F)
\tkzDrawSegments(A,G E,G C,G)
\tkzDrawPolygons[new](A,E,C A',E',C')
\tkzDrawPoints(A,...,G,A',E',C',O,P)
\begin{scope}[font=\scriptsize]
\tkzLabelPoints(A,...,F)
\tkzLabelPoints[above left](G,A',E',C')
\tkzLabelCircle[left](O,B)(30){$(\beta)$}
\tkzLabelCircle[below](P,A)(40){$(\gamma)$}
\tkzLabelCircle[right](w,C)(90){$(\alpha)$}
\tkzLabelCircle[left](x,B)(-230){$(\delta)$}
\end{scope}
\end{tikzpicture}

```

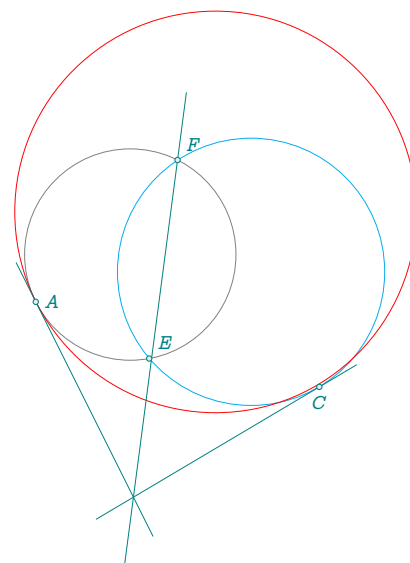


23.51 Trois tangentes

```

\begin{tkzelements}
  z.A   = point: new (-1 , 0)
  z.C   = point: new (4 , -1.5)
  z.E   = point: new (1 , -1)
  z.F   = point: new (1.5 , 2.5)
  T.AEF = triangle : new (z.A,z.E,z.F)
  T.CEF = triangle : new (z.C,z.E,z.F)
  z.w   = T.AEF.circumcenter
  z.x   = T.CEF.circumcenter
  C.wE  = circle : new (z.w,z.E)
  C.xE  = circle : new (z.x,z.E)
  L.Aw  = line : new (z.A,z.w)
  L.Cx  = line : new (z.C,z.x)
  z.G   = intersection (L.Aw,L.Cx)
  L.TA  = C.wE : tangent_at (z.A)
  L.TC  = C.xE : tangent_at (z.C)
  z.I   = intersection (L.TA,L.TC)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircles(w,E)
  \tkzDrawCircles[cyan](x,E)
  \tkzDrawCircles[red](G,A)
  \tkzDrawLines(A,I C,I F,I)
  \tkzDrawPoints(A,C,E,F)
  \tkzLabelPoints[right](A)
  \tkzLabelPoints[above right](E,F)
  \tkzLabelPoints[below](C)
\end{tikzpicture}

```

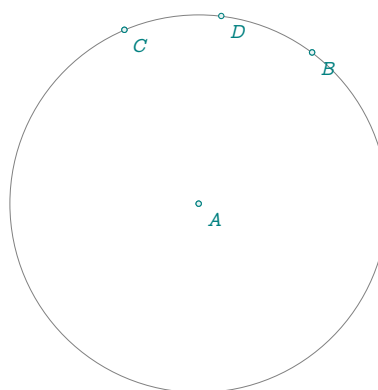


23.52 Midarc

```

\begin{tkzelements}
  z.A   = point: new (-1,0)
  z.B   = point: new (2,4)
  C.AB  = circle: new (z.A,z.B)
  z.C   = z.A: rotation (math.pi/3,z.B)
  z.D   = C.AB: midarc (z.B,z.C)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPoints(A,B,C)
  \tkzDrawCircles(A,B)
  \tkzDrawPoints(A,...,D)
  \tkzLabelPoints(A,...,D)
\end{tikzpicture}

```

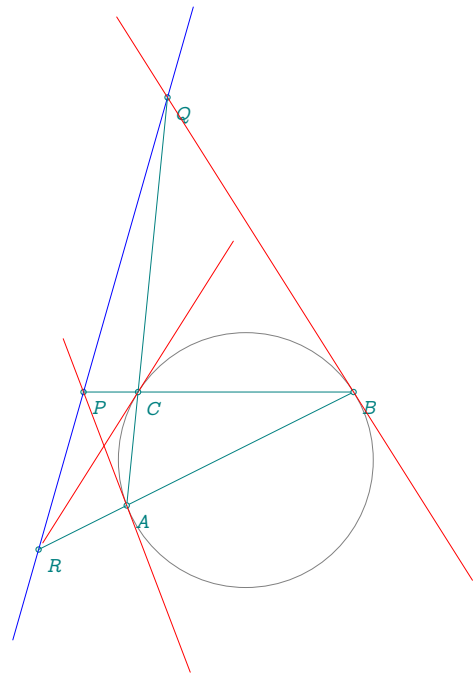


23.53 Droite de Lemoine sans macro

```

\begin{tkzelements}
  scale      = 1.6
  z.A        = point: new (1,0)
  z.B        = point: new (5,2)
  z.C        = point: new (1.2,2)
  T          = triangle: new(z.A,z.B,z.C)
  z.O        = T.circumcenter
  L.AB       = line: new (z.A,z.B)
  L.AC       = line: new (z.A,z.C)
  L.BC       = line: new (z.B,z.C)
  C.OA       = circle: new (z.O,z.A)
  z.Ar,z.A1  = get_points (C.OA: tangent_at (z.A))
  z.Br,z.B1  = get_points (C.OA: tangent_at (z.B))
  z.Cr,z.C1  = get_points (C.OA: tangent_at (z.C))
  L.tA       = line: new (z.Ar,z.A1)
  L.tB       = line: new (z.Br,z.B1)
  L.tC       = line: new (z.Cr,z.C1)
  z.P        = intersection (L.tA,L.BC)
  z.Q        = intersection (L.tB,L.AC)
  z.R        = intersection (L.tC,L.AB)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon[teal](A,B,C)
  \tkzDrawCircle(O,A)
  \tkzDrawPoints(A,B,C,P,Q,R)
  \tkzLabelPoints(A,B,C,P,Q,R)
  \tkzDrawLine[blue](Q,R)
  \tkzDrawLines[red](Ar,A1 Br,Q Cr,C1)
  \tkzDrawSegments(A,R C,P C,Q)
\end{tikzpicture}

```

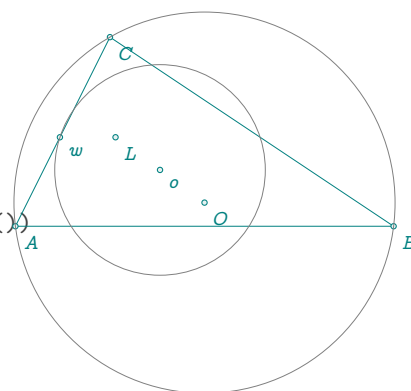


23.54 Premier cercle de Lemoine

```

\begin{tkzelements}
  z.A        = point: new (1,1)
  z.B        = point: new (5,1)
  z.C        = point: new (2,3)
  T          = triangle: new (z.A,z.B,z.C)
  z.O        = T.circumcenter
  z.o,z.w    = get_points (T : first_lemoine_circle ())
  z.L        = T : lemoine_point ()
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygons(A,B,C)
  \tkzDrawPoints(A,B,C,o,w,O,L)
  \tkzLabelPoints(A,B,C,o,w,O,L)
  \tkzDrawCircles(o,w O,A)
\end{tikzpicture}

```

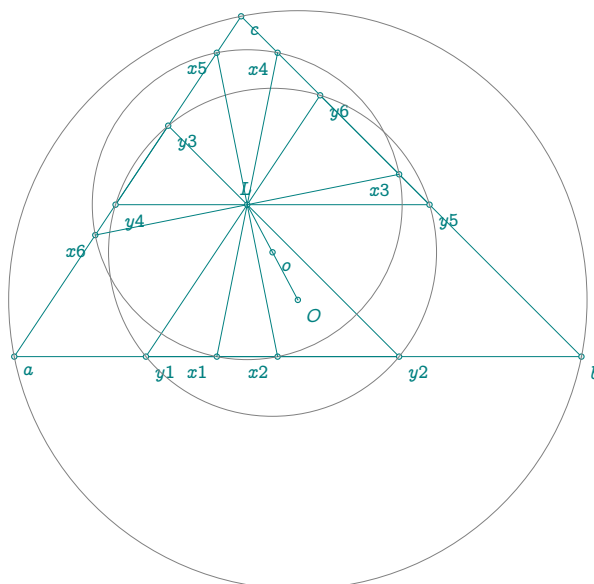


23.55 Premier et deuxième cercle de Lemoine

```

\begin{tkzelements}
  scale          = 2
  z.a            = point: new (0,0)
  z.b            = point: new (5,0)
  z.c            = point: new (2,3)
  T              = triangle: new (z.a,z.b,z.c)
  z.O            = T.circumcenter
  z.o,z.p        = get_points (T : first_lemoine_circle ())
  L.ab           = line : new (z.a,z.b)
  L.ca           = line : new (z.c,z.a)
  L.bc           = line : new (z.b,z.c)
  z.L,z.x        = get_points (T : second_lemoine_circle ())
  C.first_lemoine = circle : new (z.o,z.p)
  z.y1,z.y2      = intersection (L.ab,C.first_lemoine)
  z.y5,z.y6      = intersection (L.bc,C.first_lemoine)
  z.y3,z.y4      = intersection (L.ca,C.first_lemoine)
  C.second_lemoine = circle : new (z.L,z.x)
  z.x1,z.x2      = intersection (L.ab,C.second_lemoine)
  z.x3,z.x4      = intersection (L.bc,C.second_lemoine)
  z.x5,z.x6      = intersection (L.ca,C.second_lemoine)
  L.y1y6         = line : new (z.y1,z.y6)
  L.y4y5         = line : new (z.y4,z.y5)
  L.y2y3         = line : new (z.y2,z.y3)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygons(a,b,c y1,y2,y3,y4,y5,y6)
  \tkzDrawPoints(x1,x2,x3,x4,x5,x6,L)
  \tkzDrawPoints(a,b,c,o,0,y1,y2,y3,y4,y5,y6)
  \tkzLabelPoints[below right](a,b,c,o,0,y1,y2,y3,y4,y5,y6)
  \tkzLabelPoints[below left](x1,x2,x3,x4,x5,x6)
  \tkzLabelPoints[above](L)
  \tkzDrawCircles(L,x o,p 0,a)
  \tkzDrawSegments(L,0 x1,x4 x2,x5 x3,x6)
\end{tikzpicture}

```

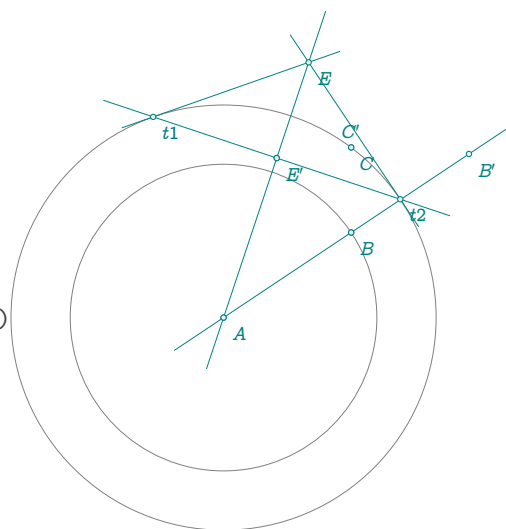


23.56 Inversion

```

\begin{tkzelements}
z.A = point: new (-1,0)
z.B = point: new (2,2)
z.C = point: new (2,4)
z.E = point: new (1,6)
C.AC = circle: new (z.A,z.C)
L.Tt1,
L.Tt2 = C.AC: tangent_from (z.E)
z.t1 = L.Tt1.pb
z.t2 = L.Tt2.pb
L.AE = line: new (z.A,z.E)
z.H = L.AE : projection (z.t1)
z.Bp,
z.Ep,
z.Cp = C.AC: inversion ( z.B, z.E, z.C )
\end{tkzelements}

```



```

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(A,B,C)
\tkzDrawCircles(A,C A,B)
\tkzDrawLines(A,B' E,t1 E,t2 t1,t2 A,E)
\tkzDrawPoints(A,B,C,E,t1,t2,H,B',E')
\tkzLabelPoints(A,B,C,E,t1,t2,B',E')
\tkzLabelPoints[above](C')
\end{tikzpicture}

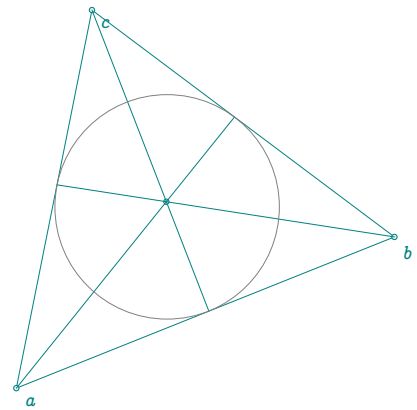
```

23.57 Point de Gergonne

```

\begin{tkzelements}
z.a = point: new(1,0)
z.b = point: new(6,2)
z.c = point: new(2,5)
T = triangle : new (z.a,z.b,z.c)
z.g = T : gergonne_point ()
z.i = T.incenter
z.ta,z.tb,z.tc = get_points (T : intouch ())
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygons(a,b,c)
\tkzDrawPoints(a,b,c,g)
\tkzLabelPoints(a,b,c)
\tkzDrawSegments (a,ta b,tb c,tc)
\tkzDrawCircle(i,ta)
\end{tikzpicture}

```



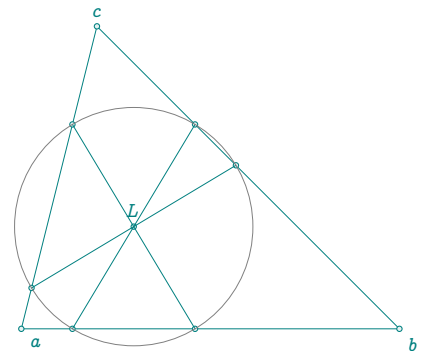
23.58 Antiparallèle au point de Lemoine

```

\begin{tkzelements}
z.a = point: new (0,0)
z.b = point: new (5,0)
z.c = point: new (1,4)
T = triangle: new (z.a,z.b,z.c)
z.L = T : lemoine_point ()
L.anti = T : antiparallel (z.L,0)
z.x_0,z.x_1 = get_points (L.anti)
L.anti = T : antiparallel (z.L,1)
z.y_0,z.y_1 = get_points (L.anti)
L.anti = T : antiparallel (z.L,2)
z.z_0,z.z_1 = get_points (L.anti)
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygons(a,b,c)
\tkzDrawPoints(a,b,c,L,x_0,x_1,y_0,y_1,z_0,z_1)
\tkzLabelPoints(a,b)
\tkzLabelPoints[above](L,c)
\tkzDrawSegments(x_0,x_1 y_0,y_1 z_0,z_1)
\tkzDrawCircle(L,x_0)
\end{tikzpicture}

```



23.59 Le cercle de Soddy sans fonction

```

\begin{tkzelements}
z.A = point : new ( 0 , 0 )
z.B = point : new ( 5 , 0 )
z.C = point : new ( 0.5 , 4 )
T.ABC = triangle : new ( z.A,z.B,z.C )
z.I = T.ABC.incenter

```

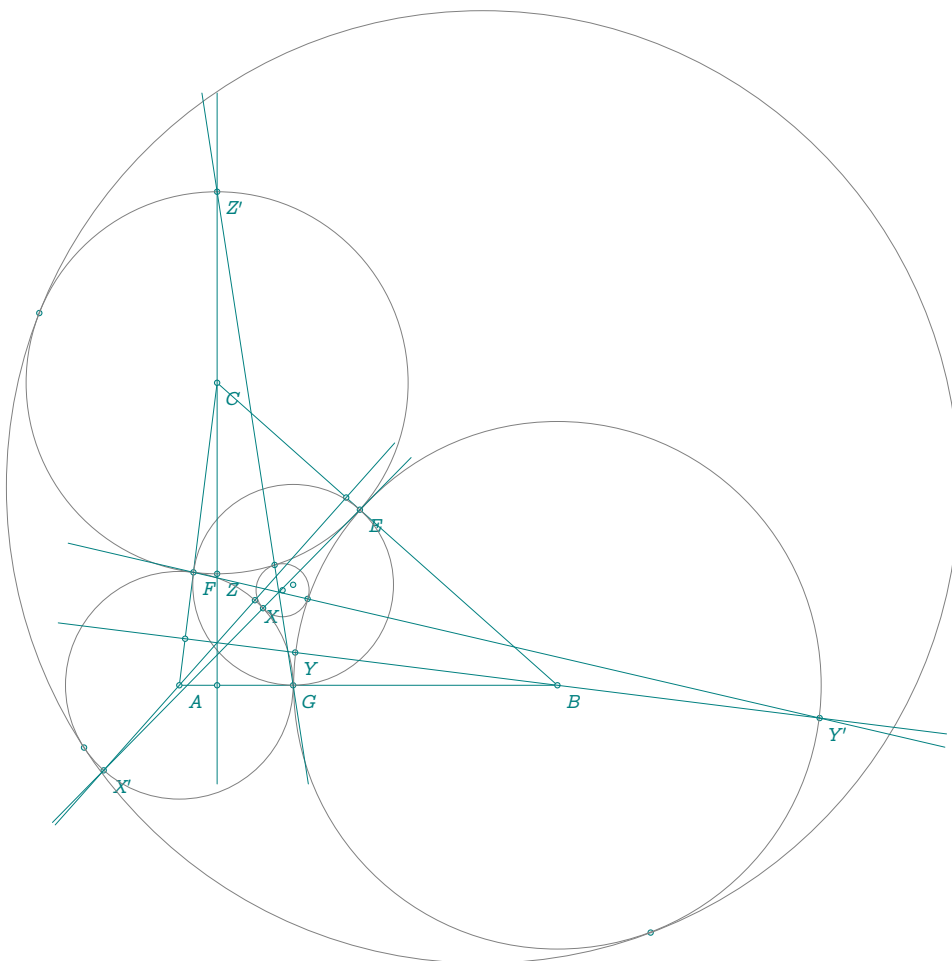


```

z.E,z.F,z.G = T.ABC : projection (z.I)
C.ins = circle : new (z.I,z.E)
T.orthic = T.ABC : orthic ()
z.Ha,z.Hb,z.Hc = get_points (T.orthic)
C.CF = circle : new ( z.C , z.F )
C.AG = circle : new ( z.A , z.G )
C.BE = circle : new ( z.B , z.E )
L.Ah = line : new ( z.A , z.Ha )
L.Bh = line : new ( z.B , z.Hb )
L.Ch = line : new ( z.C , z.Hc )
z.X,z.Xp = intersection (L.Ah,C.AG)
z.Y,z.Yp = intersection (L.Bh,C.BE)
z.Z,z.Zp = intersection (L.Ch,C.CF)
L.XpE = line : new (z.Xp,z.E)
L.YpF = line : new (z.Yp,z.F)
L.ZpG = line : new (z.Zp,z.G)
z.S = intersection (L.XpE,L.YpF)
z.Xi = intersection(L.XpE,C.AG)
z.Yi = intersection(L.YpF,C.BE)
_,z.Zi = intersection(L.ZpG,C.CF)
z.S = triangle : new (z.Xi,z.Yi,z.Zi).circumcenter
C.soddy_int = circle : new (z.S,z.Xi)
C.soddy_ext = C.ins : inversion (C.soddy_int)
z.w = C.soddy_ext.center
z.s = C.soddy_ext.through
z.Xip,z.Yip,z.Zip = C.ins : inversion (z.Xi,z.Yi,z.Zi)
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawPoints(A,B,C,E,F,G,Ha,Hb,Hc,X,Y,Z,X',Y',Z',Xi,Yi,Zi,I)
\tkzDrawPoints(Xi',Yi',Zi',S)
\tkzLabelPoints(A,B,C,E,F,G,X,Y,Z,X',Y',Z')
\tkzDrawCircles(A,G B,E C,F I,E S,Xi w,s)
\tkzDrawLines(X',Ha Y',Hb Z',Hc)
\tkzDrawLines(X',E Y',F Z',G)
\end{tikzpicture}

```



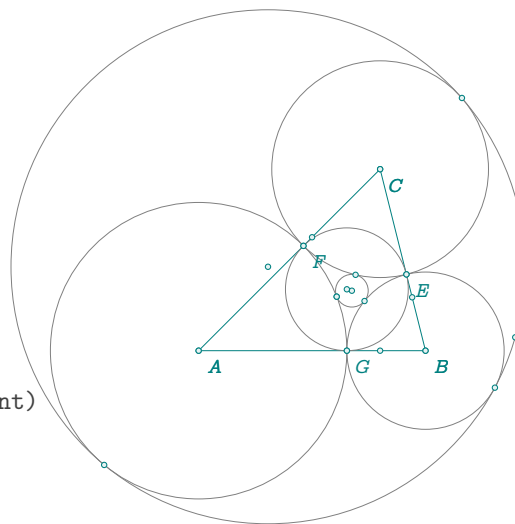
23.60 Cercle de Soddy avec fonction

```

\begin{tkzelements}
z.A = point : new ( 0 , 0 )
z.B = point : new ( 5 , 0 )
z.C = point : new ( 4 , 4 )
T.ABC = triangle : new ( z.A,z.B,z.C )
z.I = T.ABC.incenter
z.E,z.F,z.G = T.ABC : projection (z.I)
T.orthic = T.ABC : orthic ()
z.Ha,z.Hb,z.Hc = get_points (T.orthic)
C.ins = circle : new (z.I,z.E)
z.s,z.xi,z.yi,
z.zi = T.ABC : soddy_center ()
C.soddy_int = circle : new (z.s,z.xi)
C.soddy_ext = C.ins : inversion (C.soddy_int)
z.w = C.soddy_ext.center
z.t = C.soddy_ext.through
z.Xip,z.Yip,
z.Zip = C.ins : inversion (z.xi,z.yi,z.zi)
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)

```



```

\tkzDrawCircles(A,G B,E C,F I,E s,xi w,t)
\tkzDrawPoints(A,B,C,E,F,G,s,w,xi,t)
\tkzLabelPoints(A,B,C)
\tkzDrawPoints(A,B,C,E,F,G,Ha,Hb,Hc,xi,yi,zi,I)
\tkzDrawPoints(Xi',Yi',Zi')
\tkzLabelPoints(A,B,C,E,F,G)
\end{tikzpicture}

```

23.61 Chaîne de Pappus

```

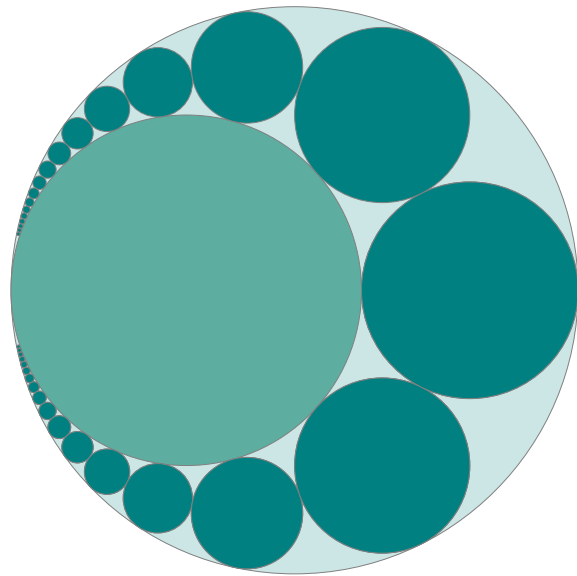
\begin{tkzelements}
  xC,nc    = 10,16
  xB       = xC/tkzphi
  xD       = (xC*xC)/xB
  xJ       = (xC+xD)/2
  r        = xD-xJ
  z.A      = point : new ( 0 , 0 )
  z.B      = point : new ( xB , 0 )
  z.C      = point : new ( xC , 0 )
  L.AC     = line : new (z.A,z.C)
  z.i      = L.AC.mid
  L.AB     = line:new (z.A,z.B)
  z.j      = L.AB.mid
  z.D      = point : new ( xD , 0 )
  C.AC     = circle: new (z.A,z.C)
  for i    = -nc,nc do
    z["J"..i] = point: new (xJ,2*r*i)
    z["H"..i] = point: new (xJ,2*r*i-r)
    z["J"..i.."p"], z["H"..i.."p"] = C.AC : inversion (z["J"..i],z["H"..i])
    L.AJ      = line : new (z.A,z["J"..i])
    C.JH      = circle: new ( z["J"..i] , z["H"..i])
    z["S"..i], z["T"..i] = intersection (L.AJ,C.JH)
    z["S"..i.."p"], z["T"..i.."p"] = C.AC : inversion (z["S"..i],z["T"..i])
    L.SpTp    = line:new ( z["S"..i.."p"], z["T"..i.."p"])
    z["I"..i] = L.SpTp.mid
  end
\end{tkzelements}

```

```

\def\nc{\tkzUseLua{\nc}}
\begin{tikzpicture}[ultra thin]
  \tkzGetNodes
  \tkzDrawCircle[fill=teal!20](i,C)
  \tkzDrawCircle[fill=PineGreen!60](j,B)
  \foreach \i in {-\nc,...,0,...,\nc} {
    \tkzDrawCircle[fill=teal]({I\i},{S\i'})
  }
\end{tikzpicture}

```



23.62 Trois cercles

```

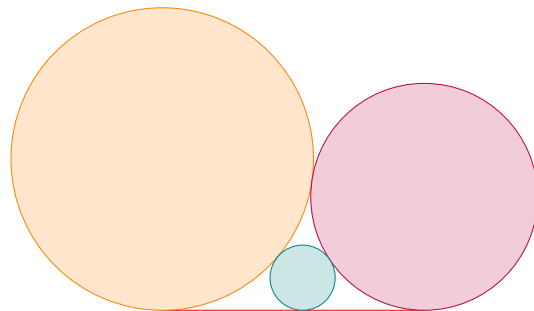
\begin{tkzelements}
function threecircles(c1,r1,c2,r2,c3,h1,h3,h2)
  local xk = math.sqrt (r1*r2)
  local cx = (2*r1*math.sqrt(r2))/(math.sqrt(r1)+math.sqrt(r2))
  local cy = (r1*r2)/(math.sqrt(r1)+math.sqrt(r2))^2
  z[c2] = point : new ( 2*xk , r2 )
  z[h2] = point : new (2*xk,0)
  z[c1] = point : new (0,r1)
  z[h1] = point : new (0,0)
  L.h1h2 = line: new(z[h1],z[h2])
  z[c3] = point : new (cx,cy)
  z[h3] = L.h1h2: projection (z[c3])
end
threecircles("A",4,"B",3,"C","E","G","F")
\end{tkzelements}

```

```

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegment[color = red](E,F)
  \tkzDrawCircle[orange,fill=orange!20](A,E)
  \tkzDrawCircle[purple,fill=purple!20](B,F)
  \tkzDrawCircle[teal,fill=teal!20](C,G)
\end{tikzpicture}

```




```
\tkzDrawPoints(D_2,E_2,F_2,G_2)
\tkzDrawPoints[red](F_1)
\tkzLabelPoints(A,B,C,O_Q,O_2)
\tkzLabelPoints[below](O_1,G)
\tkzLabelPoints[above right](D,H)
\tkzLabelPoints[above left](E,E_1,E_2)
\tkzLabelPoints[below left](F,F_1,F_2)
\tkzLabelPoints(D_1,G_1)
\tkzLabelPoints(D_2,G_2)
\end{tikzpicture}
```

Index

attribut, 19

circle: attribute
center, 43
ct, 43
east, 43
north, 43
opp, 43
radius, 43
south, 43
through, 43
type, 43
west, 43

circle: function
diameter(A,B), 44
new(O,A), 44
radius(O,r), 44

circle: method
Circles_position, 50
antipode (pt), 44
circles_position (C1), 44
common_tangent (C), 44
draw (), 44
external_similitude (C), 44
in_out (pt), 44
in_out_disk (pt), 44
in_out_disk, 52
in_out, 52
internal_similitude (C), 44
inversion (...), 44
inversion (line), 44
inversion (pt), 44
midarc (pt,pt), 44
midcircle (C), 44
orthogonal_from (pt), 44
orthogonal_through (pta,ptb), 44
point (r), 44
power (pt), 44
radical_axis (C), 44
radical_center (C1<,C2>), 44
radical_circle (C1<,C2>), 44
random_pt(lower, upper), 44
tangent_at (pt), 44
tangent_common, 50
tangent_from (pt), 44

Class
circle, 9, 43, 44
class, 24
ellipse, 9, 63
line, 9, 32
matrix, 81
parallelogram, 9, 75
point, 9, 25
Quadrilateral, 68
quadrilateral, 9
rectangle, 9, 72
regular_polygon, 9, 77
square, 9, 70
triangle, 9, 54
vector, 9, 78

ellipse: attribute
,, 63
Fa, 63
Fb, 63
Rx, 63
Ry, 63
center, 63, 65
covertex, 63, 65
slope, 63
type, 63
vertex, 63, 65

ellipse: method
east, 63
foci (f1,f2,v), 65
foci, 65
in_out (pt) , 65
new (pc, pa ,pb) , 65
new, 65
north, 63
nouveau, 65
orthoptic_circle () , 65
point (t) , 65
point, 66
radii (c,a,b,s1) , 65
radii, 66
south, 63
tangent_at (pt) , 65
tangent_from (pt) , 65
west, 63

Environment
luacode, 9, 14
tikzpicture, 14, 18, 64
tkzelements, 9, 14, 18, 19, 22, 64, 66

line: attribute
east, 32
length, 32
mid, 32
north_pa, 32
north_pb, 32
pa, 32
pb, 32
slope, 32
south_pa, 32
south_pb, 32
type, 32
vec, 32
west, 32

line: function
new(pt, pt), 34

line: method

_east(d), 34
 _north_pa(d), 34
 _north_pb(d), 34
 _south_pa(d), 34
 _south_pb(d), 34
 _west(d), 34
 apollonius (r), 35
 apollonius, 41
 as (r,an), 34
 barycenter (r,r), 34
 barycenter, 39
 cheops (), 34, 37
 circle (), 35
 circle_swap (), 35
 colinear_at, 38
 distance (pt), 35
 distance, 41
 divine (), 34, 37
 egyptian (), 34
 equilateral (<swap>), 34
 euclide (<swap>), 34
 gold (<swap>), 34, 37
 gold_ratio (), 34
 golden (<swap>), 34, 37
 harmonic_both (r), 34
 harmonic_ext (pt), 34
 harmonic_int (pt), 34
 in_out (pt), 35
 in_out_segment (pt), 35
 in_out, 105
 isosceles (an<,swap>), 34
 ll_from (pt), 34
 ll_from, 39
 mediator (), 34
 midpoint (), 34
 new, 33
 normalize (), 34
 normalize_inv (), 34
 normalize, 38
 ortho_from (pt), 34
 point (r), 34
 point, 37
 projection (obj), 35
 projection, 39
 pythagoras (), 37
 reflection (obj), 35
 reflection, 41
 report(d,pt), 34
 report, 35
 sa (r,an), 34
 school (), 34
 slope (), 35
 square (), 34
 sss (r,r), 34
 sublime (), 37
 translation (obj), 35
 translation, 40
 two_angles (an,an), 34
 two_angles, 35
 \LuaCodeDebugOn, 19
 math: function
 altitude (z1,z2,z3), 90
 angle_normalize (an) , 90
 bisector (z1,z2,z3), 90
 bisector_ext (z1,z2,z3), 90
 get_angle (z1,z2,z3), 90
 islinear (z1,z2,z3) , 90
 isortho (z1,z2,z3), 90
 length (a,b) , 90
 real (v) , 90
 solve_quadratic (a,b,c) , 90
 tkzinphi, 90
 tkzphi, 90
 tkzsqrtphi, 90
 valeur (r) , 36
 valeur (v) , 21
 value (v) , 90
 math: method
 aligned, 106
 islinear, 105, 106
 isortho, 106
 matrix: attribute
 cols, 82
 det, 82
 rows, 82
 set, 82
 type, 82
 matrix: function
 htm(), 84
 new(...), 84
 square(), 84
 vector(), 84
 matrix: metamethod
 __add(M1,M2), 83
 __eq(M1,M2), 83
 __mul(M1,M2), 83
 __pow(M,n), 83
 __sub(M1,M2), 83
 __tostroing(M,n), 83
 __unm(M, 83
 matrix: method
 adjugate(), 84
 adjugate, 88
 eq, 83
 get(), 84
 get, 87
 homogenization(), 84
 htm_apply(...), 84
 identity, 88
 inverse(), 84
 inverse, 87
 is_diagonal(), 84
 is_diagonal, 89
 is_orthogonal(), 84
 is_orthogonal, 89

- print(s,n), 84
- print, 81, 86
- transpose(), 84
- transpose, 88
- misc: function
 - barycenter ({z1,n1},{z2,n2}, ...), 90
- obj: method
 - new, 24
- Object
 - circle, 24
 - ellipse, 24
 - line, 24, 34
 - parallelogram, 24
 - point, 24, 28
 - quadrilateral, 24
 - rectangle, 24
 - regular_polygon, 24
 - square, 24
 - triangle, 24
- Package
 - ifthen, 18, 104
 - luacode, 9
 - tkz-elements.sty, 9
- package: function
 - \tkzUseLua, 91
 - set_lua_to_tex (list), 90
 - tkzUseLua (variable), 65, 90
- parallelogram: attribute
 - ab, 75
 - ac, 75
 - ad, 75
 - bc, 75
 - bd, 75
 - cd, 75
 - i, 75
 - pa, 75
 - pb, 75
 - pc, 75
 - pd, 75
 - type, 75
- parallelogram: method
 - fourth (za,zb,zc), 76
 - fourth, 76
- point: attribute
 - argument, 25
 - im, 25
 - modulus, 25
 - re, 25
 - type, 25
- point: function
 - new(r,r), 28
 - polar (d,an), 28
 - polar_deg (d,an), 28
- point: metamethod
 - __add(z1,z2), 101
 - __concat(z1,z2), 101
 - __div(z1,z2), 101
 - __eq(z1,z2), 101
 - __mul(z1,z2), 101
 - __pow(z1,z2), 101
 - __sub(z1,z2), 101
 - __tonumber(z), 101
 - __tostring(z), 101
 - __unm(z), 101
- point: method
 - Orthogonal (d), 30
 - abs (z), 101
 - arg (z), 101
 - at (), 28
 - at, 30
 - conj(z), 101
 - east(r), 28
 - get(z), 101
 - get_points (obj), 28
 - homothety(r,obj), 28
 - mod(z), 101
 - norm (z), 101
 - normalize (), 29
 - normalize(), 28
 - north (d), 28
 - north(r), 28
 - orthogonal (d), 28
 - polar, 29
 - print(), 28
 - rotation of points, 30
 - rotation(an , obj), 28
 - rotation, 31
 - south(r), 28
 - sqrt(z), 101
 - symmetry(obj), 28
 - symmetry, 31
 - west(r), 28
- prime, 18
- quadrilateral: attribute
 - ab, 68
 - ac, 68
 - ad, 68
 - a, 68
 - bc, 68
 - bd, 68
 - b, 68
 - cd, 68
 - c, 68
 - d, 68
 - g, 68
 - i, 68
 - pa, 68
 - pb, 68
 - pc, 68
 - pd, 68
 - type, 68
- quadrilateral: method
 - iscyclic (), 69

- rectangle: attribute
 - ab, 72
 - ac, 72
 - ad, 72
 - bc, 72
 - bd, 72
 - cd, 72
 - center, 72
 - diagonal, 72
 - length, 72
 - pa, 72
 - pb, 72
 - pc, 72
 - pd, 72
 - type, 72
 - width, 72
- rectangle: method
 - angle (zi,za,angle), 73
 - angle, 73
 - diagonal (za,zc), 73
 - diagonal, 74
 - get_lengths (), 73
 - gold (za,zb), 73
 - gold, 74
 - side (za,zb,d), 73
 - side, 73
- regular_polygon: method
 - incircle (), 77
 - name (string), 77
 - new(O,A,n), 77
- regular: attribute
 - angle, 77
 - center, 77
 - circle, 77
 - extradius, 77
 - inradius, 77
 - proj, 77
 - side, 77
 - table, 77
 - through, 77
 - type, 77
- souigné, 19
- square: attribute
 - ab, 70
 - ac, 70
 - ad, 70
 - bc, 70
 - bd, 70
 - cd, 70
 - center, 70
 - extradius, 70
 - inradius, 70
 - pa, 70
 - pb, 70
 - pc, 70
 - pd, 70
 - proj, 70
 - side, 70
 - type, 70
- square: method
 - rotation (zi,za), 71
 - side (za,zb), 71
 - side, 71
- \tkzDrawEllipse, 64
- \tkzGetNodes, 12, 14, 18, 22, 100
- \tkzUseLua(value), 23
- \tkzUseLua, 64
- triangle: attribute
 - ab, 54
 - alpha, 54
 - a, 54
 - bc, 54
 - beta, 54
 - b, 54
 - ca, 54
 - centroid, 54
 - circumcenter, 54
 - c, 54
 - eulercenter, 54
 - gamma, 54
 - incenter, 54
 - orthocenter, 54
 - pa, 54
 - pb, 54
 - pc, 54
 - spiekercenter, 54
 - type, 54
- triangle: method
 - altitude (n) , 56
 - anti () , 57
 - antiparallel(pt,n), 56
 - area (), 57
 - barycenter (ka,kb,kc), 56
 - barycentric_coordinates (pt), 57
 - base (u,v) , 56
 - bevan_point (), 56
 - bisector (n) , 56
 - bisector_ext(n) , 56
 - cevian (pt), 57
 - cevian_circle (), 56
 - cevian_circle, 57
 - cevian, 57
 - check_equilateral (), 57
 - circum_circle (), 56
 - conway_circle (), 56
 - conway_circle, 58
 - conway_points (), 56
 - conway_points, 58
 - euler (), 57
 - euler_circle (), 56
 - euler_ellipse (), 57
 - euler_line () , 56
 - euler_points () , 56
 - ex_circle (n), 56

excentral () ,57
extouch () ,57
feuerbach () ,57
feuerbach_point () ,56
first_lemoine_circle () ,56
gergonne_point () ,56
in_circle () ,56
in_out (pt) ,57
incentral () ,57
intouch () ,57
lemoine_point () ,56
medial () ,57
mittenpunkt_point () ,56
nagel_point () ,56
new, 54, 56
nine_points () ,56
orthic () ,57
parallelogram () ,56
pedal (pt) ,57
pedal_circle () ,56
pedal_circle, 58
pedal, 58
projection (p) ,56
second_lemoine_circle () ,56
spieker_center () ,56
spieker_circle () ,56
steiner_circumellipse () ,57
steiner_inellipse () ,57
symmedian () ,57
symmedian_line (n) ,56
symmedian_point () ,56
tangential () ,57

vector: attribute
head, 78
length, 78
mtx, 78
slope, 78
tail, 78
type, 78

vector: method
__add (u,v) ,79
__mul (k,u) ,79
__sub (u,v) ,79
__unm (u) ,79
at (V) ,79
new(pt, pt) ,79
normalize(V) ,79
orthogonal(d) ,79
scale(d) ,79

24 Aide-mémoire

r désigne un nombre réel, cx un nombre complexe, d un nombre réel positif, n un entier, an un angle, b un booléen, s une chaîne de caractères, pt un point, t une table, m une matrice, v une variable, L une droite, C un cercle, T un triangle, E une ellipse, V un vecteur, Q un quadrilatère, P un parallélogramme, R un rectangle, S un carré, RP un polygone régulier, M une matrice, O un objet (pt, L, C, T), .. une liste de points ou un objet, <> argument facultatif.

point		Methods table(6)		triangle	
Attributes table(1)		new (pt,pt)	-> d	Attributes table(9)	
re	-> r	distance (pt)	-> d	pa,pb,pc	-> pt
im	-> r	slope ()	-> r	circumcenter	-> pt
type	-> s	in_out (pt)	-> b	centroid	-> pt
argument	-> r	in_out_segment (pt)	-> b	incenter	-> pt
modulus	-> d	barycenter (r,r)	-> pt	eulercenter	-> pt
Functions table(1)		point (t)	-> pt	orthocenter	-> pt
new	-> pt	midpoint ()	-> pt	spiekercenter	-> pt
polar	-> pt	harmonic_int (pt)	-> pt	type	-> s
polar_deg	-> pt	harmonic_ext (pt)	-> pt	a	-> d
Methods table(30)		harmonic_both (d)	-> pt	b	-> d
+ - * / (pt,pt)	-> pt	gold_ratio()	-> pt	c	-> d
.. (pt,pt)	-> r	normalize ()	-> pt	ab	-> L
^ (pt,pt)	-> r	normalize_inv ()	-> pt	bc	-> L
=	-> b	_north_pa (d)	-> pt	ca	-> L
tostring	-> s	_north_pb (d)	-> pt	alpha	-> r
Methods table(2) table(31)		_south_pa (d)	-> pt	beta	-> r
conj	-> pt	_south_pb (d)	-> pt	gamma	-> r
abs	-> r	_east (d)	-> pt	Methods table(10)	
mod	-> d	_west (d)	-> pt	new (pt,pt,pt)	-> pt
norm	-> d	report (r,pt)	-> pt	trilinear (r,r,r)	-> pt
arg	-> d	colinear_at (pt,k)	-> pt	barycentric (r,r,r)	-> pt
get	-> r,r	translation (...)	-> O	bevan_point ()	-> pt
sqrt	-> pt	projection (...)	-> O	mittenpunkt_point ()	-> pt
north(d)	-> pt	reflection (...)	-> O	gergonne_point ()	-> pt
south(d)	-> pt	ll_from (pt)	-> L	nagel_point ()	-> pt
east(d)	-> pt	ortho_from (pt)	-> L	feuerbach_point ()	-> pt
west(d)	-> pt	mediator ()	-> L	lemoine_point()	-> pt
normalize(pt)	-> pt	circle ()	-> C	symmedian_point()	-> pt
symmetry (...)	-> O	circle_swap ()	-> C	spieker_center()	-> pt
rotation (an , ...)	-> O	diameter ()	-> C	barycenter (r,r,r)	-> pt
homothety (r , ...)	-> O	apollonius (r)	-> C	base (u,v)	-> pt
orthogonal(d)	-> pt	equilateral (<swap>)	-> T	euler_points ()	-> pt
at()	-> pt	isosceles (an,<swap>)	-> T	nine_points ()	-> pt
print()	-> s	school ()	-> T	point (t)	-> pt
		two_angles (an,an)	-> T	soddy_center ()	-> pt
		half ()	-> T	conway_points ()	-> pts
line		sss (r,r,r)	-> T	euler_line ()	-> L
Attributes table(3)		sas (r,an)	-> T	symmedian_line (n)	-> L
pa,pb	-> pt	ssa (r,an)	-> T	altitude (n)	-> L
type	-> s	gold (<swap>)	-> T	bisector (n)	-> L
mid	-> pt	euclide (<swap>)	-> T	bisector_ext(n)	-> L
north_pa	-> pt	golden (<swap>)	-> T	antiparallel(pt,n)	-> L
north_pb	-> pt	divine ()	-> T	euler_circle ()	-> C
south_pa	-> pt	cheops ()	-> T	circum_circle()	-> C
south_pb	-> pt	pythagoras ()	-> T	in_circle ()	-> C
east	-> pt	sublime ()	-> T	ex_circle (n)	-> C
west	-> pt	egyptian ()	-> T	first_lemoine_circle()	-> C
slope	-> r	square (<swap>)	-> T	second_lemoine_circle()	-> C
length	-> d			spieker_circle()	-> C
vec	-> V				

soddy_circle ()	-> C	midcircle(C)	-> C	ab bc cd da	-> L
conway_circle ()	-> C	external_tangent(C)	-> L,L	ac bd	-> L
pedal_circle ()	-> C	internal_tangent(C)	-> L,L	Methods table(19)	
cevian_circle ()	-> C	common_tangent(C)	-> L,L	new (pt,pt,pt,pt)	-> R
orthic()	-> T	tangent_from (pt)	-> L,L	angle (pt,pt,an)	-> R
medial()	-> T	inversion (...)	-> 0	gold (pt,pt,<swap>)	-> R
incentral()	-> T			diagonal (pt,pt,<swap>)	-> R
excentral()	-> T	ellipse		side (pt,pt,r,<swap>)	-> R
intouch()	-> T	Attributes table(13)		get_lengths ()	->r,r
contact()	-> T	center	-> pt		
extouch()	-> T	vertex	-> pt	quadrilateral	
feuerbach()	-> T	covertex	-> pt	Attributes table(14)	
anti ()	-> T	Fa	-> pt	pa,pb,pc,pd	-> pt
tangential ()	-> T	Fb	-> pt	ab bc cd da	-> L
cevian (pt)	-> T	north	-> pt	ac bd	-> L
symmedian ()	-> T	south	-> pt	type	-> s
euler ()	-> T	east	-> pt	i	-> pt
pedal (pt)	-> T	west	-> pt	g	-> pt
projection (pt)	-> pt,pt,pt	Rx	-> d	a b c d	-> r
parallelogram ()	-> pt	Ry	-> d	Methods table(15)	
area ()	-> d	slope	-> r	new (pt,pt,pt,pt)	-> Q
barycentric_coordinates(pt)		type	-> s	iscyclic ()	-> b
-> r,r,r		Methods table(13)			
in_out (pt)	-> pt	new (pt,pt,pt)	-> E	parallelogram	
check_equilateral ()	-> b	foci (pt,pt,pt)	-> E	Attributes table(20)	
		radii (pt,r,r,an)	-> E	pa,pb,pc,pd	-> pt
circle		in_out (pt)	-> b	ab bc cd da	-> L
Attributes table(7)		tangent_at (pt)	-> L	ac bd	-> L
center	-> pt	tangent_from (pt)	-> L	type	-> s
through	-> pt	point (r)	-> pt	center	-> pt
north	-> pt			Methods table(21)	
south	-> pt	square		new (pt,pt,pt,pt)	->
east	-> pt	Attributes table(16)		fourth (pt,pt,pt)	->
west	-> pt	pa,pb,pc,pd	-> pt		
opp	-> pt	type	-> s	Regular polygon	
type	-> s	side	-> d	Attributes table(22)	
radius	-> d	center	-> pt	center	-> pt
ct	-> L	exradius	-> d	through	-> pt
Methods table(8)		inradius	-> d	circle	-> C
new (pt,pt)	-> C	diagonal	-> d	type	-> s
radius (pt, r)	-> C	proj	-> pt	side	-> d
diameter (pt,pt)	-> C	ab bc cd da	-> L	exradius	-> d
in_out (pt)	-> b	ac bd	-> L	inradius	-> d
in_out_disk (pt)	-> b	Methods table(17)		proj	-> pt
circles_position (C)	-> s	new (pt,pt,pt,pt)	-> S	nb	-> i
power (pt)	-> r	rotation (pt,pt)	-> S	angle	-> an
antipode (pt)	-> pt	side (pt,pt,<swap>)	-> S	Methods table(23)	
midarc (pt,pt)	-> pt			new (pt,pt,n)	-> PR
point (r)	-> pt	rectangle		incircle ()	-> C
random_pt (lower, upper)	-> pt	Attributes table(18)		name (s)	-> ?
internal_similitude (C)	-> pt	pa,pb,pc,pd	-> pt		
external_similitude (C)	-> pt	type	-> s	vector	
radical_center(C,<C>)	-> pt	center	-> pt	Attributes table(24)	
tangent_at (pt)	-> L	exradius	-> d	type	-> s
radical_axis (C)	-> L	length	-> r	norm	-> d
radical_circle(C,<C>)	-> C	width	-> r	slope	-> r
orthogonal_from (pt)	-> C	diagonal	-> d	mtx	-> M
orthogonal_through(pt,pt)	-> C				

Methods table(25)	\wedge (m,n)	-> m	length	-> d
new (pt,pt)	-> V =	-> b	islinear(pt,pt,pt)	-> b
+ - *	-> pt tostring	-> s	isortho(pt,pt,pt)	-> b
normalize (V)	-> V Method table(28)		value{r}	-> r
orthogonal (d)	-> V print	-> s	real	-> r
scale (r)	-> V get	-> r/cx	angle_normalize (an)	-> an
at (pt)	-> V inverse	-> m	barycenter (...)	-> pt
matrix	adjugate	-> m	bisector (pt,pt,pt)	-> L
Attributes table(26)	transpose	-> m	bisector_ext (pt,pt,pt)	-> L
set	-> t is_diagonal	-> b	altitude (pt,pt,pt)	-> L
rows	-> n is_orthogonal	-> b	midpoint (pt,pt)	-> pt
cols	-> n homogenization	-> m	equilateral (pt,pt)	-> T
type	-> s htm_apply	-> m	format_number(r,n)	-> r
det	-> r		solve_quadratic(cx,cx,cx)	-> cx,cx
Functions table(28)	Misc.		\tkzUseLua{v}	-> s
new	-> m Attributes table(29)			
square	-> m scale (default =1)	-> r	Macros	
htm	-> m tkzphi	-> r	\tkzDN[n]{r}	-> r
vector	-> m tkzinvphi	-> r	\tkzDrawLuaEllipse((pt,pt,pt))	
Metamethods table(27)	tkzsqrtphi	-> r		
+ - * (m,m)	-> m tkz_epsilon (default=1e-8)	-> r		