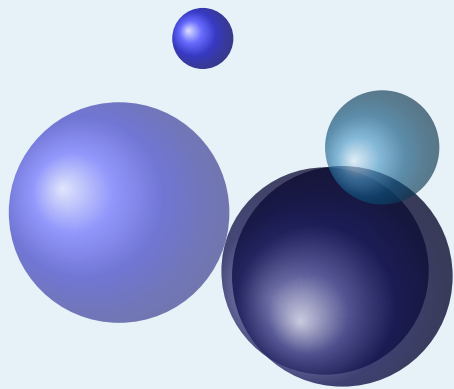


Euclidean Geometry 1.00



Alain Matthes

March 11, 2026 Documentation V.1.00

<http://altermundus.fr>

Euclidean Geometry

Alain Matthes

This document brings together a variety of geometric results, including definitions, theorems, lemmas, applications, and numerous examples. All figures are produced using [tkz-elements](#) and [tkz-euclide](#).

🔗 You will find this document, along with many additional examples, on my website: altermundus.fr.

Please report any typos or comments concerning this documentation to [Alain Matthes](#).

Contents

1	Notation	4	2.34	Inversion	31
2	Theory and Examples	4	2.35	Cocyclic points	32
2.1	Length transfer	4	2.36	Archimedes' Book of Lemmas: Proposition 1	33
2.2	Varignon's Theorem	4	2.37	D'Alembert's Theorem	34
2.3	Wittenbauer's Parallelogram	5	2.38	Altshiller-Court's theorem	35
2.4	Angle Bisector Theorem: Internal angle bisector	5	2.39	Reim's theorem 1	35
2.5	Angle Bisector Theorem: External angle bisector	6	2.40	Reim's theorem 2	36
2.6	Menelaus' theorem (weak version)	7	2.41	Three chords	37
2.7	Application: Pascal' theorem weak version	7	2.42	South Pole	39
2.8	Pascal's theorem with a triangle	9	2.43	Circle through incircle	39
2.9	Application of Pascal's theorem	9	2.44	Euler's relation	40
2.10	Construction of a harmonic division	11	2.45	Circle in circular segment	41
2.11	Harmonic division	12	2.46	Tangent-Chord Theorem	41
2.12	Harmonic division with golden ratio	12	2.47	Iran Lemma	42
2.13	Gold division	13	2.48	Tangent of two circles	43
2.14	Gold ratio with segment	14	2.49	Angle of two intersecting circles	47
2.15	Bisector and harmonic division	14	2.50	Specific lines of a triangle	49
2.16	Alternate segment theorem	15	2.51	Existence of the Euler line	50
2.17	Thébault's problem II	16	2.52	Viviani's Theorem	51
2.18	Apollonius circle	16	2.53	Reuschle's theorem	51
2.19	Midarc	17	2.54	Thébault's problem III	52
2.20	Orthogonal circles	17	2.55	Thébault's problem III with a method	53
2.21	Circle orthogonal to two given circles	19	2.56	Soddy circles of a triangle	53
2.22	Orthogonal Circles – Secant Criterion	19	2.57	Soddy circle without function	53
2.23	Radical center and radical circle	20	2.58	Soddy circle with function	54
2.24	Apollonius circles and circumcircle of a triangle	22	2.59	Six circles in a triangle	55
2.25	Apollonius circles of a triangle	23	2.60	Circle-Point Midpoint Theorem	56
2.26	Apollonius circles in a triangle with method	25	2.61	Bankoff circle	57
2.27	Power of a point with respect to a circle	25	2.62	Adams's circle	59
2.28	Properties of the power	26	2.63	Van Lamoen's circle	59
2.29	Radical axis	27	2.64	Yiu's circles variant one	60
2.30	Construction of the radical axis for two non-intersecting circles	29	2.65	Yiu's circles variant two	60
2.31	Powers of a point with respect to two circles	30	2.66	Yff's circles variant one	61
2.32	Pencil of circles	30	2.67	Yff's circles variant two	61
2.33	Fixed point pencil	30	2.68	Construction of Soddy's circles	62
			2.69	Radical axis of circumcircle and Apollonius circle	64
			2.70	Symmedian and tangents	64
			2.71	Gold Arbelos	64
			2.72	Lemoine	65

2.73	Lemoine point	66	3.8	Class circle: method CCP(C, p, mode)	102
2.74	Example: Cevian circle with orthocenter	67	3.9	Class circle: method CCL(C2, L)	104
2.75	Excircles	67	3.10	Case circle: method CCC(C2, C3 [, opts])	106
2.76	Morley triangle	68	3.11	Class circle: method CCC_gergonne(C_2, C_3)	109
2.77	Feuerbach theorem	68			
2.78	Orthopole and Simson line	69			
2.79	Gold Arbelos properties	69			
2.80	Apollonius circle v1 with inversion	70			
2.81	Construction of the midcircle of two disjoint circles	72			
2.82	Midcircles and Arbelos	74			
2.83	Circle and path	76			
2.84	Tangent and circle	76			
2.85	First Lemoine circle	76			
2.86	First and second Lemoine circles	77			
2.87	Pappus chain	78			
2.88	Three Circles	78			
2.89	Pentagons in a golden arbelos	79			
2.90	Polar and Pascal's theorem	80			
2.91	Ellipse	80			
2.92	Ellipse with radii	81			
2.93	Ellipse_with_foci	81			
2.94	Parabola with Focus, Axis of Symmetry, and a Point on the Curve	82			
2.95	Ellipse with Center, Vertex, and Covertex	82			
2.96	Ellipse with Foci and a Point	83			
2.97	Orthic_inellipse with search_ellipse	84			
2.98	Kiepert hyperbola	84			
2.99	Kiepert parabola	85			
2.100	Euler ellipse	86			
2.101	Related circles from Leon Bankoff	88			
3	Apollonius' problem	89			
3.1	Class point: method PPP(p, p)	90			
3.2	Class line: method LPP(p, p)	91			
3.3	Class line: method LLP(L, p)	93			
3.4	Class line: method LLL(L, L)	95			
3.5	Class: method CLL(L, L)	96			
3.6	Class circle: method CPP(p, p)	99			
3.7	Class circle: method CLP(L, p, mode)	100			

1 Notation

[?] Unless otherwise stated, the notation used throughout this document follows the conventions of classical French geometry and is consistent with the terminology adopted in the tkz-elements package.

- A *point* is denoted by a capital letter, such as A .
- The *line* determined by the points A and B is denoted by (AB) .
- The *segment* with endpoints A and B is denoted by $[AB]$. Its (unsigned) length is denoted by AB . The notation $\overline{AB}$ denotes the algebraic measure of the segment, that is, a signed length.
- The *ray* with origin A and passing through B is denoted by $[AB)$.
- The *vector* from A to B is denoted by $\overrightarrow{AB}$.
- The relations $(AB) \parallel (CD)$ and $(AB) \perp (CD)$ indicate that the lines (AB) and (CD) are parallel or perpendicular, respectively.
- The *angle* with vertex B and sides $[BA)$ and $[BC)$ is denoted by $\widehat{ABC}$.
- The notation ABC or $\triangle ABC$ denotes the *triangle* with vertices A , B , and C .
- The notation $C_{(O,A)}$ denotes the *circle* with centre O passing through the point A .
- The notation $C_{(O,r)}$ denotes the *circle* with centre O and radius r .

2 Theory and Examples

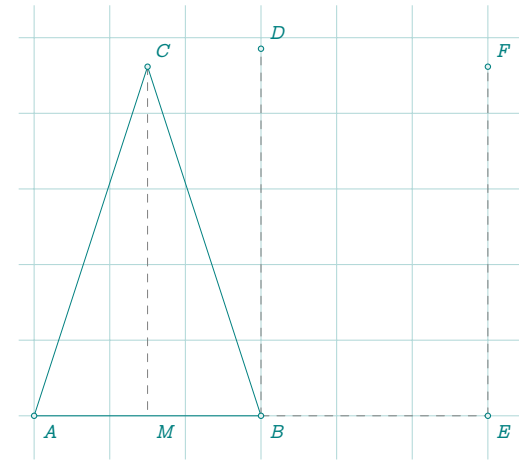
2.1 Length transfer

Example 1: Length transfer

Use of `north` and `east` functions linked to points, to transfer lengths

File `transfer.lua`:

```
init_elements()
z.A = point(0, 0)
z.B = point(3, 0)
L.AB = line(z.A, z.B)
T.ABC = L.AB:sublime()
z.C = T.ABC.pc
z.D = z.B:north(tkz.length(z.B, z.C))
z.E = z.B:east(L.AB.length)
z.M = L.AB.mid
z.F = z.E:north(tkz.length(z.C, z.M))
```



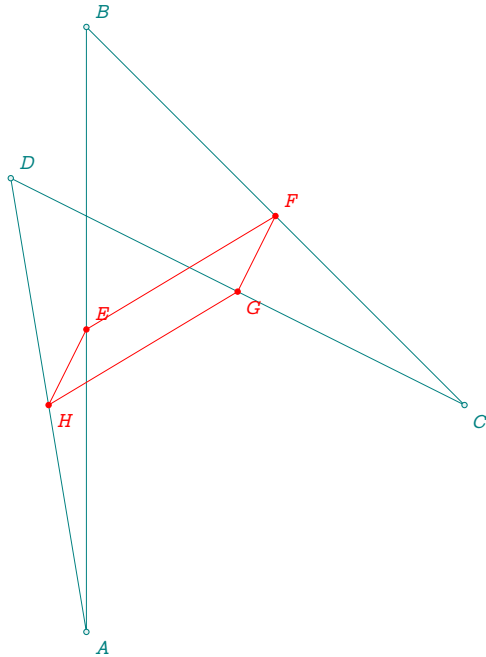
2.2 Varignon's Theorem

Theorem 1: Varignon's Theorem

In Euclidean geometry, Varignon's theorem holds that the midpoints of the sides of an arbitrary quadrilateral form a parallelogram, called the Varignon parallelogram. It is named after Pierre Varignon, whose proof was published posthumously in 1731. [Wikipedia]

```
init_elements()
z.A = point(0, 0)
z.B = point(0, 8)
z.C = point(5, 3)
z.D = point(-1, 6)
```

```
Q.ABCD = quadrilateral(z.A, z.B, z.C, z.D)
z.E, z.F,
z.G, z.H = tkz.midpoints(z.A, z.B, z.C, z.D)
```



2.3 Wittenbauer's Parallelogram

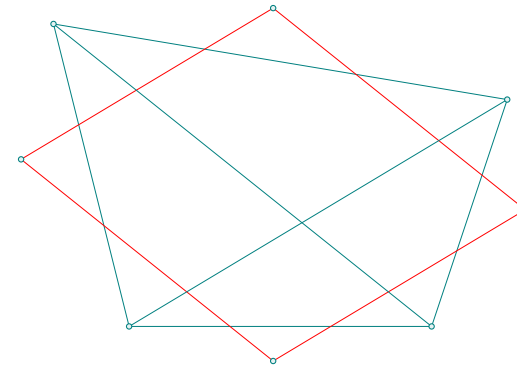
Theorem 2: Wittenbauer's Parallelogram

Divide the sides of a quadrilateral into three equal parts. The figure formed by connecting and extending adjacent points on either side of a polygon vertex is a parallelogram known as Wittenbauer's parallelogram.

Weisstein, Eric W. "Wittenbauer's Parallelogram." From MathWorld—A Wolfram Web Resource.

```
init_elements()
z.A = point(0, 0)
z.B = point(4, 0)
```

```
z.C = point(5, 3)
z.D = point(-1, 4)
Q.ABCD = quadrilateral(z.A, z.B, z.C, z.D)
z.P_1 = Q.ABCD.ab:point(1 / 3)
z.P_2 = Q.ABCD.ab:point(2 / 3)
z.P_3 = Q.ABCD.bc:point(1 / 3)
z.P_4 = Q.ABCD.bc:point(2 / 3)
z.P_5 = Q.ABCD.cd:point(1 / 3)
z.P_6 = Q.ABCD.cd:point(2 / 3)
z.P_7 = Q.ABCD.da:point(1 / 3)
z.P_8 = Q.ABCD.da:point(2 / 3)
L.P18 = line(z.P_1, z.P_8)
L.P23 = line(z.P_2, z.P_3)
L.P45 = line(z.P_4, z.P_5)
L.P67 = line(z.P_6, z.P_7)
z.K = intersection(L.P18, L.P23)
z.L = intersection(L.P23, L.P45)
z.M = intersection(L.P45, L.P67)
z.N = intersection(L.P67, L.P18)
```



2.4 Angle Bisector Theorem: Internal angle bisector

Theorem 3: Internal angle bisector

In a triangle, the angle bisector of any angle will divide the opposite side in the ratio of the sides containing the angle.

Example 2: Alternate Interior Angles

In the triangle ABC , let the angle at A be bisected by the segment AD . The ray AD meets the line through C parallel to AB at a point E .

Assume that

$$AC = CE.$$

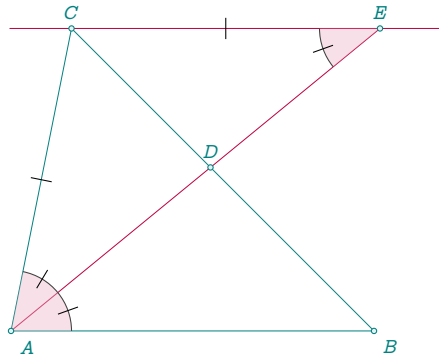
Proof. Let E be the intersection of (AD) and the line parallel to (AB) through C . $\widehat{AEC} = \widehat{BAE}$, meaning that ACE is isosceles and thus $AC = CE$. The triangles ABD and ECD are similar, implying

$$\frac{AB}{CE} = \frac{BD}{CD}$$

With $CE = AC$, we obtain $\frac{AB}{AC} = \frac{BD}{CD}$.

File internal.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(1, 5)
T = triangle(z.A, z.B, z.C)
z.I = T.incenter
L.AI = line(z.A, z.I)
z.D = intersection(L.AI, T.bc)
L.LL = T.ab:ll_from(z.C)
L.AD = line(z.A, z.D)
z.E = intersection(L.LL, L.AD)
```

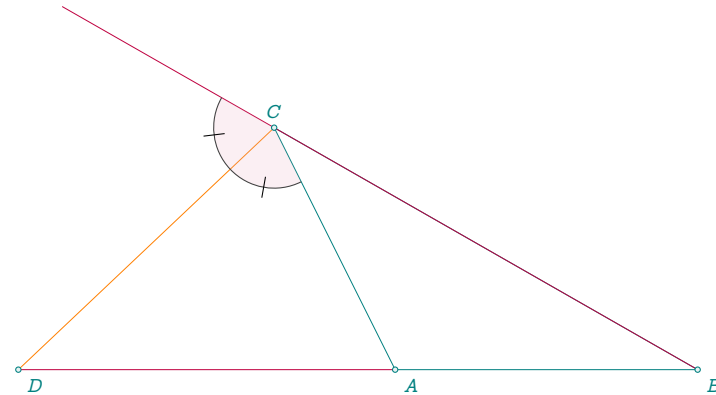
**2.5 Angle Bisector Theorem: External angle bisector****Theorem 4: External angle bisector**

The external bisector of an angle $(\widehat{ACB})$ of a triangle (ABC) divides the opposite side externally in the ratio of the corresponding sides containing the angle.

If the exterior bisector intersects the line (AB) at point D then $\frac{DA}{DB} = \frac{CA}{CB}$.

File external.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
z.C = point(-2, 4)
T.ABC = triangle(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)
```



2.6 Menelaus' theorem (weak version)

Theorem 5: Menelaus (weak version)

If line (PR) intersecting (AB) on triangle ABC , where R is on the extension of $[AB]$, P is on (AC) , and Q on the intersection of (PR) and (BC) , then

$$\frac{RA}{RB} \cdot \frac{QB}{QC} \cdot \frac{PC}{PA} = 1.$$

Weak version as segment lengths are used and not signed lengths.

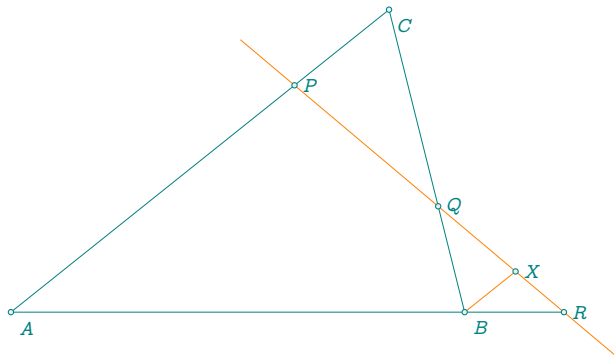
Proof. With similar triangles: draw a line parallel to (AC) from B ...

Remark It's possible to find the result by calculation because

$$\frac{RA}{RB} = \text{length}(z.R, z.A) / \text{length}(z.R, z.B) \text{ etc.}$$

File `menelaus.lua`:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(5, 4)
T.ABC = triangle(z.A, z.B, z.C)
z.P = T.ABC.ca:point(0.25)
z.Q = T.ABC.bc:point(0.35)
L.PQ = line(z.P, z.Q)
z.R = intersection(L.PQ, T.ABC.ab)
z.X = intersection(T.ABC.ca:ll_from(z.B), L.PQ)
```



The converse of Menelaus' theorem is indeed true but we must be precise about the positioning of the points if we consider only the lengths of the segments. Suppose three points P, Q, R are on sides (or extension)

$[AB], [BC], [AC]$ respectively, such that one or three of P, Q, R are in the extensions of the sides. Then points P, Q, R are collinear if and only if

$$\frac{RA}{RB} \cdot \frac{QB}{QC} \cdot \frac{PC}{PA} = 1$$

2.7 Application: Pascal' theorem weak version

The statement and proof are from the site :

Theorem 6: Pascal' theorem

Given 6 points (which can be coincident) on the circumference of a circle labelled A, C, E, B, F and D in that order around the circle, the intersections P, Q and R of (AB) and (DE) , (AF) and (CD) , and (BC) and (EF) are collinear.

Proof. This demonstration is based on a general case. In some cases, points may be confused and some lines may be parallel, so for these particular cases, the proof must be adapted.

Let Q be the intersection of (CD) and (AF) , let P be the intersection of (AB) and (DF) , and let R be the intersection of (BC) and (EF) . We will prove these three points are collinear.

Let U be the intersection of (CD) and (EF) , let V be the intersection of (AB) and (EF) , and let W be the intersection of (AB) and (CD) . By Menelaus in the triangle UVW and the transversal (CD) , we have $\frac{VQ}{UQ} \cdot \frac{UC}{WC} \cdot \frac{WD}{VD} = 1$.

Then with the transversal (AB) , we have $\frac{VA}{UA} \cdot \frac{UB}{WB} \cdot \frac{WP}{VP} = 1$.

Finally with the transversal (EF) , we get $\frac{VF}{UF} \cdot \frac{UR}{WR} \cdot \frac{WE}{VE} = 1$.

Multiply the three identities and rearrange the terms:

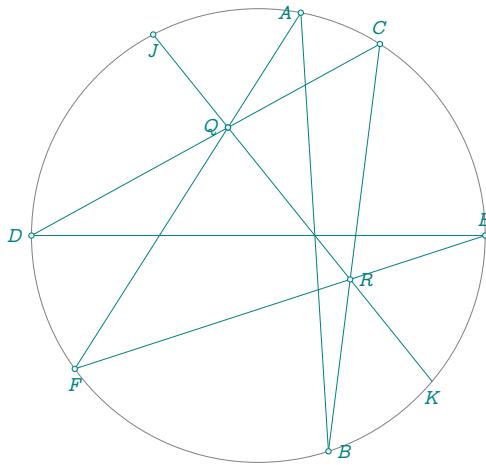
$$\frac{VQ}{UQ} \cdot \frac{UR}{WR} \cdot \frac{WP}{VP} = 1.$$

File `pascal.lua`:

```
init_elements()
z.O = point(0, 0)
```

Proof. Let's call Ω the circle passing through A, B, C, D, E and F .
 Projecting through F onto Ω .
 $(J, K; Q, R) = (J, K; A, E)$
 Projecting through C onto Ω .
 $(J, K; Q, R) = (J, K; D, B)$
 Projecting through B onto (J, K) .
 $(J, K; A, E) = (J, K; (AB) \cap (JK), (EB) \cap (JK))$
 Projecting through E onto (J, K) .
 $(J, K; D, B) = (J, K; (DE) \cap (JK), (BE) \cap (JK))$
 $(EB) \cap (JK) = (BE) \cap (JK)$ and $(AB) \cap (JK) = (DE) \cap (JK) = P$
 Finally P, Q, R are collinear.

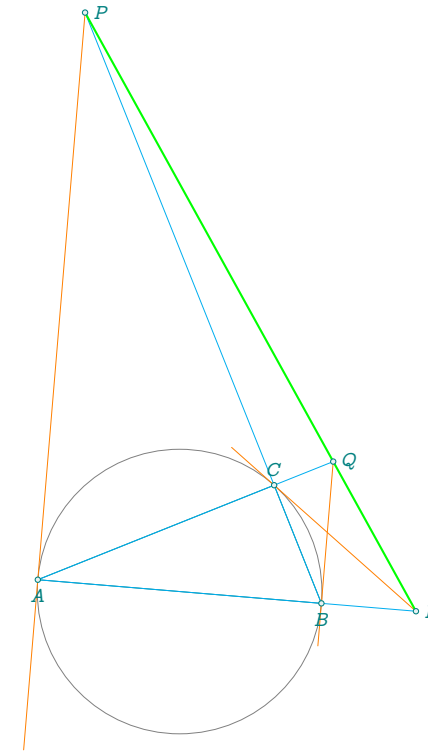
```
init_elements()
z.O = point(O, O)
z.E = point(3, O)
C.OE = circle(z.O, z.E)
z.C = C.OE:point(O.16)
z.A = C.OE:point(O.22)
z.D = C.OE:point(O.5)
z.F = C.OE:point(O.6)
z.B = C.OE:point(O.8)
L.AB = line(z.A, z.B)
L.DE = line(z.D, z.E)
L.AF = line(z.A, z.F)
L.CD = line(z.C, z.D)
L.BC = line(z.B, z.C)
L.EF = line(z.E, z.F)
z.P = intersection(L.AB, L.DE)
z.Q = intersection(L.AF, L.CD)
z.R = intersection(L.BC, L.EF)
L.QR = line(z.Q, z.R)
z.J, z.K = intersection(C.OE, L.QR)
```



2.8 Pascal's theorem with a triangle

File pascal_tr.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(3, -0.25)
z.C = point(2.5, 1)
T.ABC = triangle(z.A, z.B, z.C)
z.O = T.ABC.circumcenter
C.OA = circle(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(T.ABC.bc, L.Ta)
z.Q = intersection(T.ABC.ca, L.Tb)
z.R = intersection(T.ABC.ab, L.Tc)
```



2.9 Application of Pascal's theorem

Lemma 1: Polar and Pole

If through a point X there be drawn any straight line to meet the circle in A and B , the locus of the point of intersection of tangents at A and B is called the polar of X ; also X is called the pole of the polar.

Proof. The triangles OXU and OMN are similar:

$$XO \times MO = UO \times NO$$

The triangles OUA and OAN are similar:

$$UO \times NO = AO \times AO$$

$$\text{Therefore, } XO \times MO = AO^2$$

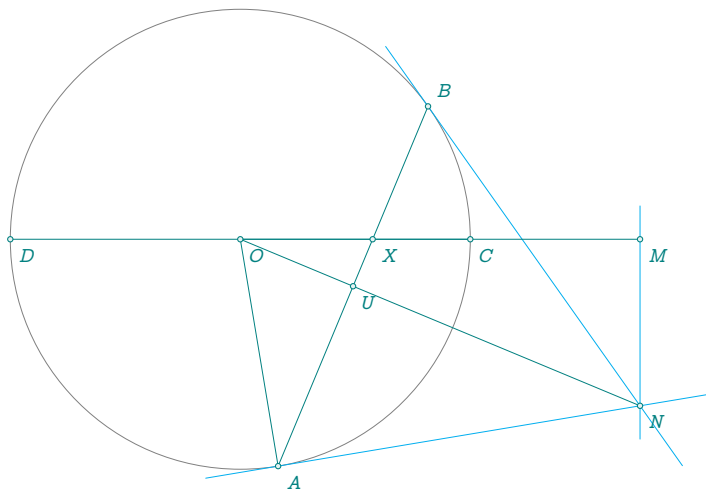


File pole.lua:

```

init_elements()
z.O = point(0, 0)
z.A = point(0.5, -3)
C.OA = circle(z.O, z.A)
z.X = point(1.75, 0)
L.OX = line(z.O, z.X)
L.RS = line(L.OX.north_pb, L.OX.south_pb)
z.R, z.S = intersection(C.OA, L.RS)
L.AX = line(z.A, z.X)
L.TA = C.OA:tangent_at(z.A)
z.Ax, z.Ay = L.TA:get()
z.B = intersection(L.AX, C.OA)
L.TB = C.OA:tangent_at(z.B)
z.Bx, z.By = L.TB:get()
L.TR = C.OA:tangent_at(z.R)
L.TS = C.OA:tangent_at(z.S)
z.Rx, z.Ry = L.TR:get()
z.Sx, z.Sy = L.TS:get()
z.M = intersection(L.TR, L.TS)
z.N = intersection(L.TA, L.TB)
z.U = tkz.midpoint(z.A, z.B)
z.C, z.D = intersection(C.OA, L.OX)

```



$$\frac{CX}{DX} = \frac{CM}{DM}$$

Exercise 1: Polar and Pascal

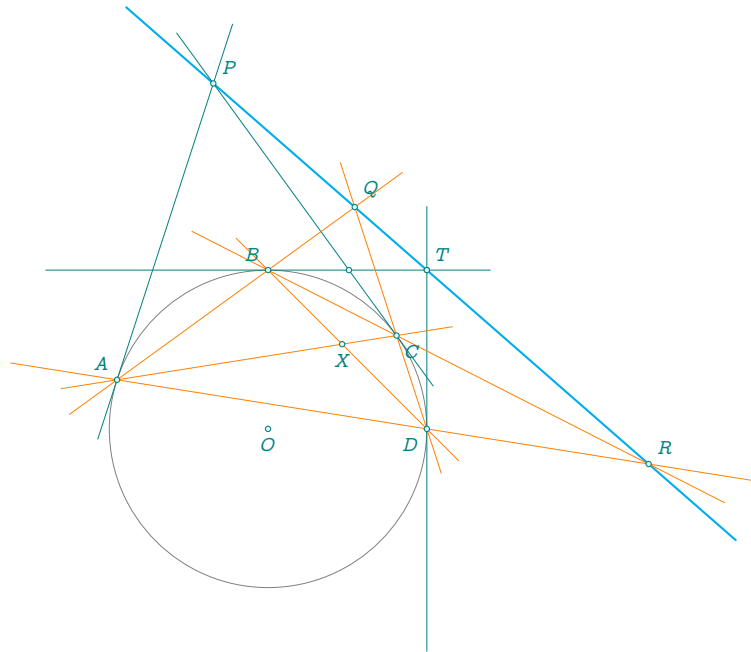
Given a cyclic quadrilateral $ABCD$ if the intersection of (AC) and (BD) is P , the intersection of (AB) and (CD) is Q and the intersection of (AD) and (BC) is R , prove the polar of P with respect of $C(O, A)$ passes through Q and R .

File polar.lua:

```

init_elements()
z.O = point(0, 0)
z.D = point(3, 0)
C.OD = circle(z.O, z.D)
z.B = C.OD:point(0.25)
z.A = C.OD:point(0.45)
z.C = C.OD:point(0.10)
L.AC = line(z.A, z.C)
L.AB = line(z.A, z.B)
L.CD = line(z.C, z.D)
L.AD = line(z.A, z.D)
L.BD = line(z.B, z.D)
L.BC = line(z.B, z.C)
z.X = intersection(L.AC, L.BD)
z.Q = intersection(L.AB, L.CD)
z.R = intersection(L.AD, L.BC)
L.QR = line(z.Q, z.R)
L.Ta = C.OD:tangent_at(z.A)
L.Tb = C.OD:tangent_at(z.B)
L.Tc = C.OD:tangent_at(z.C)
L.Td = C.OD:tangent_at(z.D)
z.Ax, z.Ay = L.Ta:get()
z.Bx, z.By = L.Tb:get()
z.Cx, z.Cy = L.Tc:get()
z.Dx, z.Dy = L.Td:get()
z.Ibd = intersection(L.Tb, L.Td)
z.P = intersection(L.Ta, L.Tc)
z.T = intersection(L.Tb, L.Td)
z.Ibc = intersection(L.Tb, L.Tc)

```



Proof. By the Pascal's Theorem on $AACBBD$, we have P, Q, R collinear. Then with hexagon $ABBCCD$, we have Q, R, T collinear. Finally, P, Q, R, T are collinear.

T_a and T_c are tangent at the circle at A and C , the intersection $X(AC \cap BD)$ lies on the polar of P , i.e. (AC) (Refer to previous Lemma). Similarly, X lies on the polar of T , which leads to P and T belonging at the polar of X , so (PT) is the polar of X . Since P, Q, R and T are collinear, then Q and R are also on the polar of X . RQ is the polar of X . ■

2.10 Construction of a harmonic division

Proof. Given three distinct collinear points A, C , and B , with C not the midpoint of AB , the harmonic conjugate D of C with respect to A and B may be constructed as follows.

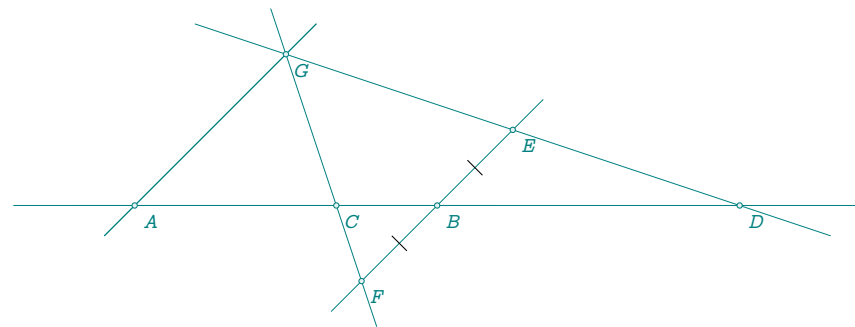
Choose a point G not on line AB . The line through B parallel to GA meets GC at a point F . Let E be the reflection of F across B ; then the line GF intersects AB at a point D , which is the desired harmonic conjugate. Indeed, the configuration produces a harmonic bundle $(G, A; B, F)$, whose image under

the symmetry with center B yields $(G, A; B, E)$; projecting from G gives the harmonic quadruple $(A, B; C, D)$.

If GE were parallel to AB , the quadrilateral $AGBE$ would be a parallelogram, forcing C to be the midpoint of AB , contrary to assumption. Thus the construction ensures both existence and uniqueness. When C is* the midpoint of AB , the line GE becomes parallel to AB , and the harmonic conjugate reduces to the point at infinity on line AB , in agreement with projective geometry. ■

File harmonic_div1.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(4, 0)
z.G = point(2, 2)
L.AG = line(z.A, z.G)
L.AB = line(z.A, z.B)
z.E = L.AG:collinear_at(z.B, 0.5)
L.GE = line(z.G, z.E)
z.D = intersection(L.GE, L.AB)
z.F = z.B:symmetry(z.E)
L.GF = line(z.G, z.F)
z.C = intersection(L.GF, L.AB)
```



2.11 Harmonic division

Definition 1: Harmonic division

Let A, C, B , and D four points which lie in this order on a line d . The four-point $(A, B; C, D) = \frac{CA}{CB} \div \frac{DA}{DB}$ is called a harmonic division if $(A, B; C, D) = 1$.
 C and D are harmonic conjugates with respect to A and B .
 If X is a point not lying on d , then we say that pencil which consists of the four lines $(XA), (XB), (XC), (XD)$ is harmonic if $(A, B; C, D)$ is harmonic.

Theorem 7

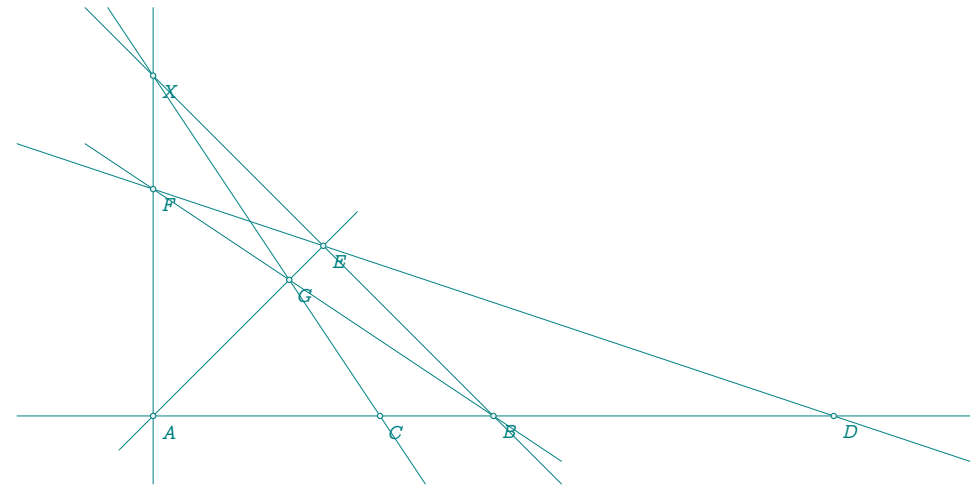
How to get the harmonic conjugate C of a point D outside the segment $[AB]$.

Proof. We choose a point X not aligned with A and B . Here I choose X such that (AX) is orthogonal to (AB) and with $AB = AX$. Then I denote by E the middle of $[XB]$.

F is the intersection of (DE) with (XA) , then G is the intersection of (AE) with (BF) and finally C is the intersection of lines (XG) and (AB) ■

File harmonic.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.D = point(12, 0)
L.AB = line(z.A, z.B)
z.X = L.AB.north_pa
L.XB = line(z.X, z.B)
z.E = L.XB.mid
L.ED = line(z.E, z.D)
L.AX = line(z.A, z.X)
L.AE = line(z.A, z.E)
z.F = intersection(L.ED, L.AX)
L.BF = line(z.B, z.F)
z.G = intersection(L.AE, L.BF)
L.GX = line(z.G, z.X)
z.C = intersection(L.GX, L.AB)
```



2.12 Harmonic division with golden ratio

Example 3: Harmonic division with golden ratio

Two line segments of lengths a and b are said to be in the golden ratio if

$$\frac{a+b}{a} = \frac{a}{b} = \varphi.$$

Its principal representations are:

$$\varphi \approx 1.618033988749894...$$

$$\varphi = \frac{1 + \sqrt{5}}{2}$$

$$\varphi = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \ddots}}}$$

The golden ratio was known to Euclid as the extreme and mean ratio, and was later referred to as the divine proportion by Luca Pacioli.

File: gold.lua:

```

init_elements()
z.a = point(0, 0)
z.b = point(5, 0)
L.ab = line(z.a, z.b)
z.m, z.n = L.ab:harmonic_both (tkz.phi)

```



2.13 Gold division

Here is a construction that allows segment AB to be divided into a golden ratio.

Proof. By invariance under homothety, we may assume without loss of generality that

$$A = (0, 0), \quad B = (1, 0),$$

so that $AB = 1$.

The circles with centres A and B passing respectively through B and A intersect at two points E and F . We choose the point E lying below the line (AB) . Its coordinates are

$$E = \left(\frac{1}{2}, -\frac{\sqrt{3}}{2} \right).$$

Let J be the midpoint of $[AB]$, hence

$$J = \left(\frac{1}{2}, 0 \right).$$

Let K be the midpoint of $[JB]$, so that

$$K = \left(\frac{3}{4}, 0 \right).$$

The perpendicular bisector of $[JB]$ is the vertical line of equation $x = \frac{3}{4}$. It intersects the circle with centre B and radius 1 at a point G lying above (AB) , with coordinates

$$G = \left(\frac{3}{4}, \frac{\sqrt{15}}{4} \right).$$

The line (EG) has slope

$$m = \frac{\frac{\sqrt{15}}{4} - \left(-\frac{\sqrt{3}}{2} \right)}{\frac{3}{4} - \frac{1}{2}} = \sqrt{15} + 2\sqrt{3}.$$

Its equation is therefore

$$y + \frac{\sqrt{3}}{2} = (\sqrt{15} + 2\sqrt{3}) \left(x - \frac{1}{2} \right).$$

Let C be the intersection of (EG) with the line (AB) , whose equation is $y = 0$. Substituting $y = 0$ yields

$$\frac{\sqrt{3}}{2} = (\sqrt{15} + 2\sqrt{3}) \left(x - \frac{1}{2} \right),$$

and hence

$$x - \frac{1}{2} = \frac{\frac{\sqrt{3}}{2}}{\sqrt{15} + 2\sqrt{3}} = \frac{\sqrt{5} - 2}{2}.$$

Thus

$$x = \frac{\sqrt{5} - 1}{2},$$

and consequently

$$C = \left(\frac{\sqrt{5} - 1}{2}, 0 \right).$$

Since $AB = 1$, it follows that

$$\frac{AC}{AB} = AC = \frac{\sqrt{5} - 1}{2}.$$

Now the golden ratio is $\varphi = \frac{1 + \sqrt{5}}{2}$, and we have

$$\frac{1}{\varphi} = \varphi - 1 = \frac{\sqrt{5} - 1}{2}.$$

Therefore,

$$\boxed{\frac{AC}{AB} = \frac{1}{\varphi}}.$$

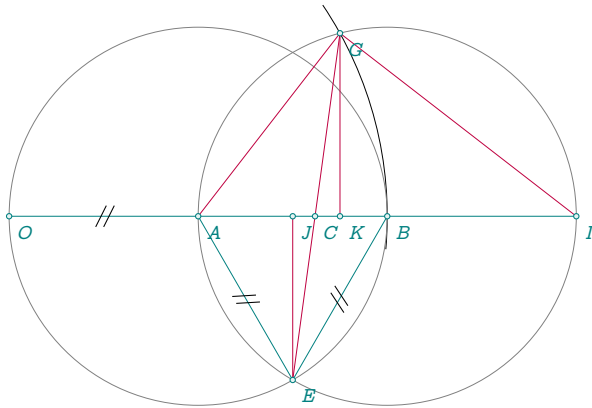
Hence the point C divides the segment $[AB]$ according to the golden ratio. ■

File: gold_division.lua

```

init_elements()
z.A = point(0, 0)
z.B = point(2.5, 0)
L.AB = line(z.A, z.B)
C.AB = circle(z.A, z.B)
C.BA = circle(z.B, z.A)
z.J = L.AB:midpoint()
L.JB = line(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(z.E, z.G)
z.C = intersection(L.EG, L.AB)
z.O = C.AB:antipode(z.B)

```



2.14 Gold ratio with segment

File gold_ratio.lua:

```

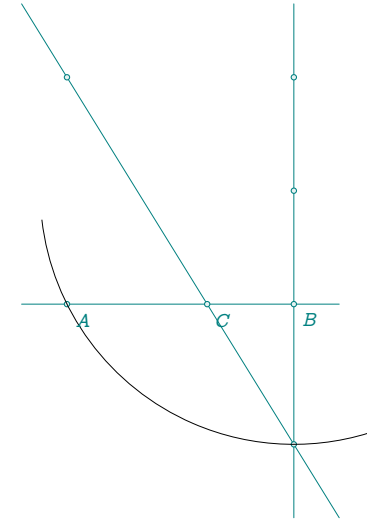
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
L.AB = line(z.A, z.B)
_, _, z.X, z.Y = L.AB:square():get()
L.BX = line(z.B, z.X)

```

```

z.M = L.BX.mid
C.MA = circle(z.M, z.A)
_, z.K = intersection(L.BX, C.MA)
L.AK = line(z.Y, z.K)
z.C = intersection(L.AK, L.AB)

```



2.15 Bisector and harmonic division

Theorem 8: Bisector and harmonic division

Let four points A, C, B and D , in this order, lying on the straight line (d) and M un point pris hors de (d) . Then, if two of the following three propositions are true, then the third is also true:

1. The division $(A,B;C,D)$ is harmonic. ($CA/CB = DA/DB$)
2. (MC) is the internal angle bisector of $\widehat{AMB}$.
3. $(MD) \perp (MC)$.

File bisector.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(6, 0)

```

We conclude that $(A, B; C, D)$ is harmonic.

Theorem 9: Alternate segment

The Alternate Segment theorem states: An angle that lies between a tangent and a chord is equal to the angle subtended by the same chord in the alternate segment.

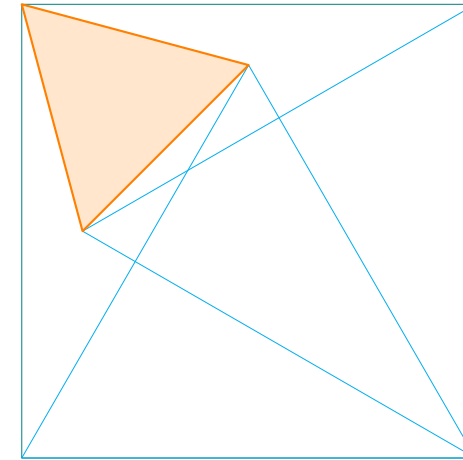
Proof. AOB is an isosceles triangle. $[OH]$ is the perpendicular bisector of $[AB]$ and finally $\widehat{AOB} = 2\widehat{ACB}$ then ... ■

Converse: If a line is drawn at one end of a chord of a circle in such a way that angles made with the chord are equal to alternate angles made by the chord in the segment then this line is the tangent line to the circle.

2.17 Thébault's problem II**Theorem 10: Thébault's problem II**

Given a square, construct equilateral triangles on two adjacent edges, either both inside or both outside the square. Then the triangle formed by joining the vertex of the square distant from both triangles and the vertices of the triangles distant from the square is equilateral. [wikipedia]

```
init_elements()
z.A = point(0, 0)
z.B = point(8, 0)
L.AB = line(z.A, z.B)
S.ABCD = L.AB:square()
z.C = S.ABCD.pc
z.D = S.ABCD.pd
z.E = S.ABCD.ab:equilateral().pc
z.F = S.ABCD.bc:equilateral().pc
```

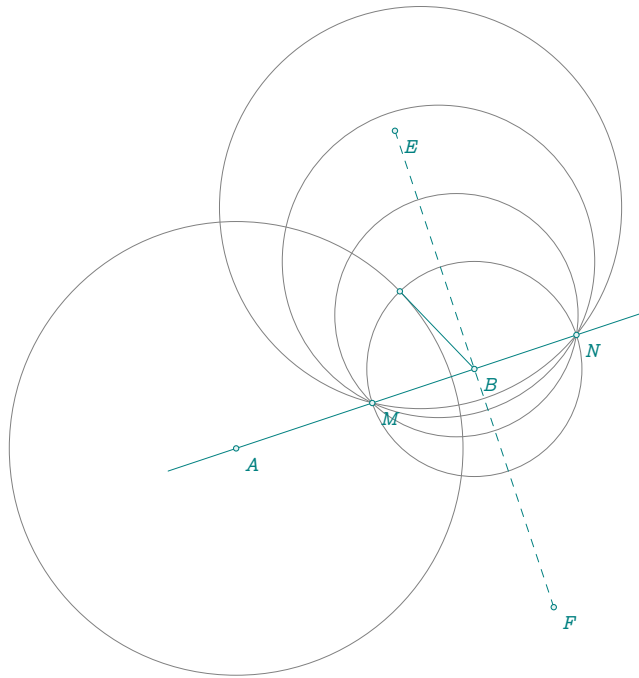
**2.18 Apollonius circle****Theorem 11: Apollonius circle**

Apollonius of Perga proposed the following problem: "Find the locus of a point the ratio of whose from two fixed points is constant". Let $k \neq 1$ be the ratio such that $\frac{MA}{MB} = k$.

Proof. Consider the point C on the segment $[AB]$ satisfying the ratio $\frac{CA}{CB} = \frac{MA}{MB} = k$. From the "angle bisector theorem" $\widehat{AMC} = \widehat{CMB}$. Next take the point D on the line (AB) that satisfies the ratio. By the "angle bisector theorem" the line (MD) bisects the exterior angle $\widehat{EMB}$. It's easy to show that (CM) and (DM) are orthogonal in M . So M is on the circle with diameter $[CD]$. ■

File apollonius.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.M = point(5, 3)
T.MAB = triangle(z.M, z.A, z.B)
L.bis = T.MAB:bisector()
z.C = L.bis.pb
```

$\frac{NB}{NA} = \frac{MB}{MA}$ The division is harmonic: $(A, B; M, N) = -1$
 All circles whose center lies on the radical axis (EF) and passing through N are orthogonal to the circle $C(A)$.
 see the pencils

2.21 Circle orthogonal to two given circles

Theorem 12: Circle orthogonal to two given circles

If a circle is orthogonal to two given circles, its center lies on the radical axis of those two circles.

File ortho_two.lua:

```

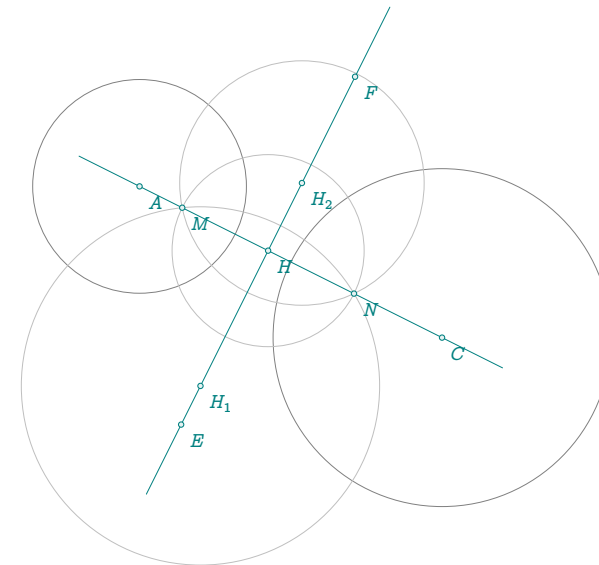
init_elements()
z.A = point(0, 1)
z.B = point(1, 0)
z.C = point(4, -1)

```

```

L.AC = line(z.A, z.C)
z.D = point(3, 1)
C.AB = circle(z.A, z.B)
C.CD = circle(z.C, z.D)
L.EF = C.AB:radical_axis(C.CD)
z.E, z.F = L.EF:get()
z.H = C.AB:radical_center(C.CD)
C.ortho = C.AB:orthogonal_from(z.H)
z.T = C.ortho.through
z.M, z.N = intersection(C.ortho, L.AC)
z.H_1 = L.EF:report(-2, z.H)
z.H_2 = L.EF:report(1, z.H)

```



2.22 Orthogonal Circles -- Secant Criterion

For two circles to be orthogonal, it is necessary and sufficient for a secant passing through one of their common points to be seen from the other common point at a right angle.

Theorem 13: Orthogonal Circles – Secant Criterion

Let two circles intersect at points P and Q . Then the circles are orthogonal if and only if any secant through P is seen from Q at a right angle; that is, for every point X on any secant through P , one has

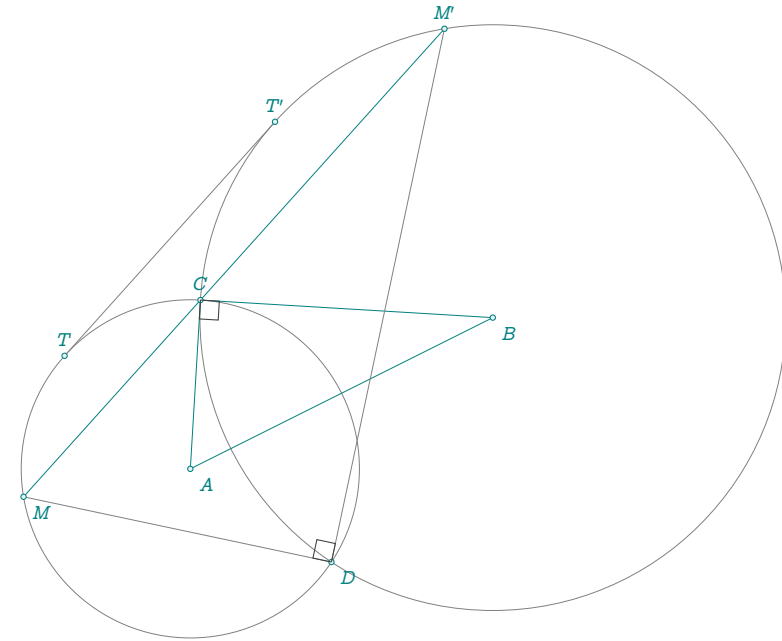
$$\angle XQP = 90^\circ.$$

File `ortho_circles.lua`

```

init_elements()
z.A = point(0, 0)
z.B = point(4, 2)
L.AB = line(z.A, z.B)
z.a = point(1, 2)
C.Aa = circle(z.A, z.a)
C.BC = C.Aa:orthogonal_from(z.B)
z.C, z.D = intersection(C.Aa, C.BC)
C.AC = circle(z.A, z.C)
L.TTp = C.AC:common_tangent(C.BC)
z.T, z.Tp = L.TTp:get()
z.M = C.AC:point(0.45)
L.MC = line(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)

```

**2.23 Radical center and radical circle**

In the implementation of the package, the method `radical_center(C1, C2)` also has a well-defined meaning when applied to only two circles. One of the circles can be reduced to a point.

The point returned by

$$\mathbf{z.P} = \mathbf{C1:radical_center(C2)}$$

is the intersection of the *radical axis* of the two circles with the line joining their centres.

This point may be regarded as an *extended radical center*. It is the centre of a distinguished circle orthogonal to both given circles. This circle belongs to the coaxial pencil of circles orthogonal to **C1** and **C2**; its intersections with the line joining the centres coincide with the two fixed base points of this pencil.

The construction naturally extends to the case of a circle and a point. A point P is interpreted as a *circle-point*, that is, a circle of zero radius:

$$\mathbf{z.P} = \mathbf{point(a,b)}, \quad \mathbf{C.P} = \mathbf{circle(z.P,z.P)}.$$

In this setting, the radical center is the centre of the unique circle orthogonal to the given circle and passing through the point P .

- Case of three circles. The most general and common case:

Definition 3: Circle orthogonal to three circles

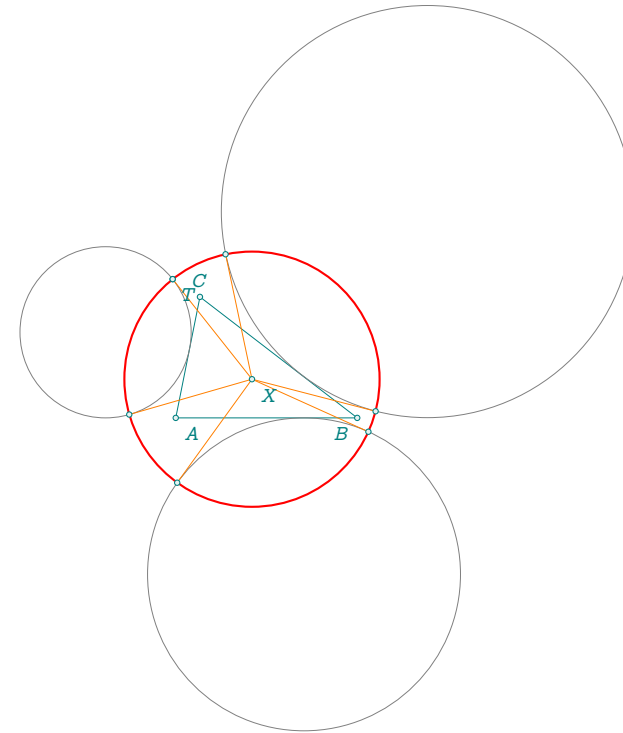
The radical axes of three circles, taken in pairs, generally coincide at the same point, which has the same power in relation to all three circles and is called the radical center of the three circles.

```
C.ortho = radical_circle (C.exa,C.exb,C.exc)
```

The center is intersection of the three radical axis.

File radical_three.lua:

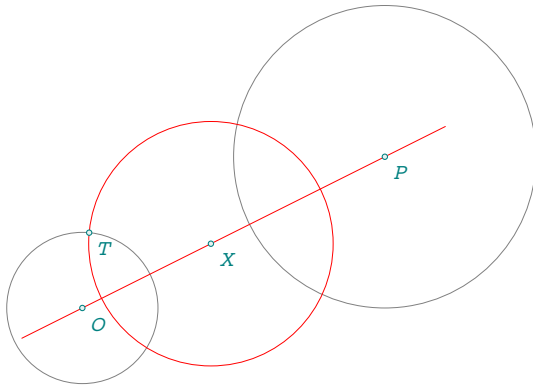
```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(0.8, 4)
T.ABC = triangle(z.A, z.B, z.C)
C.exa = T.ABC:ex_circle()
z.I_a, z.Xa = C.exa:get()
C.exb = T.ABC:ex_circle(1)
z.I_b, z.Xb = C.exb:get()
C.exc = T.ABC:ex_circle(2)
z.I_c, z.Xc = C.exc:get()
C.ortho = C.exa:radical_circle(C.exb, C.exc)
z.X, z.T = C.ortho:get()
z.Ta, z.Tap = intersection(C.ortho, C.exa)
z.Tb, z.Tbp = intersection(C.ortho, C.exb)
z.Tc, z.Tcp = intersection(C.ortho, C.exc)
z.E1, z.F1 = C.exa:radical_axis(C.exb):get()
z.E2, z.F2 = C.exb:radical_axis(C.exc):get()
z.E3, z.F3 = C.exc:radical_axis(C.exa):get()
```



- Case of two circles. The functions below can be used to obtain the center (located on the center axis and the radical axis) of a circle orthogonal to the two given circles.

File radical_two.lua:

```
init_elements()
z.O = point(0, 0)
z.x = point(1, 0)
z.z = point(4, 0)
z.P = point(4, 2)
C.Ox = circle(z.O, z.x)
C.Pz = circle(z.P, z.z)
z.X = C.Ox:radical_center(C.Pz)
C.X = C.Ox:radical_circle(C.Pz)
z.w, z.T = C.X:get()
```



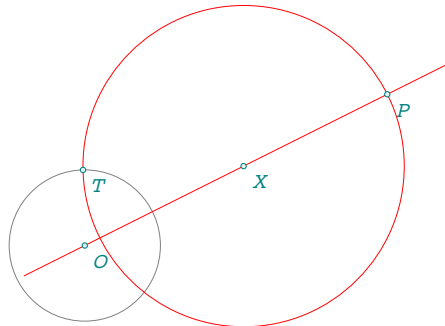
– Case of one circle and one point

The functions used for the next case are valid. We use a circle_point

`C.P = circle : new (z.P,z.P).`

File `radical_one.lua`:

```
init_elements()
z.O = point(0, 0)
z.x = point(1, 0)
z.P = point(4, 2)
C.Ox = circle(z.O, z.x)
C.P = circle(z.P, z.P)
z.X = C.Ox:radical_center(C.P)
C.X = C.Ox:radical_circle(C.P)
z.w, z.T = C.X:get()
```



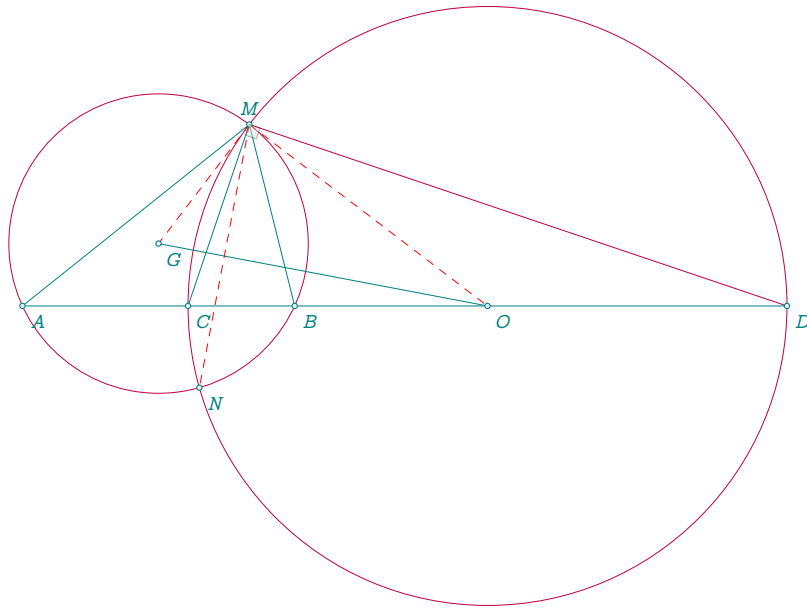
2.24 Apollonius circles and circumcircle of a triangle

Theorem 14: Apollonius circles and circumcircle

The apollonius circles of triangle are orthogonal to the circumcircle of this triangle.

File `apollonius_circum.lua`:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.M = point(5, 4)
T.AMB = triangle(z.A, z.M, z.B)
L.AB = T.AMB.ca
z.I = T.AMB.incenter
L.MI = line(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(z.C, z.D)
z.O = L.CD.mid
z.G = T.AMB.circumcenter
C.GA = circle(z.G, z.A)
C.OC = circle(z.O, z.C)
_, z.N = intersection(C.GA, C.OC)
```



Proof. $\widehat{OMB} + \widehat{BMC} = \widehat{OMC} = \widehat{OCM} = \widehat{CAM} + \widehat{AMC}$
 or $\widehat{AMC} = \widehat{BMC}$ so $\widehat{OMB} = \widehat{CAM} = \widehat{BAM}$

Hence by the alternate segment theorem the result follows. (OM) is tangent at M to the circumcircle of the triangle ABC . ■

Proof. Another solution is to prove that $OM^2 = OB \cdot OA$ i.e. A and B are inverses with respect to the circle with diameter $[CD]$.

From $\frac{CA}{CB} = \frac{DA}{DB}$ and $OM = R$

$CB = R - OB$, $DB = R + OB$, $AD = R + OA$ and $AC = OA - R$

then $(OA - R)(OB + R) = (R - OB)(OA + R)$.

Finally after simplification

$R^2 = OM^2 = OA \cdot OB$ ■

2.25 Apollonius circles of a triangle

Definition 4: Apollonius circles in a triangle

Let ABC be a triangle. For each side, say BC , and any positive real number $k \neq 1$, the Apollonius circle corresponding to the ratio k is the locus of points X in the plane such that

$$\frac{XB}{XC} = k.$$

When $k = \frac{AB}{AC}$, the circle is said to be the Apollonius circle of side BC in triangle ABC .

Each triangle has three such circles:

$$A_A: \frac{XB}{XC} = \frac{AB}{AC}, \quad A_B: \frac{XC}{XA} = \frac{BC}{BA}, \quad A_C: \frac{XA}{XB} = \frac{CA}{CB}.$$

These are called the Apollonius circles of A , B , and C , respectively.

Main Properties.

1. Each Apollonius circle is orthogonal to the circumcircle of ABC .
2. The centers of the three Apollonius circles lie on the internal angle bisectors of the triangle.
3. The Apollonius circle of side BC passes through the vertex A if and only if the ratio is $k = \frac{AB}{AC}$, in which case it is the unique circle through A dividing the side BC internally in the same ratio:

$$\frac{AB}{AC} = \frac{XB}{XC}.$$

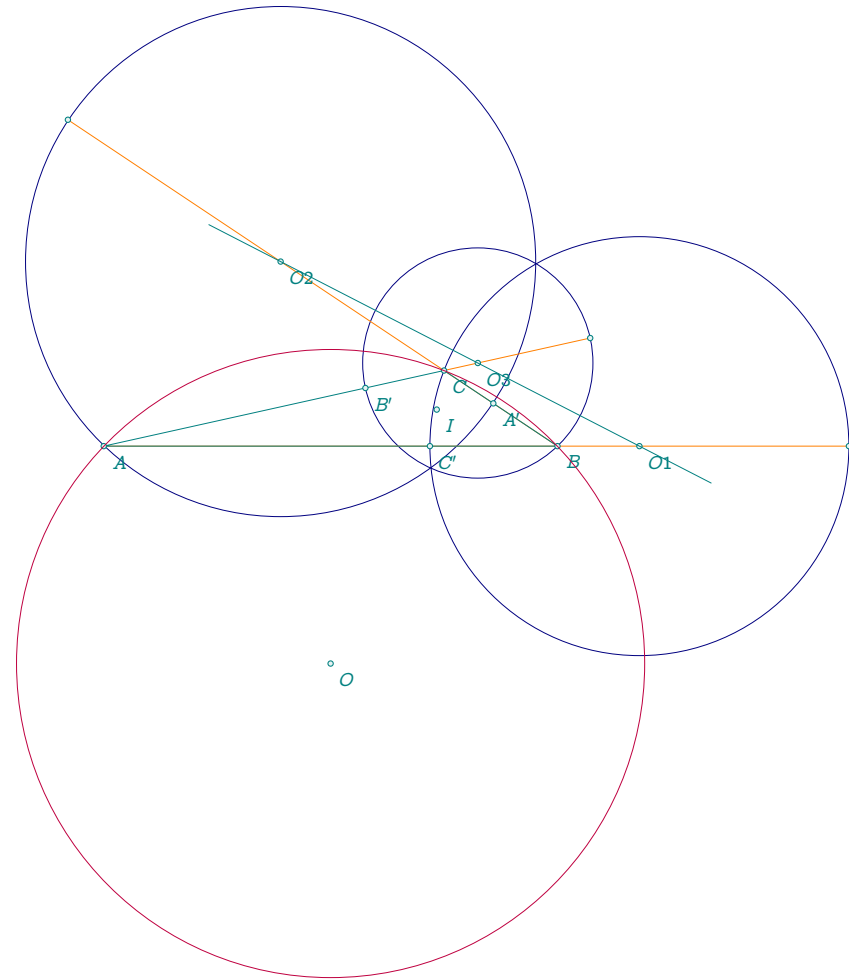
4. The three Apollonius circles intersect pairwise in two points; each pair determines one of the isodynamic points of the triangle.
5. The radical axes of each Apollonius circle with the circumcircle are the internal angle bisectors of the triangle.
6. Under inversion in any Apollonius circle, the triangle is mapped to an equilateral triangle.

File `apollonius_triangle.lua`:

```

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

```



Theorem 15: Apollonius circles of a triangle: common points

The three Apollonius circles of a (non-equilateral) triangle meet at exactly two points.

Proof. Let ABC be a non-equilateral triangle, say with $AB \neq AC$. By definition, we have directly that if a point belongs to two of the circles, then it belongs to the third. Indeed, if

$$\frac{MB}{MC} = \frac{AB}{AC} \text{ and } \frac{MA}{MB} = \frac{CA}{CB} \text{ then } \frac{MA}{MC} = \frac{BA}{BC}$$

However, the circles are neither tangent nor disjoint if $AB \neq AC$. ■

Theorem 16: Apollonius circles are coaxial

The three Apollonius circles are coaxial. Their centers are aligned.

(O_1C) is a tangent line to the circumscribed circle of triangle ABC . We deduce that the triangles O_1BC and O_1AC are similar which leads to $\frac{O_1B}{O_1C} = \frac{BC}{AC} = \frac{a}{b} = \frac{O_1C}{O_1A}$ with $AB = c$, $BC = a$, $AC = b$. Thus $\frac{O_1B^2}{O_1C^2} = \frac{a^2}{b^2}$. But from power of the point O_1 we have $O_1C^2 = O_1A \cdot O_1B$ (This equality can also be obtained with relations in similar triangles).

This leads to $\frac{O_1B}{O_1A} = \frac{a^2}{b^2}$. Also $\frac{O_1A}{O_1C} = \frac{c^2}{a^2}$ and $\frac{O_1C}{O_1B} = \frac{b^2}{c^2}$.

Multiplying the three expressions, from the converse of the Menelaus's theorem we conclude that O_1, O_2 and O_3 are aligned and the three Apollonius circles are coaxial.

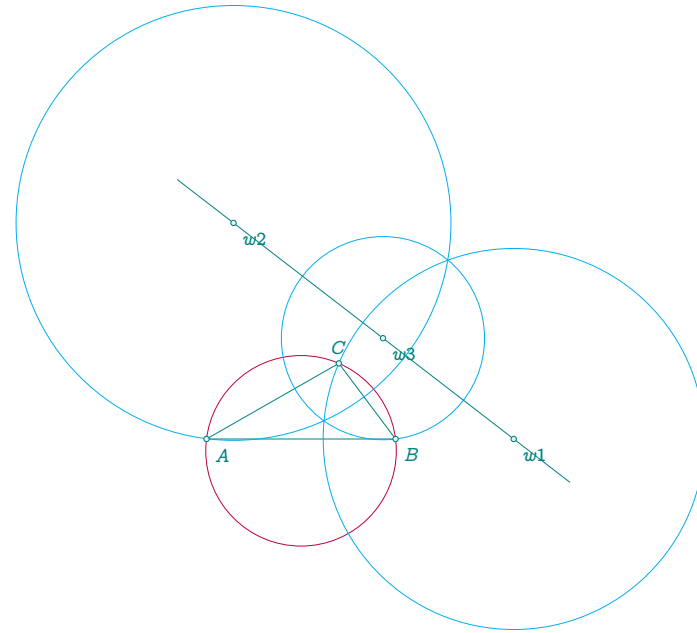
2.26 Apollonius circles in a triangle with method

Example 4: Apollonius circles with a method

Same result than the previous example using the function
T.ABC.ab : apollonius (k)

File apollonius_method.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
z.C = point(3.5, 2)
T.ABC = triangle(z.A, z.B, z.C)
z.O = T.ABC.circumcenter
C.AB = T.ABC.ab:apollonius(T.ABC.b / T.ABC.a)
z.w1, z.t1 = C.AB:get()
C.BC = T.ABC.bc:apollonius(T.ABC.c / T.ABC.b)
z.w2, z.t2 = C.BC:get()
C.AC = T.ABC.ca:apollonius(T.ABC.a / T.ABC.c)
z.w3, z.t3 = C.AC:get()
```



2.27 Power of a point with respect to a circle

Definition 5: Power of a point with respect to a circle

When a variable secant from a fixed point M intersects a given circle C at A and B , the product $\overline{MA} \times \overline{MB}$ is a constant number called the power of point M with respect to this circle.

These points are of course merged if the line is tangent to the circle. The number k is positive if the point M is outside the circle, null if it belongs to it and negative if it belongs to the open disk defined by the circle. We can easily verify that $k = d^2 - R^2$ where d is the distance from point M to the center of the circle and R is the radius of the circle. k est aussi noté $\Gamma(M)$. We therefore have $\Gamma(M) = \overline{MA} \times \overline{MB} = d^2 - R^2$.

File power.lua:

```
init_elements()
z.O = point(0, 0)
z.A = point(2, -2)
z.M = point(-6, 0)
```


2.29 Radical axis

Definition 6: Radical axis

The geometric set of points that have the same power with respect to two given circles is a line perpendicular to the line of the centers, called radical axis of the two circles. If the circles have two points in common, the radical axis is the common secant line of the circles.

For the point M to have the same power with respect to the two circles $\Gamma = O(R)$ and $\Gamma' = O'(R')$, it is necessary and sufficient that :

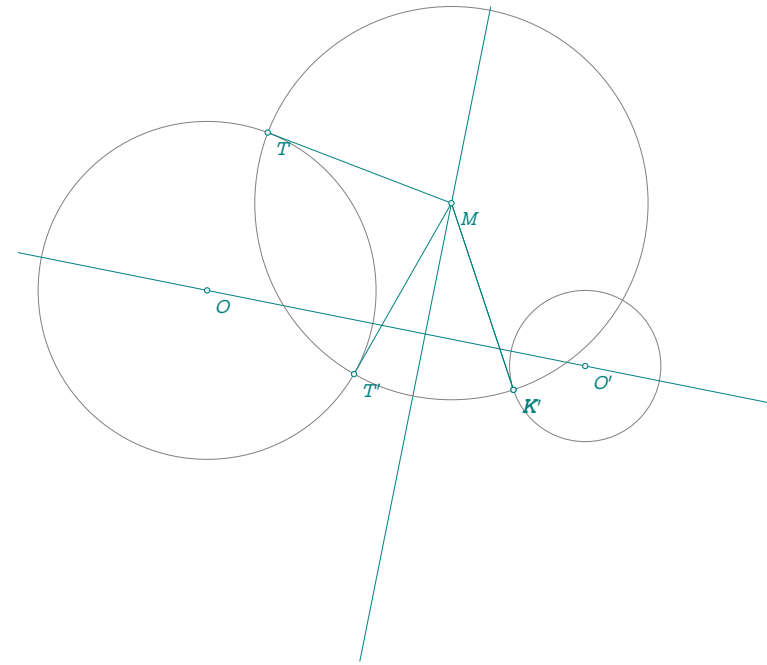
$$MO^2 - R^2 = MO'^2 - R'^2 \text{ either } MO^2 - MO'^2 = R^2 - R'^2$$

Denoting by I the middle of OO' , the set of points M is therefore a line OO' perpendicular to the line OO' at the point H defined by the relation:

$$2\overline{OO'} \cdot \overline{IH} = R^2 - R'^2$$

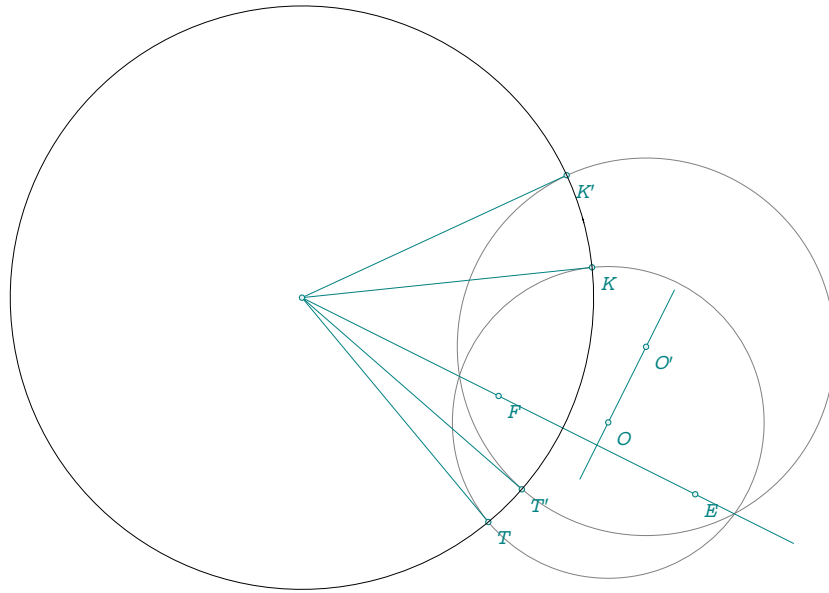
File radical_axis.lua:

```
init_elements()
z.O = point(-1, 0)
z.Op = point(4, -1)
z.B = point(0, 2)
z.D = point(4, 0)
C.OB = circle(z.O, z.B)
C.OpD = circle(z.Op, z.D)
L.EF = C.OB:radical_axis(C.OpD)
z.E, z.F = L.EF:get()
z.M = L.EF:point(0.2)
L.MT, L.MTp = C.OB:tangent_from(z.M)
_, z.T = L.MT:get()
_, z.Tp = L.MTp:get()
L.MK, L.MKp = C.OpD:tangent_from(z.M)
_, z.K = L.MK:get()
_, z.Kp = L.MKp:get()
```



File radical_axis2.lua:

```
init_elements()
z.O = point(-1, 0)
z.Op = point(0, 2)
z.C = point(3, -1)
z.D = point(3, -2)
C.OC = circle(z.O, z.C)
C.OpD = circle(z.Op, z.D)
z.E, z.F = C.OC:radical_axis(C.OpD):get()
L.EF = line(z.E, z.F)
z.M = L.EF:point(2)
L.MK, L.MT = C.OC:tangent_from(z.M)
z.K, z.T = L.MK.pb, L.MT.pb
L.MKp, L.MTp = C.OpD:tangent_from(z.M)
z.Kp, z.Tp = L.MKp.pb, L.MTp.pb
```

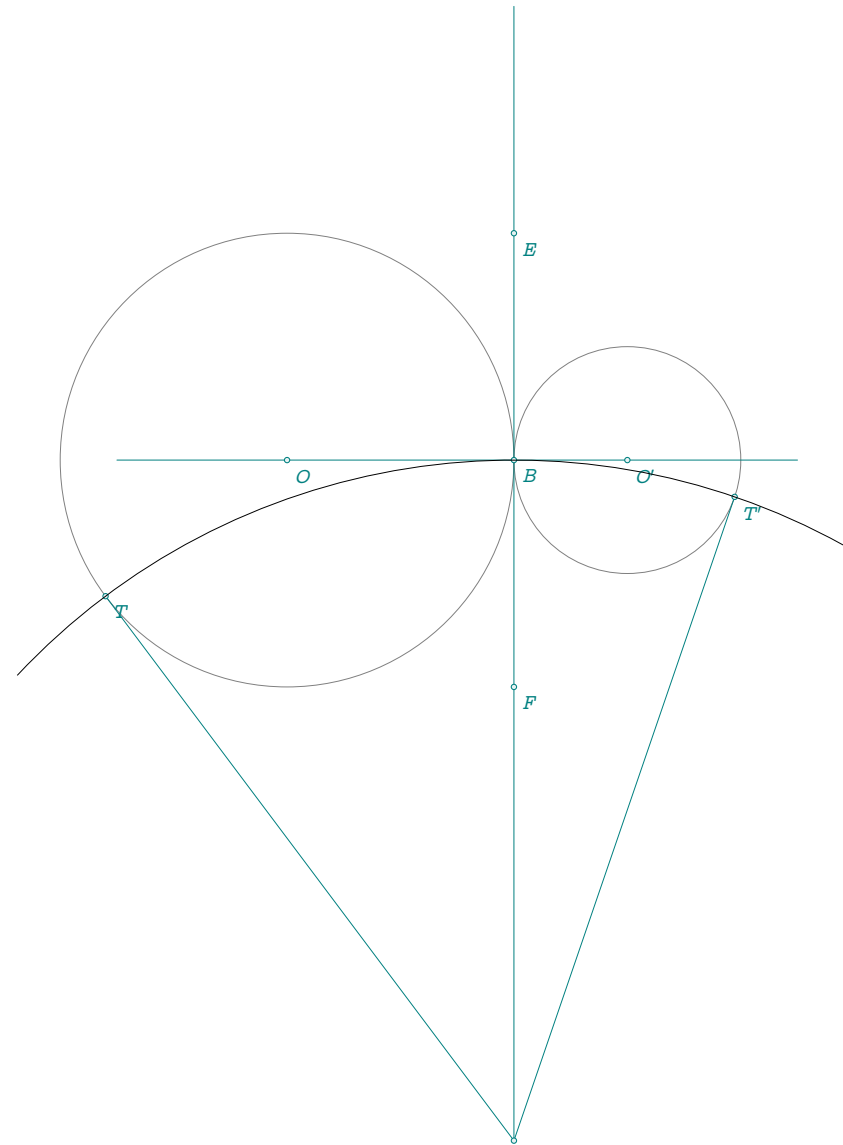


File radical_axis3.lua:

```

init_elements()
z.O = point(0, 0)
z.B = point(4, 0)
z.Op = point(6, 0)
C.OB = circle(z.O, z.B)
C.OpB = circle(z.Op, z.B)
L.EF = C.OB:radical_axis(C.OpB)
z.E, z.F = L.EF:get()
z.M = L.EF:point(2)
_, L.T = C.OB:tangent_from(z.M)
_, z.T = L.T:get()
L.T, _ = C.OpB:tangent_from(z.M)
_, z.Tp = L.T:get()

```



File radical_axis4.lua:

```

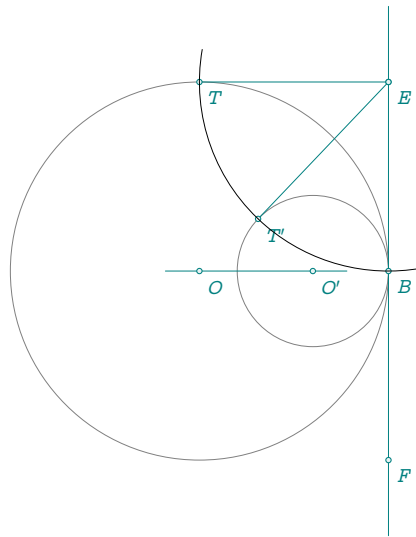
init_elements()
z.O = point(0, 0)

```

```

z.B = point(5, 0)
z.Op = point(3, 0)
C.OB = circle(z.O, z.B)
C.OpB = circle(z.Op, z.B)
L.EF = C.OB:radical_axis(C.OpB)
z.E, z.F = L.EF:get()
z.M = L.EF:point(0)
L.T, _ = C.OB:tangent_from(z.M)
_, z.T = L.T:get()
L.T, _ = C.OpB:tangent_from(z.M)
_, z.Tp = L.T:get()

```



File radical_axis5.lua:

```

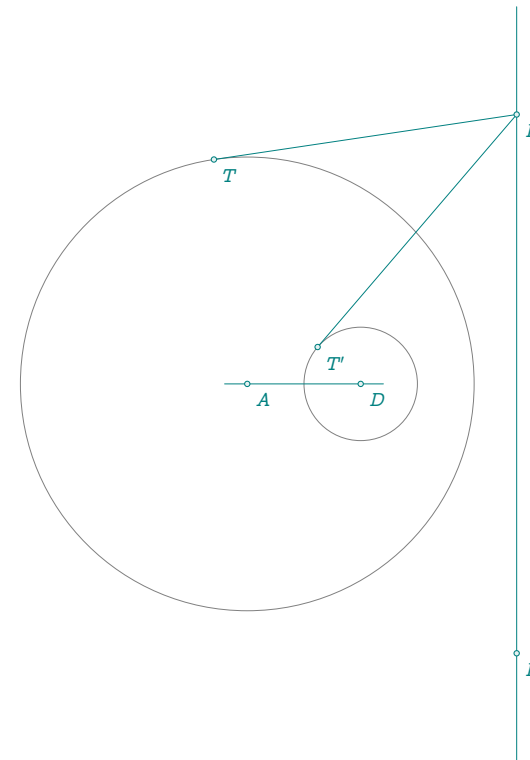
init_elements()
z.A = point(0, 0)
z.B = point(4, 0)
z.C = point(3, 0)
z.D = point(2, 0)
C.AB = circle(z.A, z.B)
C.DC = circle(z.D, z.C)
L.EF = C.AB:radical_axis(C.DC)
z.E, z.F = L.EF:get()
z.M = L.EF:point(0)

```

```

L.T, _ = C.AB:tangent_from(z.M)
_, z.T = L.T:get()
L.T, _ = C.DC:tangent_from(z.M)
_, z.Tp = L.T:get()

```



2.30 Construction of the radical axis for two non-intersecting circles

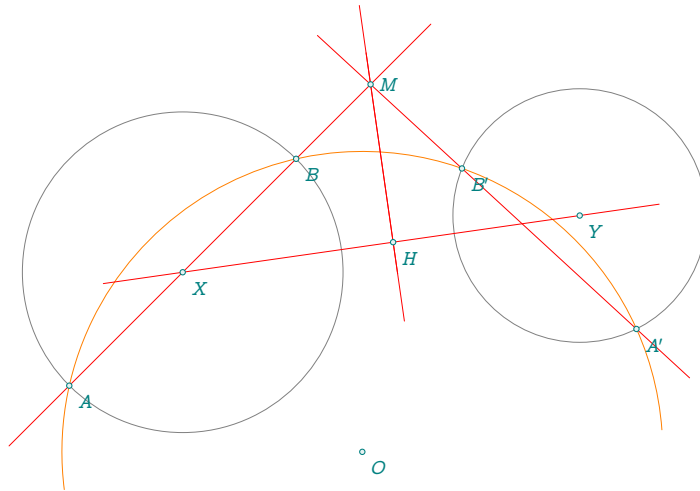
The last construction shows that in a general case the radical axes of three circles intersect at a single point. This construction allows us to construct the radical axis of two non-intersecting circles. Let us draw an auxiliary circle that intersects the first circle at A and B and the second at A' and B' . The lines (AB) and $(A'B')$ intersect at a point M which will belong to the radical axis. This one being perpendicular to the line of centers, it remains to construct the orthogonal projection of M on the line of centers.

File radical_build.lua:

```

init_elements()
z.X = point(0, 0)
z.B = point(2, 2)
z.Y = point(7, 1)
z.Ap = point(8, -1)
L.XY = line(z.X, z.Y)
C.XB = circle(z.X, z.B)
C.YAp = circle(z.Y, z.Ap)
z.E, z.F = C.XB:radical_axis(C.YAp):get()
z.A = C.XB:point(0.5)
T.ABAp = triangle(z.A, z.B, z.Ap)
z.O = T.ABAp.circumcenter
C.OAp = circle(z.O, z.Ap)
_, z.Bp = intersection(C.OAp, C.YAp)
L.AB = line(z.A, z.B)
L.ApBp = line(z.Ap, z.Bp)
z.M = intersection(L.AB, L.ApBp)
z.H = L.XY:projection(z.M)

```



The part of the radical axis, outside the two circles, is the place of the points from which equal tangents can be drawn.

Indeed, for $MT = MT'$ it is necessary and sufficient that M has the same positive power with respect to the circles O and O' . It follows that the radical axis of two circles passes through the middle of any line common to these two circles. When two circles are external, the middles of the 4 common

tangents are aligned on the radical axis.

2.31 Powers of a point with respect to two circles

Theorem 17: Powers of a point with respect to two circles

The difference of the powers of a point with respect to two circles is, in absolute value, equal to the double product of the distance of their centers by the distance of this point to their radical axis.

(Refer to 2.29)

2.32 Pencil of circles

Definition 7: Pencil of circles

A pencil of circles or coaxial system is any family of circles which have the same radical axis Δ .

Any two circles in the plane have a common radical axis, which is the line consisting of all the points that have the same power with respect to the two circles. This is the case, for example, for circles passing through two fixed points A and B or of circles orthogonal to two fixed circles. We will show that a pencil is defined when we know a circle of this pencil and the common radical axis Δ and consequently when we know two circles of the pencil. The centers of the circles of a pencil are aligned on a perpendicular to the radical axis Δ of the pencil called "line of centers of the pencil".

2.33 Fixed point pencil

The fixed points A and B are the "base points" of the pencil.

The circles of the beam are the circles passing through the base points A and B . The pencil has a smallest circle, having as diameter the segment $[AB]$.

File pencil1.lua:

```

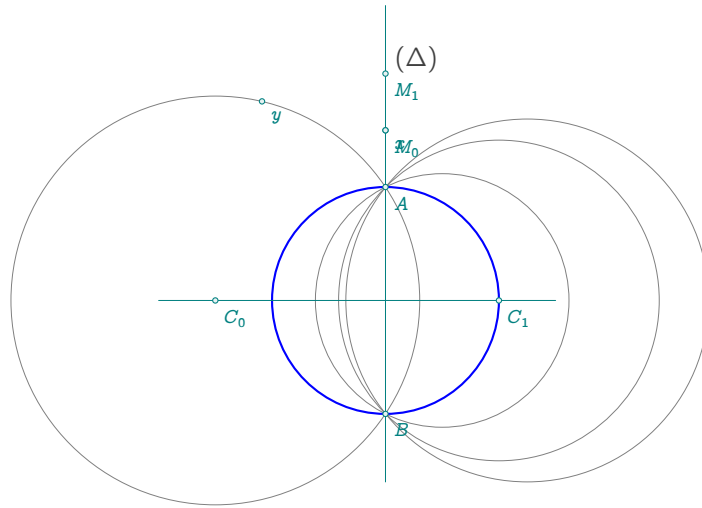
init_elements()
z.A = point(0, 2)
z.B = point(0, -2)
z.C_0 = point(-3, 0)
z.C_1 = point(2, 0)
z.C_3 = point(2.5, 0)

```

```

z.C_5 = point(1, 0)
L.BA = line(z.B, z.A)
z.M_0 = L.BA:point(1.25)
z.M_1 = L.BA:point(1.5)
C.CQA = circle(z.C_0, z.A)
z.x, z.y = C.CQA:orthogonal_from(z.M_0):get()
z.xp, z.yp = C.CQA:orthogonal_from(z.M_1):get()
z.O = L.BA.mid

```



File pencil2.lua:

```

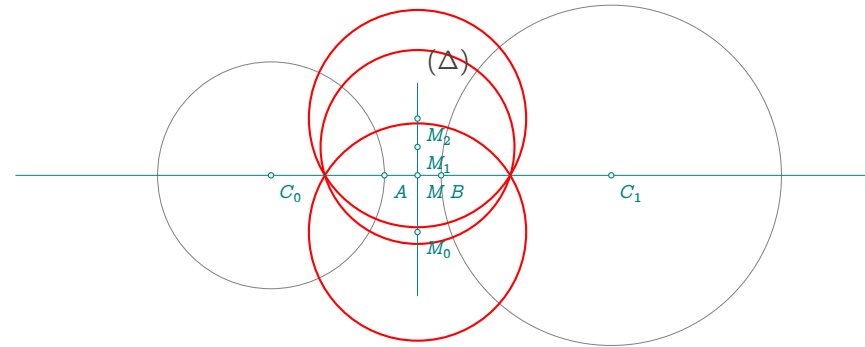
init_elements()
z.A = point(0, 0)
z.B = point(1, 0)
z.C_0 = point(-2, 0)
z.C_1 = point(4, 0)
C.CQA = circle(z.C_0, z.A)
C.C1B = circle(z.C_1, z.B)
z.F, z.E = C.CQA:radical_axis(C.C1B):get()
L.EF = line(z.E, z.F)
z.M = L.EF.mid
z.M_0 = L.EF:report(1, z.M)
z.M_1 = L.EF:report(-0.5, z.M)
z.M_2 = L.EF:report(-1, z.M)
z.oM0 = C.CQA:orthogonal_from(z.M_0)

```

```

z.x = C.oM0.through
C.oM1 = C.CQA:orthogonal_from(z.M_1)
z.xp = C.oM1.through
C.oM2 = C.CQA:orthogonal_from(z.M_2)
z.xpp = C.oM2.through

```



2.34 Inversion

Definition 8

nversion is the process of transforming points P to a corresponding set of points P' known as their inverse points. Two points P and P' are said to be inverses with respect to an inversion circle having inversion center O and inversion radius k if P' is the perpendicular foot of the altitude of, where Q is a point on the circle such that $OQ \perp PQ$.

If P and P' are inverse points, then the line L through P and perpendicular to OP is sometimes called a "polar" with respect to point P' , known as the "inversion pole". In addition, the curve to which a given curve is transformed under inversion is called its inverse curve (or more simply, its "inverse"). This sort of inversion was first systematically investigated by Jakob Steiner. [MathWorld]

Weisstein, Eric W. "Inversion." From MathWorld—A Wolfram Web Resource.

File def_inversion.lua:

```

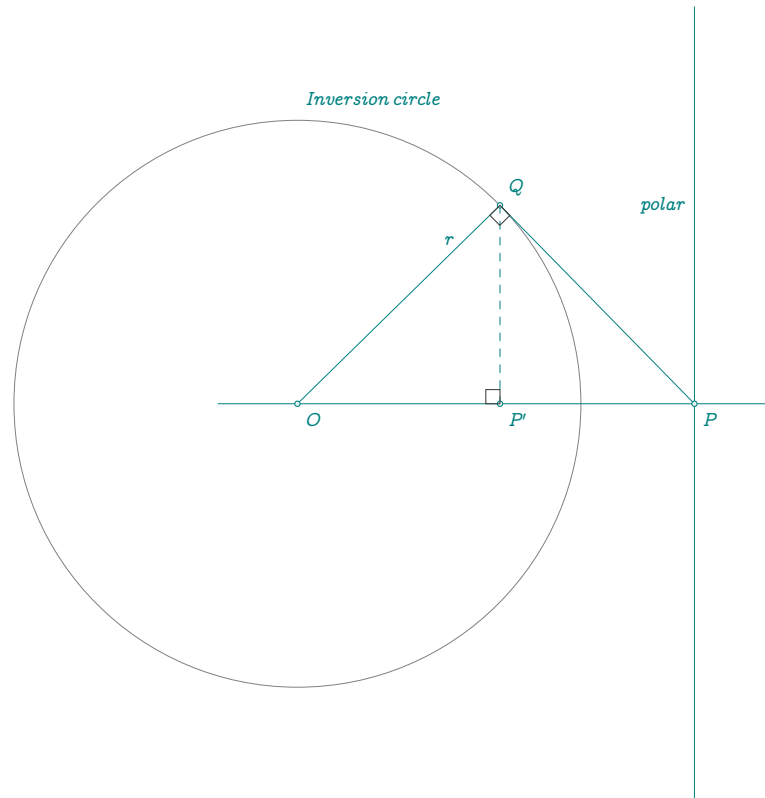
init_elements()
z.O = point(0, 0)
z.P = point(7, 0)
z.T = point(3, 4)

```

```

C.inv = circle(z.O, z.T)
z.Q = C.inv:tangent_from(z.P).pb
z.Pp = C.inv:inversion(z.P)
L.OP = line(z.O, z.P)
L.polar = L.OP:orthogonal_from(z.P)
z.u, z.v = L.polar:get()

```



File inversion.lua:

```

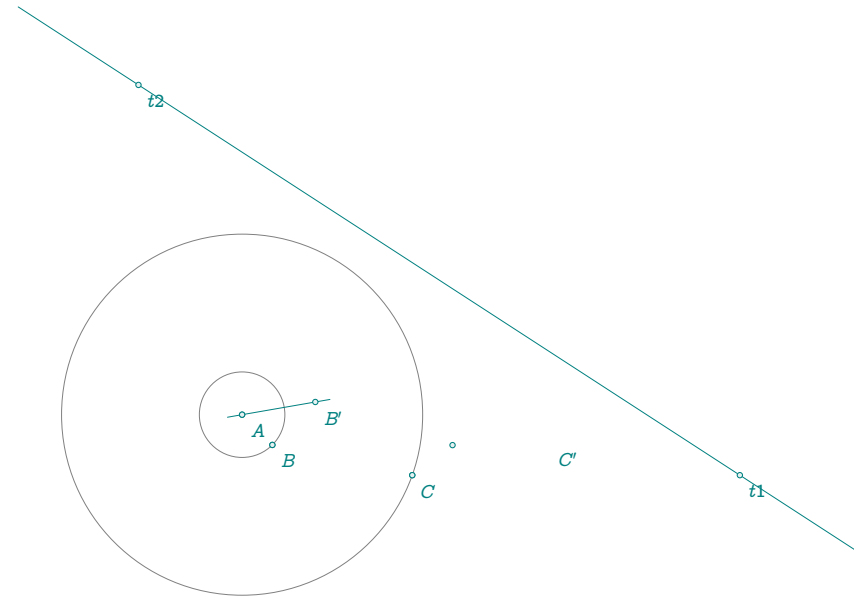
init_elements()
z.A = point(0, 2)
z.B = point(1, 1)
z.O = point(0, 0)
z.T = point(2.2, 0)
C.OT = circle(z.O, z.T)

```

```

z.Ap, z.Bp = C.OT:inversion(z.A, z.B)
T.ABAp = triangle(z.A, z.B, z.Ap)
z.I = T.ABAp.circumcenter

```



2.35 Cocyclic points

Theorem 18: Cocyclic points

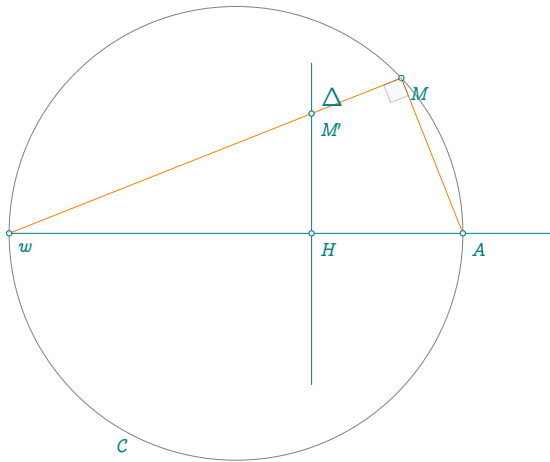
Let i be an inversion of pole w and power k , and C a circle containing w . Consider the point A of C diametrically opposite w and its inverse A' by i . Let B be any point of C distinct from A and from w , of inverse B' . The equality $\overline{wA} \cdot \overline{wA'} = \overline{wB} \cdot \overline{wB'}$ (each member being equal to k) shows that the points A, A', B', B are cocyclic.

Proof. To convince oneself of this, consider, for example, the circle γ circumscribed by the triangle AHM and the power of w with respect to this circle: $\overline{wA} \cdot \overline{wH} = \overline{wM} \cdot \overline{wN}$ where N is the other point common to the circle γ with the line (wM) . Then deduce that N is none other than M' . This being the case, since the angle $\widehat{AMM'}$ is right, so is the angle $\widehat{AHM'}$. Also, when M describes C , M' describes the line Δ perpendicular at H to the line (wA) . The image by i of the circle C (which contains the pole w) is the line Δ .

The triangles wAB and $wA'B'$ are similar or the lines (AB) and $(A'B')$ are antiparallel. ■

File cocyclic.lua:

```
init_elements()
z.w = point(0, 0)
z.A = point(6, 0)
z.H = point(4, 0)
L.wA = line(z.w, z.A)
z.I = L.wA.mid
C.IA = circle(z.I, z.A)
z.M = C.IA:point(0.12)
z.h = z.H:north()
L.Hh = line(z.H, z.h)
L.wM = line(z.w, z.M)
z.Mp = intersection(L.Hh, L.wM)
```



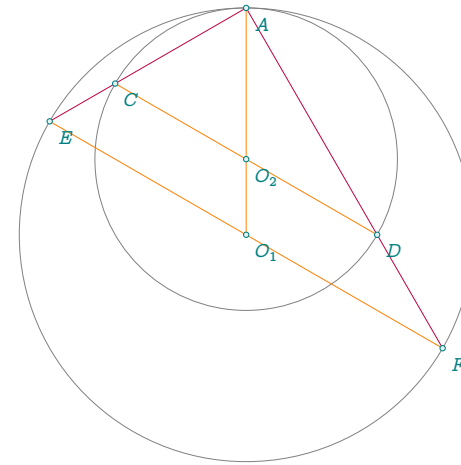
2.36 Archimedes' Book of Lemmas: Proposition 1

Theorem 19: Archimedes

If two circles touch at A, and if $[CD]$, $[EF]$ be parallel diameters in them, the points A, C, and E are aligned.

File archimedes.lua:

```
init_elements()
z.O_1 = point(0, 0)
z.O_2 = point(0, 1)
z.A = point(0, 3)
z.F = point:polar(3, -math.pi / 6)
L.FO1 = line(z.F, z.O_1)
C.O1A = circle(z.O_1, z.A)
z.E = intersection(L.FO1, C.O1A)
T.ABC = triangle(z.F, z.E, z.O_2)
z.x = T.ABC:parallelogram()
L.FO2 = line(z.x, z.O_2)
C.O2A = circle(z.O_2, z.A)
z.C, z.D = intersection(L.FO2, C.O2A)
```



Proof. $(CD) \parallel (EF)$

(AO_0) is secant to these two lines so $\widehat{AO_2C} = \widehat{AO_1E}$.

Since the triangles AO_1C and AO_2E are isosceles the angles at the base are equal $\widehat{AC}O_2 = \widehat{AE}O_1 = \widehat{CA}O_2 = \widehat{EA}O_1$. Thus A, C and E are aligned. ■

2.37 D'Alembert's Theorem

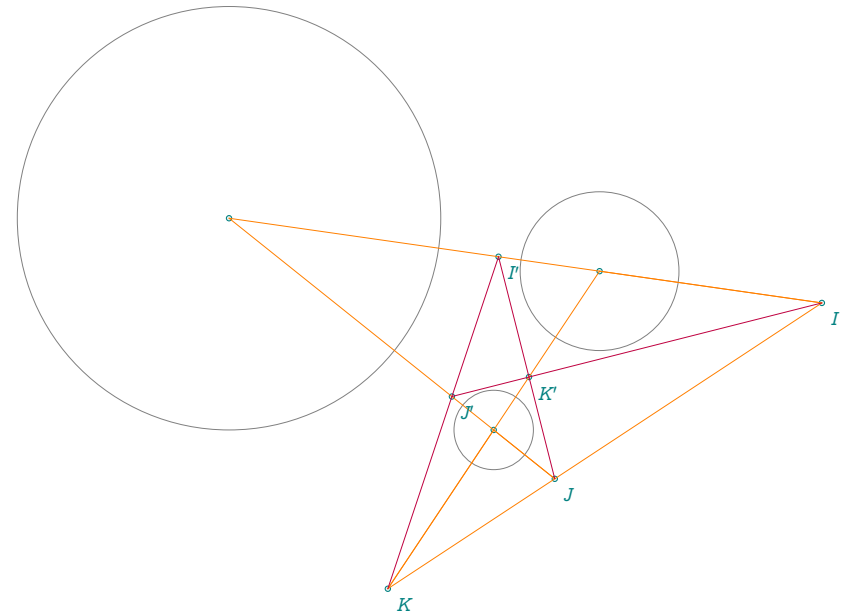
Theorem 20: D'Alembert-Monge

In geometry, Monge's theorem, named after Gaspard Monge, states that for any three circles in a plane, none of which is completely inside one of the others, the intersection points of each of the three pairs of external tangent lines are collinear. [Wikipedia].

If three circles A, B, and C are taken in pairs, the external similarity points of the three pairs lie on a straight line (I, J, K are aligned). Similarly, the external similarity point of one pair and the two internal similarity points of the other two pairs lie upon a straight line, forming a similarity axis: I', K', J or I', J', K or I, J', K').

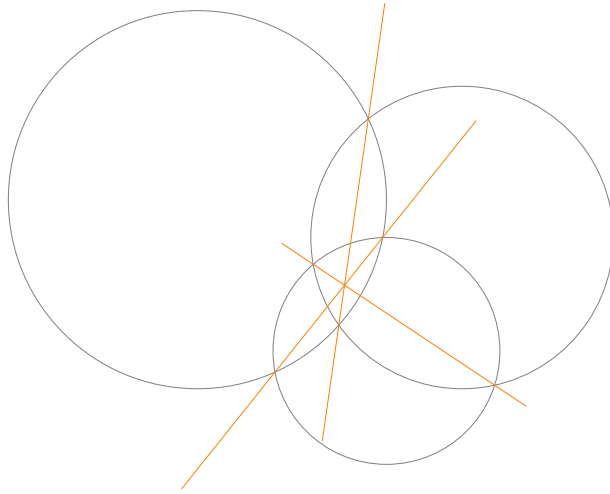
File alembert.lua:

```
init_elements()
z.A = point(0, 0)
z.a = point(4, 0)
z.B = point(7, -1)
z.b = point(5.5, -1)
z.C = point(5, -4)
z.c = point(4.25, -4)
C.Aa = circle(z.A, z.a)
C.Bb = circle(z.B, z.b)
C.Cc = circle(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)
```



File alembert2.lua:

```
init_elements()
z.A = point(0, 0)
z.a = point(5, 0)
z.B = point(7, -1)
z.b = point(3, -1)
z.C = point(5, -4)
z.c = point(2, -4)
C.Aa = circle(z.A, z.a)
C.Bb = circle(z.B, z.b)
C.Cc = circle(z.C, z.c)
z.i, z.j = C.Aa:radical_axis(C.Bb):get()
z.k, z.l = C.Aa:radical_axis(C.Cc):get()
z.m, z.n = C.Bb:radical_axis(C.Cc):get()
```



2.38 Altshiller-Court's theorem

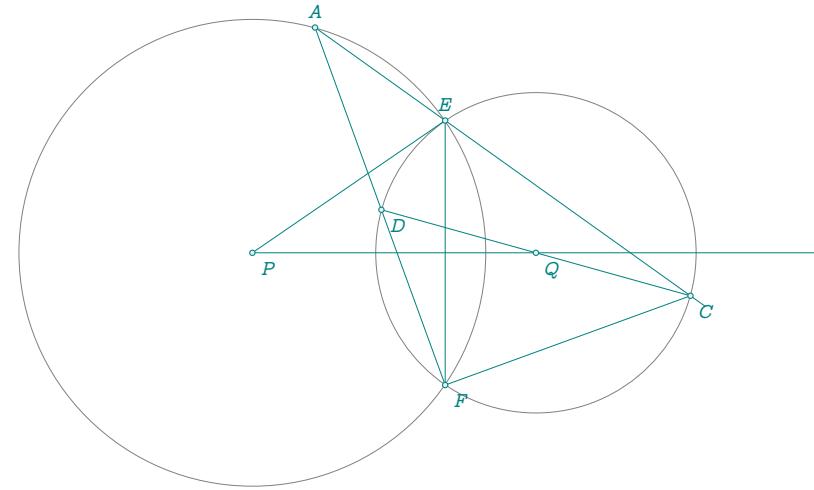
Theorem 21: Altshiller-Court's theorem_constant chord theorem

The two lines joining the points of intersection of two orthogonal circles to a point on one of the circles meet the other circle in two diametrically opposite points. Altshiller p 176. Nathan Altshiller Court described the constant chord theorem 1925 in the article sur deux cercles secants for the Belgian math journal Mathesis.[Wikipedia]

File altshiller.lua

```
init_elements()
z.P = point(0, 0)
z.Q = point(5, 0)
z.I = point(3, 2)
C.QI = circle(z.Q, z.I)
C.PE = C.QI:orthogonal_from(z.P)
z.E = C.PE.through
C.QE = circle(z.Q, z.E)
_, z.F = intersection(C.PE, C.QE)
z.A = C.PE:point(1 / 9)
L.AE = line(z.A, z.E)
_, z.C = intersection(L.AE, C.QE)
L.AF = line(z.A, z.F)
```

```
L.CQ = line(z.C, z.Q)
z.D = intersection(L.AF, L.CQ)
```



Proof. Let the lines (EA) , (FA) joining the points of intersection E, F of the two orthogonal circles (P) , (Q) to the point A of (P) meet (Q) again in C, D . We have:
 $\widehat{EAF} = \frac{1}{2}\widehat{EPF} = \widehat{EPQ}$ and $\widehat{ECF} = \frac{1}{2}\widehat{EQF} = \widehat{EQP}$
Hence $\widehat{EAF} + \widehat{ECF} = \widehat{EPQ} + \widehat{EQP} = 90^\circ$ and therefore the triangle CAF is right-angled at F . Thus CD subtends a right angle at F , which proves the proposition. ■

2.39 Reim's theorem 1

Theorem 22: Reim's theorem

Given two circles of center A and B intersecting at points C and D , and E and F any two points chosen on one of the circles, the lines (EC) and (FD) intersect the second circle at points H and G . We show that the lines (EF) and (HG) are parallel.

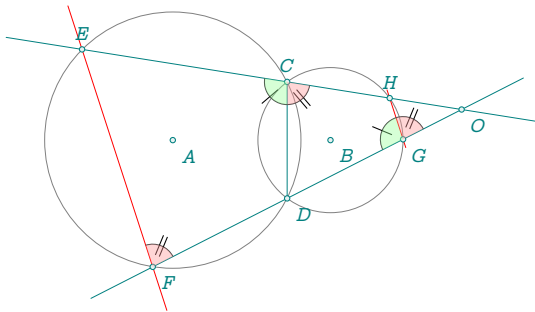
Converse - Suppose we have two circles ω_1, ω_2 that intersect at two points C, D . Let the line through C intersects the two circles at E, H (ω_1, ω_2 respectively). Then suppose a line that cuts ω_1 at F and ω_2 at G in a way that $EF \parallel HG$. Then that line passes through D .

Proof. (EF) and (CD) are antiparallel with respect to the sides of an angle O . (CD) and (GH) are antiparallel with respect to the sides of an angle O . (EF) and (GH) are parallel. (Furthermore, if (PQ) and (RS) are antiparallel, then the points P, Q, R , and S are concyclic)

Converse - Suppose that the line (FD) intersects the circle of center B at K . Through H there is only one parallel to (EF) so G, H, K are aligned. As H and K are two points of the circle they are coincident. ■

File reim.lua

```
init_elements()
z.A = point(0, 0)
z.E = point(-2, 2)
C.AE = circle(z.A, z.E)
z.C = C.AE:point(-0.3)
z.D = C.AE:point(0.55)
z.F = C.AE:point(0.35)
L.EC = line(z.E, z.C)
z.H = L.EC:point(1.5)
T.CDH = triangle(z.C, z.D, z.H)
z.B = T.CDH.circumcenter
C.BD = circle(z.B, z.D)
L.FD = line(z.F, z.D)
z.G = intersection(L.FD, C.BD)
z.O = intersection(L.EC, L.FD)
```

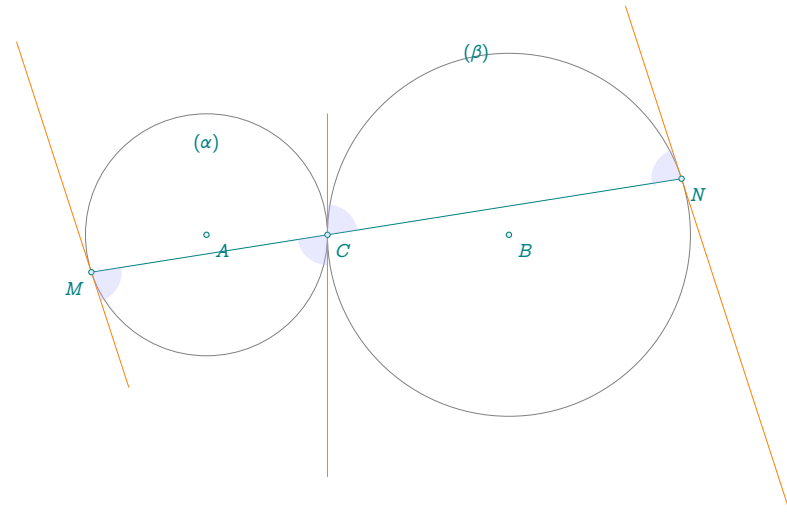


2.40 Reim's theorem 2

Let two circles (α) and (β) be tangent at C . Let a line passing through C intersect the circle (α) at M and intersect (β) at N . The tangents at M and N respectively to the circles (α) and (β) are parallel.

File reim2.lua

```
init_elements()
z.A = point(0, 0)
z.B = point(10, 0)
z.C = point(4, 0)
C.AC = circle(z.A, z.C)
z.c, z.cp = C.AC:tangent_at(z.C):get()
z.M = C.AC:point(0.55)
L.MC = line(z.M, z.C)
C.BC = circle(z.B, z.C)
z.N = intersection(L.MC, C.BC)
z.m, z.mp = C.AC:tangent_at(z.M):get()
z.n, z.np = C.BC:tangent_at(z.N):get()
```



Given a cyclic quadrilateral $ABQP$ and points P' and Q' on the extensions of $[PA]$ and $[FB]$, respectively. If $(PQ) \parallel (P'Q')$, then quadrilateral $ABQ'P'$ is cyclic.

(PQ) is parallel to $(P'Q')$ if and only if the points A, P', Q' and B are cocyclic.

It is sufficient to show that $\widehat{ABQ'} = \widehat{AP'Q'}$.

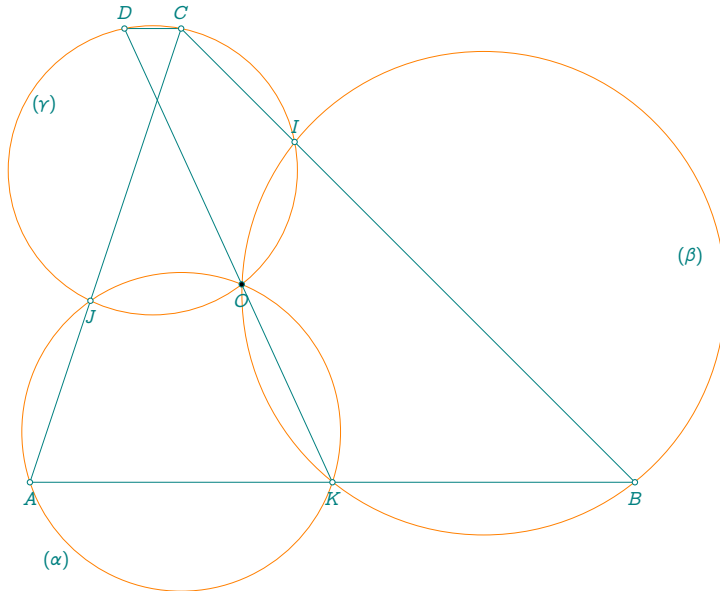
File reim2a.lua

```
init_elements()
z.A = point(0, 0)
z.B = point(8, 0)
```

```

z.C = point(2, 6)
L.AB = line(z.A, z.B)
L.AC = line(z.A, z.C)
L.BC = line(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(z.A, z.K, z.J)
T.BIK = triangle(z.B, z.I, z.K)
T.CIJ = triangle(z.C, z.I, z.J)
z.x = T.AKJ.circumcenter
z.y = T.BIK.circumcenter
z.z = T.CIJ.circumcenter
C.xK = circle(z.x, z.K)
C.yK = circle(z.y, z.K)
z.O, _ = intersection(C.xK, C.yK)
C.zO = circle(z.z, z.O)
L.KO = line(z.K, z.O)
z.D = intersection(L.KO, C.zO)

```



K, O the intersection points of (α) and (β) ,
 I one of the two points of intersection of (β) and (γ) ,

J one of the two points of intersection of (γ) and (α) ,

A a point of (α) ,

B the second point of intersection of (AK) with (β)

C the second point of intersection of (BI) with (γ) .

Let D be the second point of intersection of the line (KO) with (γ) Reim's theorem applied to the circles (α) and (γ) .

The circles (γ) and (β)

Given a cyclic quadrilateral $ABFE$ and points G and H on the extensions of (EA) and (FB) , respectively. If $GH \parallel EF$, then quadrilateral $ABHG$ is cyclic.

Reim's theorem $(CD) \parallel (AB)$

2.41 Three chords

Theorem 23: Three chords_Monge

Monge (Gaspard (1746 - 1818) Three circles intersect pairwise but let's assume there is no point shared by all three of them. There are three pairs of circles. Two circles in any pair share a chord. The problem is to prove that the three chords meet at a point.

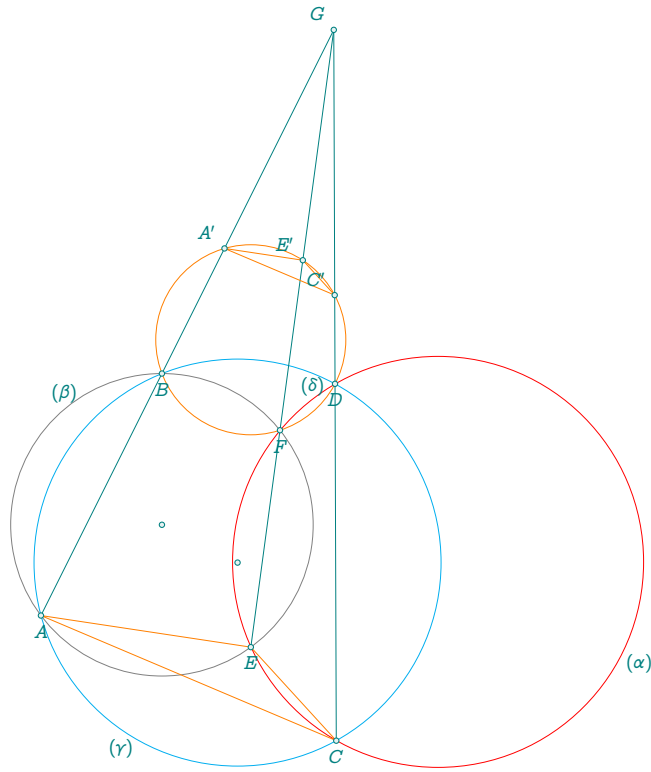
β, γ two intersecting circles A, B the points of intersection of β and γ , C, D two points of γ , E, F two points of β , and I the point of intersection of (AB) and (CD) .

Question : C, D, E, F are cocyclic if, and only if, (EF) passes through I . Let us note: (α) the circle passing through C, D, E, F . (δ) the circle passing through B, F, D . and A', C', E' the second points of intersection of $(AB), (CD), (EF)$ with (δ) .

Reim's theorem $(AC) \parallel (A'C') (CE) \parallel (C'E') (EA) \parallel (E'A')$

According to Desargues "The weak theorem" (Cf. Appendix) applied to homothetic triangles EAC and $E'A'C'$, (EE') passes through I . Conclusion: (EF) passes through I .

Monge's statement: when three circles intersect in pairs, the three intersecting lines are concurrent. Statement: If two circles are secant and the common line passes through the point of intersection of a secant of one and a secant of the other, then the four points of intersection are cocyclic.



File three_chords.lua

```

init_elements()
z.O = point(0, 0)
z.B = point(0, 2)
z.P = point(1, -0.5)
C.OB = circle(z.O, z.B)
C.PB = circle(z.P, z.B)
_, z.A = intersection(C.OB, C.PB)
z.D = C.PB:point(-0.14)
z.C = C.PB:point(-0.48)
z.E = C.OB:point(-0.40)
L.AB = line(z.A, z.B)
L.CD = line(z.C, z.D)
z.G = intersection(L.AB, L.CD)
L.GE = line(z.G, z.E)
z.F, _ = intersection(L.GE, C.OB)

```

```

T.CDE = triangle(z.C, z.D, z.E)
T.BFD = triangle(z.B, z.F, z.D)
z.w = T.CDE.circumcenter
z.x = T.BFD.circumcenter
L.GB = line(z.G, z.B)
L.GE = line(z.G, z.E)
L.GD = line(z.G, z.D)
C.xB = circle(z.x, z.B)
C.xF = circle(z.x, z.F)
C.xD = circle(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)

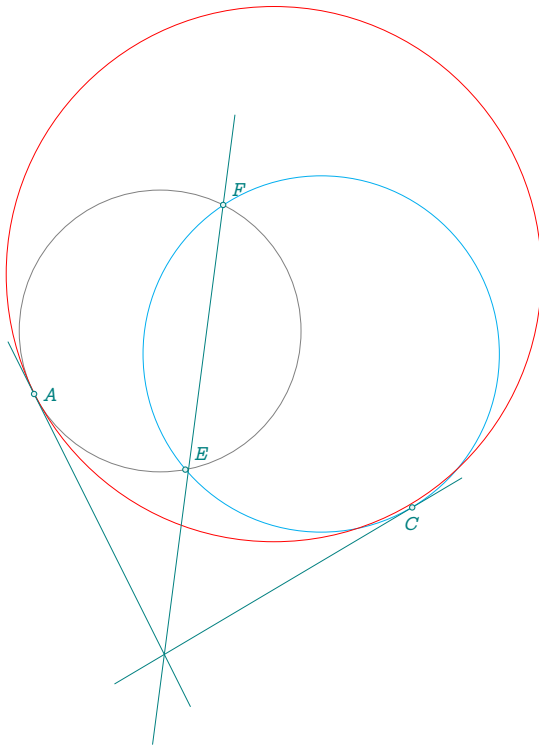
```

File three_chords2.lua

```

init_elements()
z.A = point(-1, 0)
z.C = point(4, -1.5)
z.E = point(1, -1)
z.F = point(1.5, 2.5)
T.AEF = triangle(z.A, z.E, z.F)
T.CEF = triangle(z.C, z.E, z.F)
z.w = T.AEF.circumcenter
z.x = T.CEF.circumcenter
C.wE = circle(z.w, z.E)
C.xE = circle(z.x, z.E)
L.Aw = line(z.A, z.w)
L.Cx = line(z.C, z.x)
z.G = intersection(L.Aw, L.Cx)
z.a = C.wE:tangent_at(z.A).pb
z.c = C.xE:tangent_at(z.C).pb
L.aA = line(z.a, z.A)
L.cC = line(z.c, z.C)
z.I = intersection(L.aA, L.cC)

```



2.42 South Pole

Theorem 24: South Pole

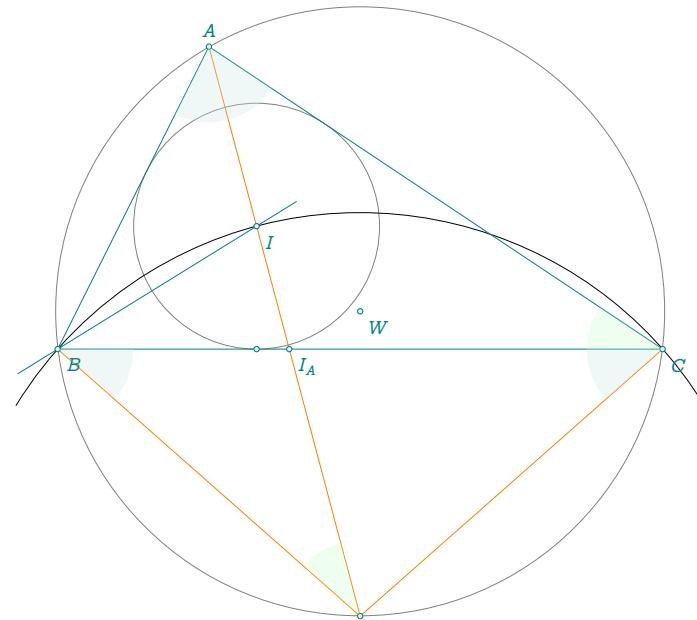
The (interior) bisector from the vertex of a triangle and the perpendicular bisector of the opposite side intersect on the circumscribed circle.

Proof. The triangle $\widehat{BOC}$ is isosceles because $\widehat{CBO}$ and $\widehat{BCO}$ are equal ($\widehat{CBO} = \widehat{OAB} = \widehat{BCO}$). O is thus on the bisector of $[BC]$ and the arcs $\widehat{BO}$ and $\widehat{OC}$ have the same length. O is called the "south pole" of triangle ABC .

Then let us show that the triangle BIO is isosceles at O . Let $\widehat{BAC} = \beta$, $\widehat{ABC} = \gamma$ and $\widehat{BCA} = \alpha$, then $\widehat{BIO} = 180^\circ - \alpha - \frac{\beta}{2} - \frac{\gamma}{2} = \frac{\beta}{2} + \frac{\gamma}{2} = \widehat{IBO}$ so triangle BIO is isosceles at L . We deduce that $OB = OI = OC$ and B, I and C are on the circle of center O . ■

File south.lua

```
init_elements()
z.A = point(2, 4)
z.B = point(0, 0)
z.C = point(8, 0)
T.ABC = triangle(z.A, z.B, z.C)
z.I, z.i = T.ABC.in_circle():get()
z.W = T.ABC.circumcenter
C.WA = circle(z.W, z.A)
z.O = C.WA.south
L.AO = line(z.A, z.O)
L.BC = line(z.B, z.C)
z.I_A = intersection(L.AO, L.BC)
```



2.43 Circle through incircle

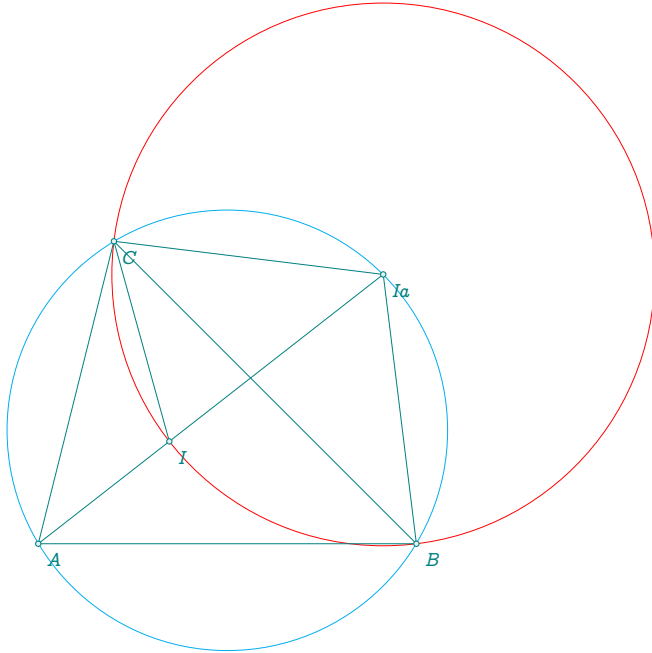
File incircle.lua

```
init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
```

```

z.C = point(1, 4)
T.ABC = triangle(z.A, z.B, z.C)
z.I = T.ABC.incenter
z.O = T.ABC.circumcenter
T.BCI = triangle(z.B, z.C, z.I)
z.Ia = T.BCI.circumcenter

```



$\widehat{IaB} = \widehat{IaC}$ so (AIa) is a bisector of $\widehat{BAC}$
 Then $\widehat{ICB} = 1/2 \widehat{IaB} = 1/2 \widehat{AIaB} = 1/2 \widehat{ACB}$
 So (CI) is a bisector of $\widehat{ACB}$
 Conclusion: I is the incenter of ABC

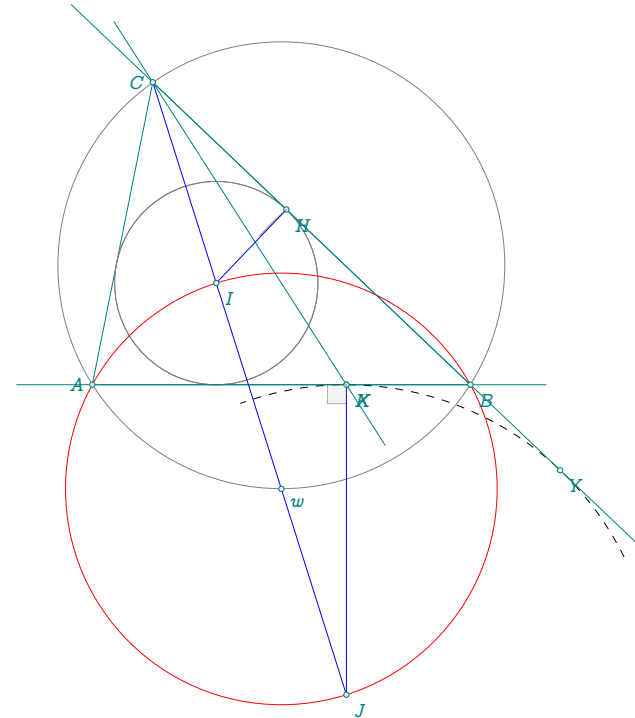
2.44 Euler's relation

Let (Γ) be the circumscribed circle of center O to the triangle ABC . Let I and J be the centers of the inscribed and exinscribed circles in the angle $\hat{A}$ to the triangle ABC admitting r_1 and r_2 as respective radii
 File euler.lua

```

init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
z.C = point(0.8, 4)
T.ABC = triangle(z.A, z.B, z.C)
z.I = T.ABC.incenter
z.J, z.K = T.ABC.ex_circle(2):get()
_, z.H = T.ABC.in_circle():get()
z.X, z.Y, _ = T.ABC.projections(z.J)
z.O = T.ABC.circumcenter
T.IBA = triangle(z.I, z.B, z.A)
z.w = T.IBA.circumcenter

```



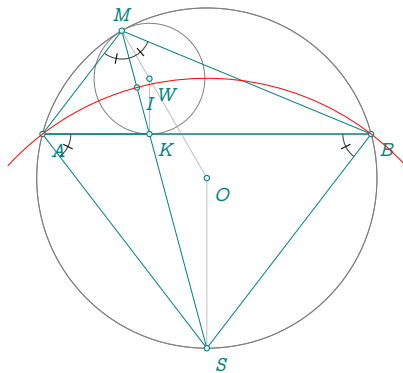
The quadrilateral $AIBJ$ having two opposite right angles is inscribed in a circle whose center ω is the middle of IJ . This center ω belongs to the perpendicular bisector of $[AB]$ and to the interior bisector of the angle $\widehat{ACB}$ and is thus the middle of the arc $\widehat{AB}$ south pole of the circle circumscribing the triangle ABC . (2.42).

2.45 Circle in circular segment

There is a tangent circle $\mathcal{C}$ to the main circle at M and I . The points M , I and S (south pole) are aligned. The circle $\mathcal{C}(S, A)$ is orthogonal to $\mathcal{C}$. The tangents at their points of contact pass through S .

File segmentaire.lua

```
init_elements()
z.O = point(O, O)
z.o = point(3, O)
z.A = point:polar(3, 11 * math.pi / 12)
z.B = point:polar(3, math.pi / 12)
z.M = point:polar(3, 2 * math.pi / 3)
T.ABM = triangle(z.A, z.B, z.M)
C.Oo = circle(z.O, z.o)
L.AB = line(z.A, z.B)
z.S = C.Oo.south
L.MS = line(z.M, z.S)
z.K = intersection(L.MS, L.AB)
z.i = z.K:north()
L.Ki = line(z.K, z.i)
L.OM = line(z.O, z.M)
z.W = intersection(L.OM, L.Ki)
--%T.AKM = triangle(z.A, z.K, z.M)
--z.w = T.AKM.circumcenter
--C.wA = circle(z.w,z.A)
--z.t1,z.t2 = C.wA.tangent_at(z.A):get()
z.I = T.ABM.incenter
```



Proof. The fact that MI passes through the midarc follows by taking the

homothety at M mapping $C_{W,M}$ to $C_{S,A}$: it should map the “south pole” of $C_{W,M}$ to that of $C_{S,A}$, ergo it maps S to I. Meanwhile, $SI \times SM = SA^2$ follows from triangle MAI similar to SBI.

Theorem "South Pole" : The bisector at M passes through the south pole S .

MWI = MOS so IMW = SMO and M , I and S are aligned.

The inversion of center S and circle $C_{S,A}$ transforms the circle $C_{O,M}$ into a straight line (A, B) . The image of M is I , which is probably sufficient to justify the alignment. The circle $C_{W,M}$ is globally invariant and therefore orthogonal to $C_{S,A}$. The tangents at the points of intersection pass through S .

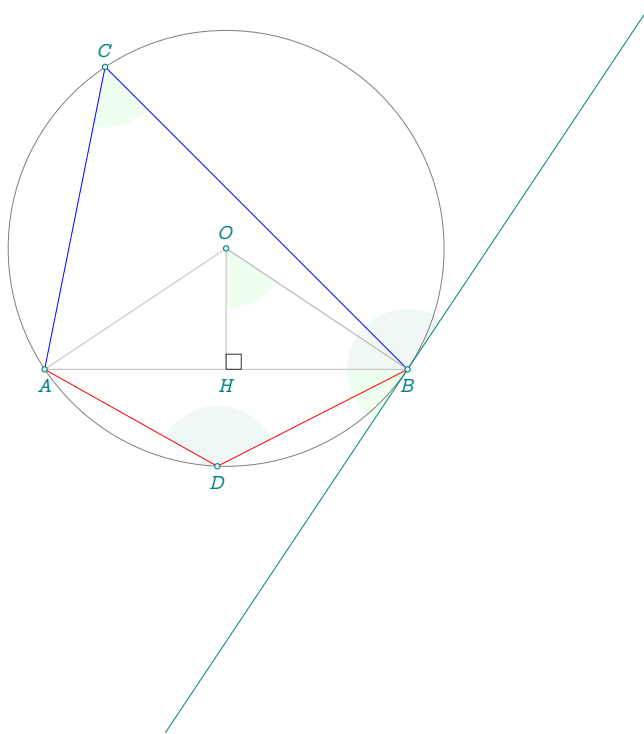
2.46 Tangent-Chord Theorem

Theorem 25: Tangent-Chord Theorem

The Tangent-Chord Theorem states that the angle formed between a chord and a tangent line to a circle is equal to the inscribed angle on the other side of the chord.

File tangent_chord.lua

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(1, 5)
z.Bp = point(2, 0)
T.ABC = triangle(z.A, z.B, z.C)
L.AB = line(z.A, z.B)
z.O = T.ABC.circumcenter
C.OA = circle(z.O, z.A)
z.D = C.OA:point(0.15)
L.AO = line(z.A, z.O)
z.b1, z.b2 = C.OA:tangent_at(z.B):get()
z.H = L.AB:projection(z.O)
```



```

z.M_b = T.ABC.ca.mid
z.M_c = T.ABC.ab.mid
L.MaMb = line(z.M_a, z.M_b)
L.MaMc = line(z.M_a, z.M_c)
L.MbMc = line(z.M_b, z.M_c)
L.AI = line(z.A, z.I)
L.BI = line(z.B, z.I)
L.CI = line(z.C, z.I)
z.G = T.ABC.circumcenter
C.Hc = circle:diameter(z.I, z.C)
z.Hc = C.Hc.center
C.Ha = circle:diameter(z.I, z.A)
z.Ha = C.Ha.center
C.Hb = circle:diameter(z.I, z.B)
z.Hb = C.Hb.center
z.J = intersection(L.AI, L.MaMb)
z.K = intersection(L.BI, L.MaMb)
z.L = intersection(L.CI, L.MbMc)

```

2.47 Iran Lemma

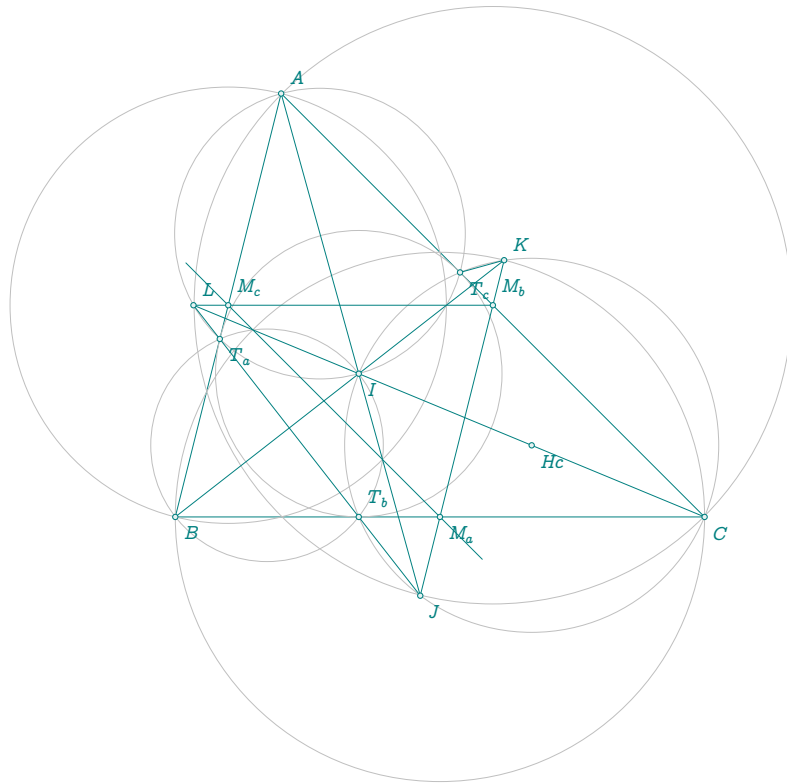
Let ABC be a triangle. Let I be the incenter, M_a, M_b, M_c be the midpoints of $[BC], [CA], [AB]$ and let T_a, T_b, T_c be the points of tangency of the incircle with $[BC], [CA], [AB]$. Then $(AI), M_aM_b, T_aT_c$, the circle $\mathcal{C}(G, A)$ and the circle $\mathcal{C}(M_b, A)$ concur.

File iran.lua:

```

init_elements()
z.A = point(1, 4)
z.B = point(0, 0)
z.C = point(5, 0)
T.ABC = triangle(z.A, z.B, z.C)
z.I = T.ABC.incenter
z.T_a, z.T_b, z.T_c = T.ABC.projections(z.I)
z.M_a = T.ABC.bc.mid

```



Proof. $\widehat{GDC} = 180^\circ - \widehat{GDB} = 180^\circ - \widehat{GFB} = \widehat{AFE} = \widehat{AEF} = \widehat{GEC}$
 $\widehat{GDC} = \widehat{GEC}$

The points I, E, G, C and D are concyclic. The quadrilateral $IGCD$ is inscriptible then $\widehat{BGC} = \widehat{IGC} = \widehat{CDI} = 90^\circ$

M is the midpoint of the hypotenuse $[BC]$. $\widehat{GMC} = 2\widehat{GBC} = \widehat{ABC}$. Thus the lines (AB) and (GM) are parallel. Then G, N and M are aligned. ■

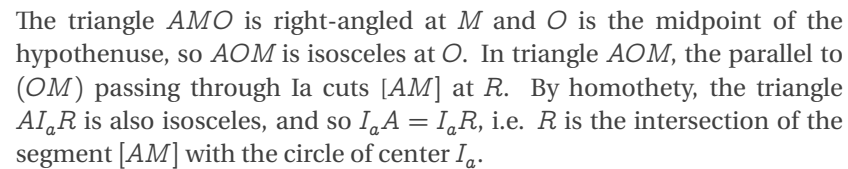
2.48 Tangent of two circles

1. Version "Outside"

File outside.lua

```
init_elements()
z.A = point(0, 0)
z.B = point(10, 0)
L.AB = line(z.A, z.B)
```

```
z.Ia = point(2.75, 0)
z.Ib = point(9, 0)
C.IaA = circle(z.Ia, z.A)
C.IbB = circle(z.Ib, z.B)
L.axis = C.IaA:radical_axis(C.IbB)
z.ra, z.rb = L.axis:get()
C.OB = L.AB:diameter()
z.O = L.AB.mid
z.M, z.Mp = intersection(L.axis, C.OB)
L.AM = line(z.A, z.M)
L.BM = line(z.B, z.M)
z.R = intersection(L.AM, C.IaA)
_, z.S = intersection(L.BM, C.IbB)
L.RS = line(z.R, z.S)
z.Ap = C.IaA:antipode(z.A)
z.Bp = C.IbB:antipode(z.B)
z.J = intersection(L.axis, L.RS)
L.RAp = line(z.R, z.Ap)
L.SBp = line(z.S, z.Bp)
z.N = intersection(L.RAp, L.SBp)
z.V, z.U = intersection(L.RS, C.OB)
C.JM = circle(z.J, z.M)
z.u, z.v = intersection(L.AB, C.JM)
T = triangle(z.A, z.R, z.S)
z.w = T.circumcenter
C.MU = circle(z.M, z.U)
C.wA = circle(z.w, z.A)
z.x, z.y = intersection(C.MU, C.wA)
```

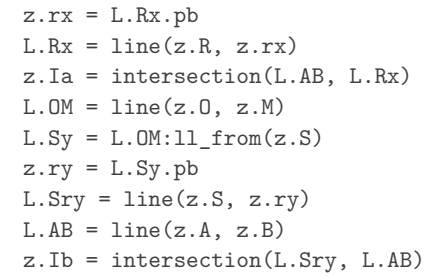


Then we show that N belongs to the radical axis $RMSN$ is a rectangle (obvious). $MR = NS$ and $MS = NR$.

$$\frac{NA'}{MB} = \frac{NB'}{MA} \text{ either } NA' \times MA = NB' \times MB$$
$$NA' \times MA = NB' \times MB \text{ and } NS \times MA = NR \times MB$$

The $ARSB$ quadrilateral is inscribable

AlterMundus



Theorem - Image of a circle passing through the ∞ - pole

AMB right-angled triangle, $[AB]$ hypotenuse and diameter of circumscribed circle.

Consider the inversion with center M and circle $\mathcal{C}(M, U)$.

AlterMundus

and V intersection of $\mathcal{C}(O, M)$ with the inversion circle are invariant. The line (UV) is orthogonal to the line (MO) (parallel to the tangent at M to $\mathcal{C}(O, M)$).

A and B have images R and S belonging to the line (UV) and to the segments $[AM]$ and $[BM]$.

The triangles RMS and AMB are similar. We deduce that the angles $\widehat{MBA}$ and $\widehat{ARS}$ are supplementary and therefore that the quadrilateral $ARSB$ is inscribable in a circle $\mathcal{C}(w, A)$ which is the image of the line (AB) . The circle $\mathcal{C}(w, A)$ is globally invariant and therefore orthogonal to the inversion circle $\mathcal{C}(M, U)$.

Let's draw the parallel through R to the line (MO) . It is orthogonal at R to the line (UV) . The line (UV) is tangent to the circle $\mathcal{C}(Ia, R)$ at R and $\mathcal{C}(Ib, S)$ at S .

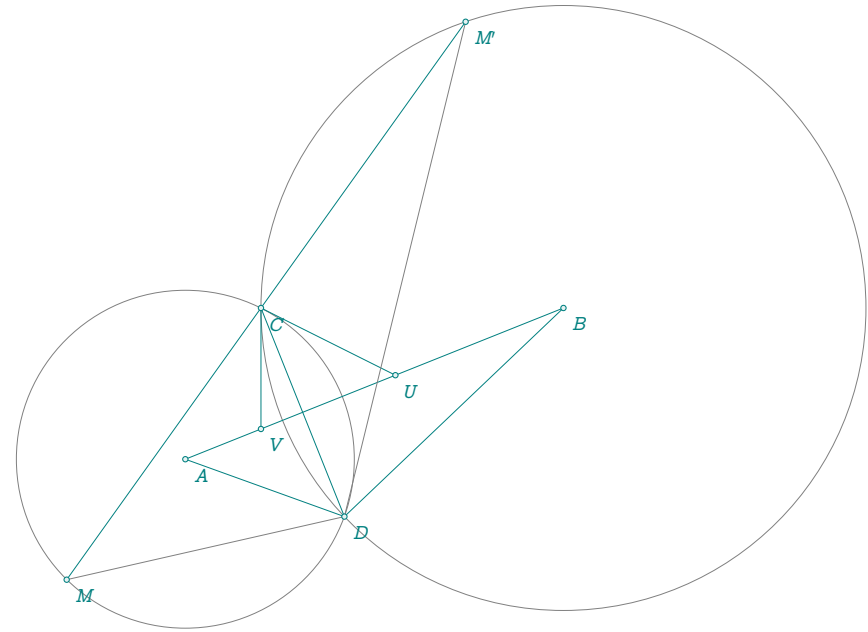
2.49 Angle of two intersecting circles

Definition 9: Angle of two intersecting circles

Angle at which one of the intersection points sees the segment formed by the two centers.

File `angle_circles.lua`

```
init_elements()
z.A = point(0, 0)
z.B = point(5, 2)
L.AB = line(z.A, z.B)
z.C = point(1, 2)
C.AC = circle(z.A, z.C)
C.BC = circle(z.B, z.C)
z.M = C.AC:point(0.45)
L.MC = line(z.M, z.C)
z.Mp = intersection(L.MC, C.BC)
_, z.D = intersection(C.AC, C.BC)
L.tAC = C.AC:tangent_at(z.C)
L.tBC = C.BC:tangent_at(z.C)
z.r, z.s = L.tAC:get()
z.u, z.v = L.tBC:get()
z.U = intersection(L.AB, L.tAC)
z.V = intersection(L.AB, L.tBC)
```



Theorem 26

Any secant passing through one of the points common to two circles is seen from the other common point at a constant angle equal to the angle of the two circles.

Proof. The triangles ABD and $MM'D$ are similar ... ■

Definition 10

The angle of two intersecting circles is therefore the angle at which a secant passing through the other common point can be seen from common point.

Theorem 27

The angle formed by the tangents at a point common to both circles is supplementary to the angle of the two circles.

Proof. $\widehat{UCD} = \widehat{DMC}$ and $\widehat{VCD} = \widehat{DM'C}$ ■

Theorem 28

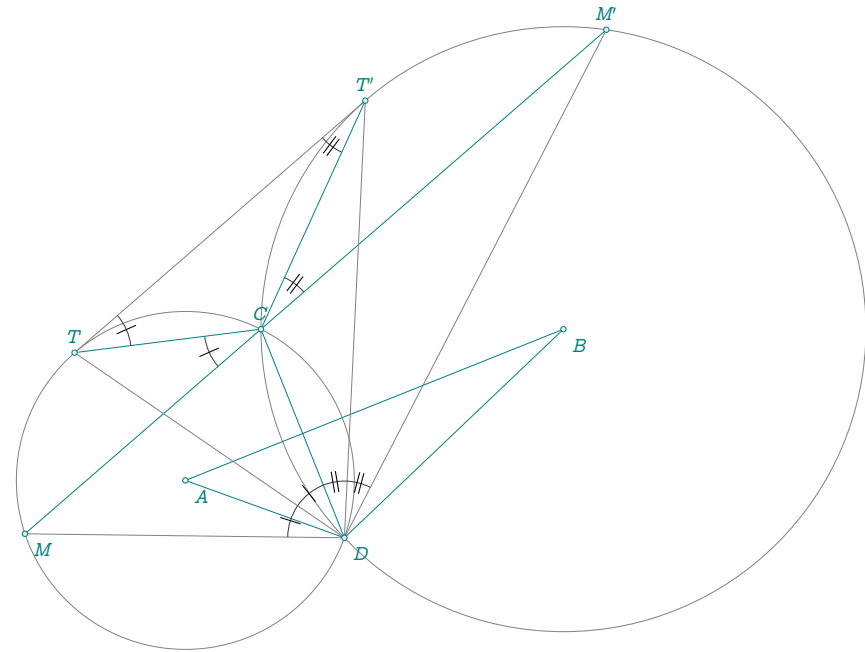
Let be a tangent common to both circles at T and T' (closest to C). Let a secant parallel to this tangent pass through C . Then the segment $[T'T']$ is seen from the other common point D at an angle equal to half the angle of the two circles.

File noname.lua

```

init_elements ()
z.A = point(0, 0)
z.B = point(5, 2)
L.AB = line(z.A, z.B)
z.C = point(1, 2)
C.AC = circle(z.A, z.C)
C.BC = circle(z.B, z.C)
L.TTp = C.AC:common_tangent(C.BC)
z.T, z.Tp = L.TTp:get()
z.M = C.AC:point(0.45)
L.MC = line(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)

```



Proof. A few angle relationships show that T and T' are the midpoints of the arcs $\widehat{MC}$ and $\widehat{M'C}$. (DT) and (DT') are bisectors of $\widehat{MDC}$ and $\widehat{CDM'}$. So $\widehat{MDM'} = 2\widehat{TD T'}$ ■

Theorem 29: Orthogonality condition

For two circles to be orthogonal, it is necessary and sufficient for a secant passing through one of their common points to be seen from the other common point at a right angle.

File orthogonality.lua

```

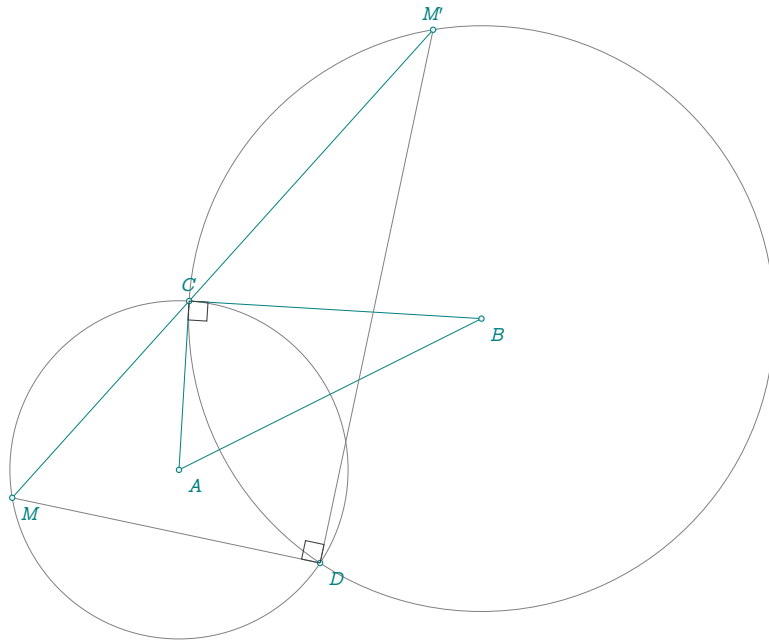
init_elements()
z.A = point(0, 0)
z.B = point(4, 2)
L.AB = line(z.A, z.B)
z.a = point(1, 2)
C.Aa = circle(z.A, z.a)
C.BC = C.Aa:orthogonal_from(z.B)
z.C, z.D = intersection(C.Aa, C.BC)
C.AC = circle(z.A, z.C)

```

```

L.TTp = C.AC:common_tangent(C.BC)
z.T, z.Tp = L.TTp:get()
z.M = C.AC:point(0.45)
L.MC = line(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)

```



2.50 Specific lines of a triangle

Theorem 30

The bisectors of a triangle intersect at a point called the centre of the circumscribed circle (circumcenter) of the triangle.

Existence of the circumscribed circle

Proof. Let ABC be a triangle. The bisectors of sides $[AB]$ and $[BC]$ intersect at a point O . Then $OA = OB$ and $OB = OC$, from which we deduce that $OA = OC$,

so that O belongs to the perpendicular bisector of $[AC]$. Since $OA = OB = OC$ the points A, B and C lie on a circle whose centre is O . ■

Theorem 31: Intersection of altitudes

The heights of a triangle intersect at a point called the orthocentre.

Proof. Simply construct the triangle whose sides are parallel to the sides of the given triangle and pass through the vertices. The perpendicular bisectors of the sides of a triangle intersect. Let O be their common point, the centre of the circle circumscribing the triangle.

We deduce that the altitudes of a triangle are concurrent. Their common point is the orthocentre marked H . We need only consider the straight lines parallel to the sides and passing through the vertices. The altitudes of the initial triangle are the bisectors of the sides of the final triangle.

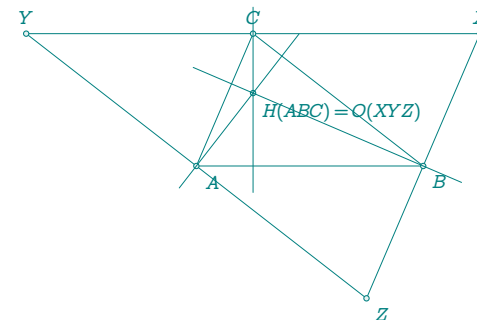
The altitudes of one triangle are the perpendicular bisectors of the other. ■

File `orthic.lua`

```

init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(1.5, 3.5)
T.ABC = triangle(z.A, z.B, z.C)
L.YZ = T.ABC.bc:ll_from(z.A)
L.XZ = T.ABC.ca:ll_from(z.B)
L.XY = T.ABC.ab:ll_from(z.C)
z.X = intersection(L.XY, L.XZ)
z.Y = intersection(L.XY, L.YZ)
z.Z = intersection(L.XZ, L.YZ)
z.H_a, z.H_b, z.H_c = T.ABC:orthic():get()
z.H = T.ABC.orthocenter

```



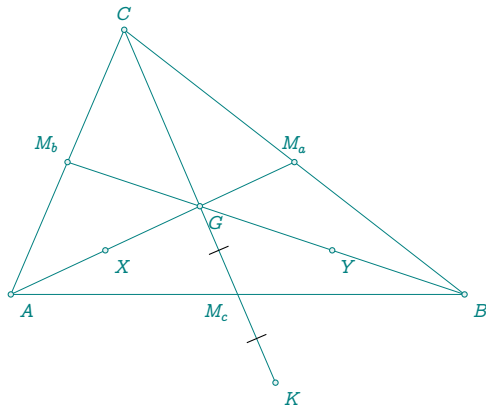
Theorem 32: Intersection of medians

The medians of a triangle intersect at a point called the isobarycentre or centre of gravity (centroid). This point, marked G , is located $2/3$ of the way along the medians.

Proof. $M_b M_a Y X$ is a parallelogram whose diagonals intersect in the middle. So G is two-thirds of the way between $[AM_a]$ and $[BM_b]$. The same applies to $[BM_b]$ and $[CM_c]$, so G is common to all three medians.

File medians.lua

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(1.5, 3.5)
T.ABC = triangle(z.A, z.B, z.C)
T.medial = T.ABC:medial()
z.G = T.ABC.centroid
z.M_a, z.M_b, z.M_c = T.medial:get()
z.X = tkz.midpoint(z.G, z.A)
z.Y = tkz.midpoint(z.G, z.B)
z.K = z.M_c:symmetry(z.G)
```



$\overrightarrow{CG} = \overrightarrow{GK}$ and $\overrightarrow{GA} + \overrightarrow{GB} = \overrightarrow{GK}$ implies that:

$$\overrightarrow{GA} + \overrightarrow{GB} + \overrightarrow{GC} = \vec{0}$$

.

G est appelé isobarycentre ou centre de gravité de A, B et C .

Remarque : Pour tout point M du plan,

$$\overrightarrow{MA} + \overrightarrow{MB} + \overrightarrow{MC} = 3\overrightarrow{MG}.$$

■

2.51 Existence of the Euler line

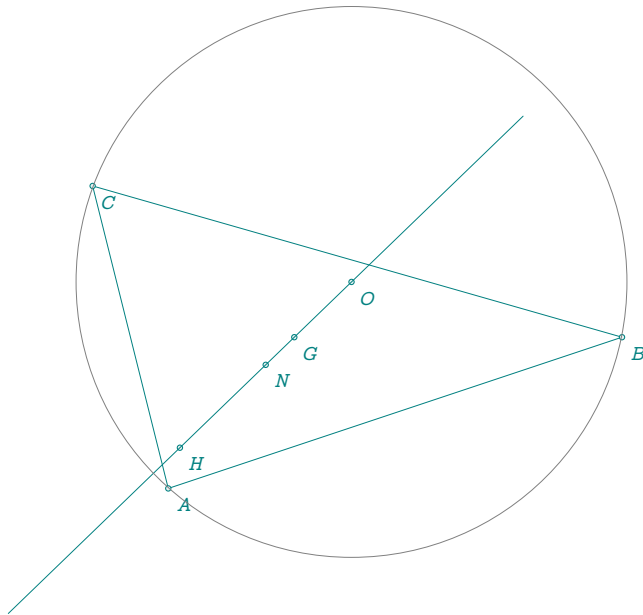
Suppose that the triangle $\mathcal{T}$ is not equilateral (otherwise O and G are merged).

The Euler line is the line defined by these two points.

Let $\mathcal{T}$ be the triangle ABC , let O be the centre of the circle circumscribing $\mathcal{T}$, let H be the orthocentre of $\mathcal{T}$ and G its centre of gravity.

File euler_line.lua

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 2)
z.C = point(-1, 4)
T.ABC = triangle(z.A, z.B, z.C)
z.O = T.ABC.circumcenter
z.H = T.ABC.orthocenter
z.G = T.ABC.centroid
z.N = T.ABC.eulercenter
```



2.52 Viviani's Theorem

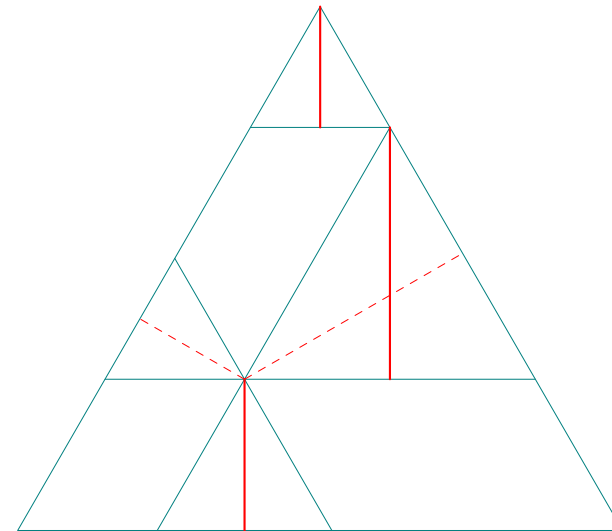
Theorem 33: Viviani's Theorem

Viviani's theorem, named after Vincenzo Viviani, states that the sum of the shortest distances from any interior point to the sides of an equilateral triangle equals the length of the triangle's altitude. [Wikipedia].

```
init_elements()
z.A = point(0, 0)
z.B = point(8, 0)
L.AB = line(z.A, z.B)
T.equ = L.AB:equilateral()
z.C = T.equ.pc
z.I = point(3, 2)
L.IAB = T.equ.ab:ll_from(z.I)
L.IBC = T.equ.bc:ll_from(z.I)
L.ICA = T.equ.ca:ll_from(z.I)
z.b = intersection(L.IAB, T.equ.ca)
z.a = intersection(L.IAB, T.equ.bc)
```

```
z.c = intersection(L.IBC, T.equ.ab)
z.d = intersection(L.IBC, T.equ.ca)
z.e = intersection(L.ICA, T.equ.ab)
z.f = intersection(L.ICA, T.equ.bc)
L.last = T.equ.ab:ll_from(z.f)
z.g = intersection(L.last, T.equ.ca)
z.pC = L.last:projection(z.C)
z.pIca = T.equ.ca:projection(z.I)
z.pIbc = T.equ.bc:projection(z.I)
z.pIAB = L.IAB:projection(z.f)
z.pIab = T.equ.ab:projection(z.I)
```

Here's the visual demonstration I gave at the CAPES oral exam in 1989 to become a teacher.



2.53 Reuschle's theorem

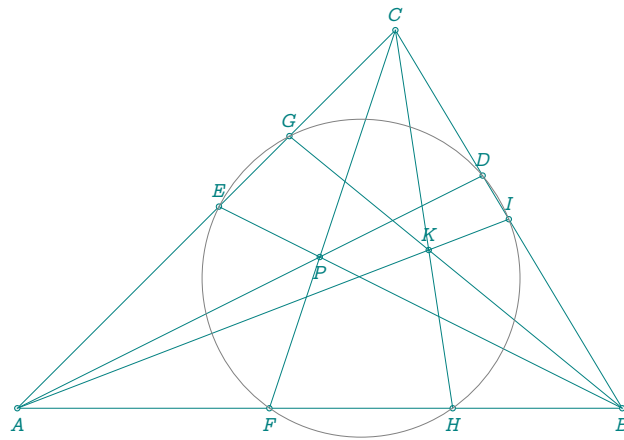
Theorem 34: Reuschle's theorem

In elementary geometry, Reuschle's theorem describes a property of the cevians of a triangle intersecting in a common point and is named after the German mathematician Karl Gustav Reuschle (1812–1875). It is also known as Terquem's theorem after the French mathematician Olry Terquem (1782–1862), who published it in 1842.

```

init_elements()
z.A = point(0, 0)
z.B = point(8, 0)
z.C = point(5, 5)
z.P = point(4, 2)
T.ABC = triangle(z.A, z.B, z.C)
T.cev = T.ABC:cevian(z.P)
z.D, z.E, z.F = T.cev:get()
C.circ = T.ABC:circum_circle(z.P)
z.O = C.circ.center
z.T = C.circ.through
z.G = intersection(C.circ, T.ABC.ca)
_, z.H = intersection(C.circ, T.ABC.ab)
z.I = intersection(C.circ, T.ABC.bc)
L.AI = line(z.A, z.I)
L.BG = line(z.B, z.G)
z.K = intersection(L.AI, L.BG)

```



```

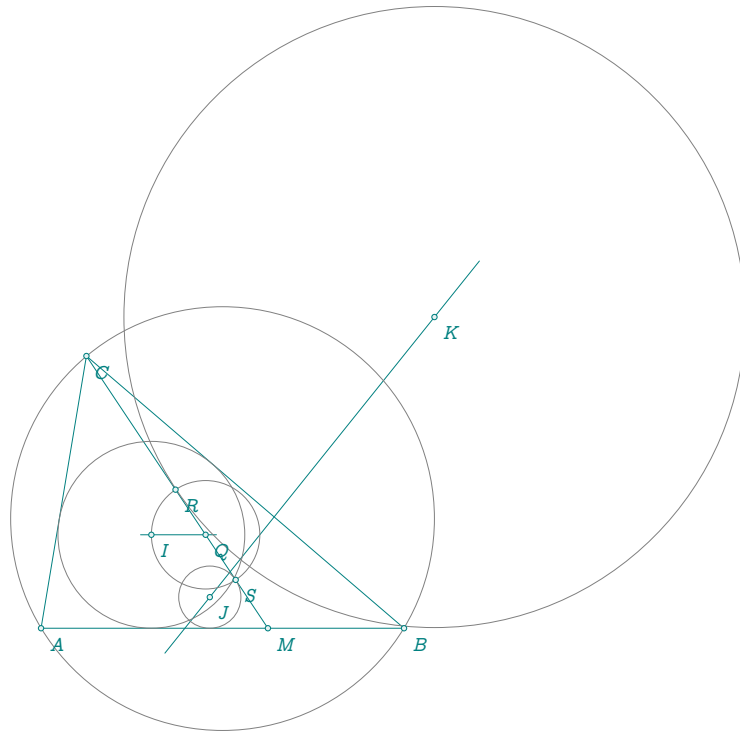
init_elements()
z.A = point(0, 0)
z.B = point(8, 0)
z.C = point(1, 6)
z.M = point(5, 0)
L.CM = line(z.C, z.M)
T.ABC = triangle(z.A, z.B, z.C)
C.circ = T.ABC:circum_circle()
z.O = C.circ.center
C.ins = T.ABC:in_circle()
z.I = C.ins.center
z.T = C.ins.through
L.l1 = T.ABC.ab:ll_from(z.I)
z.Q = intersection(L.l1, L.CM)
C.QI = circle(z.Q, z.I)
z.R, z.S = intersection(C.QI, L.CM)
L.BMC = tkz.bisector(z.M, z.B, z.C)
z.x = L.BMC.pb
L.CMA = tkz.bisector(z.M, z.C, z.A)
z.y = L.CMA.pb
L.pS = L.CM:ortho_from(z.S)
L.pR = L.CM:ortho_from(z.R)
z.J = intersection(L.pS, L.CMA)
z.K = intersection(L.pR, L.BMC)

```

2.54 Thébault's problem III

Theorem 35: Thébault's problem III

Given any triangle ABC , and any point M on BC , construct the incircle and circumcircle of the triangle. Then construct two additional circles, each tangent to AM , BC , and to the circumcircle. Then their centers and the center of the incircle are collinear.[wikipedia]

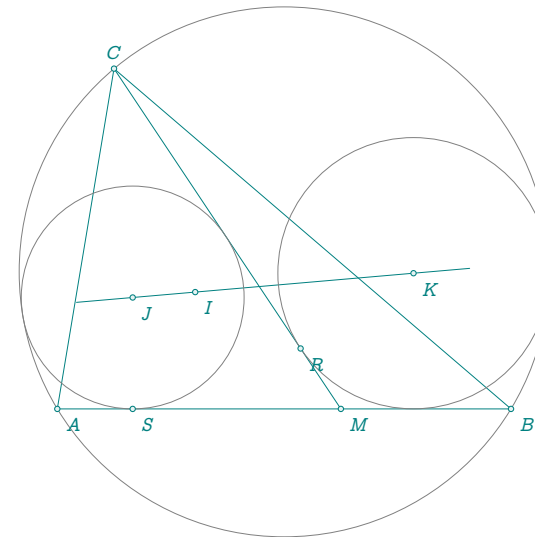


2.55 Thébault's problem III with a method

The `c_c` or `thebault` method selects a vertex of a triangle to construct a circle tangent to the two adjacent sides and to a given circle passing through the opposite vertices.

```
init_elements()
z.A = point(0, 0)
z.B = point(8, 0)
z.C = point(1, 6)
z.M = point(5, 0)
T.ABC = triangle(z.A, z.B, z.C)
C.circ = T.ABC:circum_circle()
z.O = C.circ.center
C.ins = T.ABC:in_circle()
z.I = C.ins.center
z.T = C.ins.through
```

```
T.MCA = triangle(z.M, z.C, z.A)
T.MBC = triangle(z.M, z.B, z.C)
z.J, z.S = T.MCA:thebault(z.M, C.circ):get()
z.K, z.R = T.MBC:thebault(z.M, C.circ):get()
```



2.56 Soddy circles of a triangle

Theorem 36: Soddy

In geometry, the Soddy circles of a triangle are two circles associated with any triangle in the plane.[wikipedia]

Given three noncollinear points, construct three tangent circles such that one is centered at each point and the circles are pairwise tangent to one another. Then there exist exactly two nonintersecting circles that are tangent to all three circles. These are called the inner and outer Soddy circles, and their centers are called the inner and outer Soddy centers, respectively.

Weisstein, Eric W. "Soddy Circles." From MathWorld—A Wolfram Web Resource

2.57 Soddy circle without function

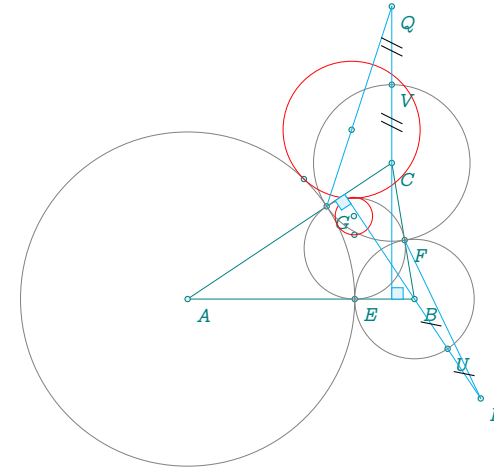
The following construction is inspired by that of the following site:

Mathafou

```

init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
z.C = point(4.5, 3)
T.ABC = triangle(z.A, z.B, z.C)
z.I = T.ABC.incenter
z.E, z.F, z.G = T.ABC.projections(z.I)
C.ins = circle(z.I, z.E)
T.orthic = T.ABC.orthic()
z.Ha, z.Hb, z.Hc = T.orthic.get()
L.Ah = line(z.A, z.Ha)
L.Bh = line(z.B, z.Hb)
L.Ch = line(z.C, z.Hc)
C.ins = circle(z.I, z.E)
C.CF = circle(z.C, z.F)
C.AG = circle(z.A, z.G)
C.BE = circle(z.B, z.E)
_, z.V = intersection(L.Ch, C.CF, {near = z.I})
z.Q = z.V:symmetry(z.C)
_, z.U = intersection(L.Bh, C.BE, {near = z.I})
z.P = z.U:symmetry(z.B)
L.PF = line(z.P, z.F)
L.QG = line(z.Q, z.G)
L.TGE = C.AG:tangent_at(z.G)
L.TGE = C.BE:tangent_at(z.E)
L.GE = line(z.G, z.E)
L.FE = line(z.F, z.E)
z.wi = intersection(L.PF, L.QG)
L.wiA = line(z.wi, z.A)
z.ti = intersection(L.wiA, C.AG, {near = z.I})
C.soddy_int = circle(z.wi, z.ti)
C.soddy_ext = C.ins:inversion(C.soddy_int)
z.wo, z.to = C.soddy_ext.get()

```

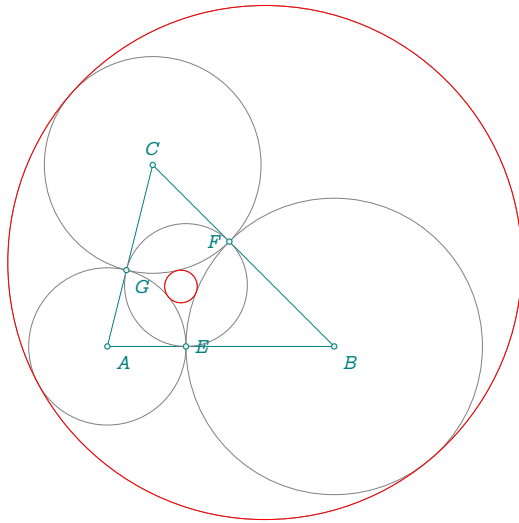


2.58 Soddy circle with function

```

init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
z.C = point(1, 4)
T.ABC = triangle(z.A, z.B, z.C)
z.I = T.ABC.incenter
z.E, z.F, z.G = T.ABC.projections(z.I)
C.ins = circle(z.I, z.E)
C.soddy_int = T.ABC:soddy_circle()
C.soddy_ext = T.ABC:soddy_circle("outer")
z.wi, z.ti = C.soddy_int.get()
z.wo, z.to = C.soddy_ext.get()

```



2.59 Six circles in a triangle

Theorem 37: Six circles in a triangle

In geometry, the six circles theorem relates to a chain of six circles together with a triangle, such that each circle is tangent to two sides of the triangle and also to the preceding circle in the chain. The chain closes, in the sense that the sixth circle is always tangent to the first circle.[1][2] It is assumed in this construction that all circles lie within the triangle, and all points of tangency lie on the sides of the triangle. [Wikipedia]

The file `lua/lemmas/six_circles.lua` used in this example:

```
-- six_circle.lua
local r = ...

function newcircle(T, C)
  -- homothétie du triangle à partir de l'incentre
  local k = 1 + C.radius / T.inradius
  local NT = T.incenter:homothety(k, T)

  -- les deux côtés qui encadrent le cercle cherché
  local Lba = line(NT.pb, NT.pa)
  local Lbc = line(NT.pb, NT.pc)
```

```
-- cercles tangent à Lba et Lbc et passant par C.center
local pc, pa = Lba:LLP(Lbc, C.center)

-- petite fonction utilitaire : construit le candidat n
local function candidate(idx)
  local w = pc:get(idx)
  if not w then return nil end
  local t = T.bc:projection(w)
  local r_new = w:length(t) -- rayon du nouveau cercle
  local d = w:length(C.center) -- distance entre centres
  return w, t, r_new, d
end

local w1, t1, r1, d1 = candidate(1)
local w2, t2, r2, d2 = candidate(2)

local EPS = tkz.epsilon or 1e-6

-- test de tangence externe avec C
local function good_ext(d, r_new)
  return math.abs(d - (r_new + C.radius)) < 10 * EPS
end

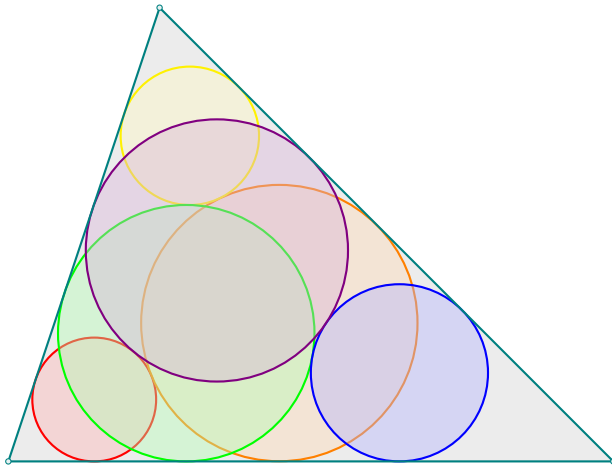
-- choix du bon candidat
if w1 and good_ext(d1, r1) and not (w2 and good_ext(d2, r2)) then
  return w1, t1
elseif w2 and good_ext(d2, r2) and not (w1 and good_ext(d1, r1)) then
  return w2, t2
elseif w1 and w2 then
  -- si les deux ou aucun ne passent le test, on prend par défaut
  -- le cercle de plus petit rayon (habituellement le cercle intérieur)
  if r1 <= r2 then
    return w1, t1
  else
    return w2, t2
  end
else
  -- pas de solution exploitable
  return nil, nil
end
end
```

```

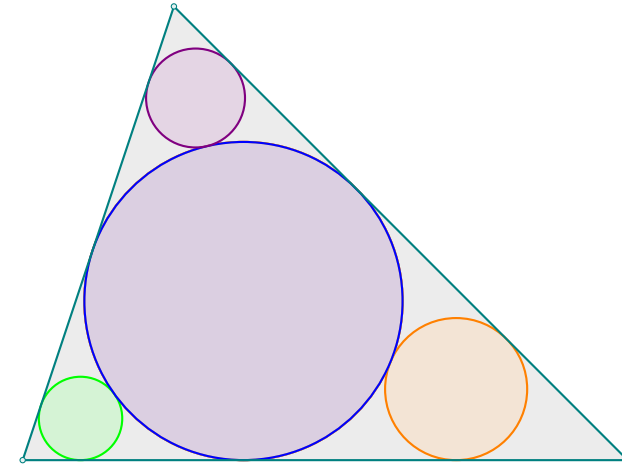
L.bA = T.ABC:bisector()
z.c1 = L.bA:report(r)
z.t1 = T.ABC.ab:projection(z.c1)
C.last = circle:new(z.c1, z.t1)

local vertices = { "A", "B", "C" }
for i = 2, 6 do
  T.used = triangle:new(
    z[vertices[math.fmod(i - 2, 3) + 1]],
    z[vertices[math.fmod(i - 1, 3) + 1]],
    z[vertices[math.fmod(i, 3) + 1]]
  )
  z["c" .. i], z["t" .. i] = newcircle(T.used, C.last)
  C.last = circle:new(z["c" .. i], z["t" .. i])
end

```



When the first circle is the incircle then you get only 4 circles.



2.60 Circle-Point Midpoint Theorem

Theorem 38: Circle-Point Midpoint Theorem

Taking the locus of midpoints from a fixed point to a circle of radius r results in a circle of radius $r/2$.

Weisstein, Eric W. "Circle-Point Midpoint Theorem." From MathWorld—A Wolfram Resource.

```

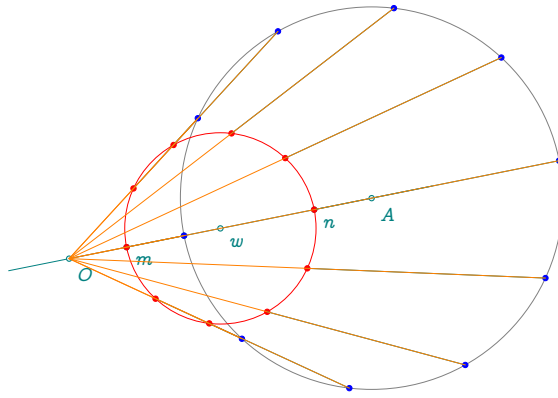
init_elements()
z.0 = point(0, 0)
z.A = point(5, 1)
z.Ta = point(8, 2)
C.A = circle(z.A, z.Ta)
L.OA = line(z.0, z.A)
z.x, z.y = intersection(C.A, L.OA, {near=z.0})
C.A.through = z.y
z.m = tkz.midpoint(z.0, z.x)
z.n = tkz.midpoint(z.0, z.y)
z.w = tkz.midpoint(z.m, z.n)
PA.A = path()
PA.c = path()
for t = 0, 1 - 1e-12, 0.1 do
  PA.A:add_point(C.A:point(t))
  local m = tkz.midpoint(z.0, C.A:point(t))

```

```

PA.c:add_point(m)
end

```



2.61 Bankoff circle

Definition 11: Bankoff circle

In geometry, the Bankoff circle or Bankoff triplet circle is a certain Archimedean circle that can be constructed from an arbelos; an Archimedean circle is any circle with area equal to each of Archimedes' twin circles. The Bankoff circle was first constructed by Leon Bankoff in 1974.[Wikipedia]

File bankoff.lua:

```

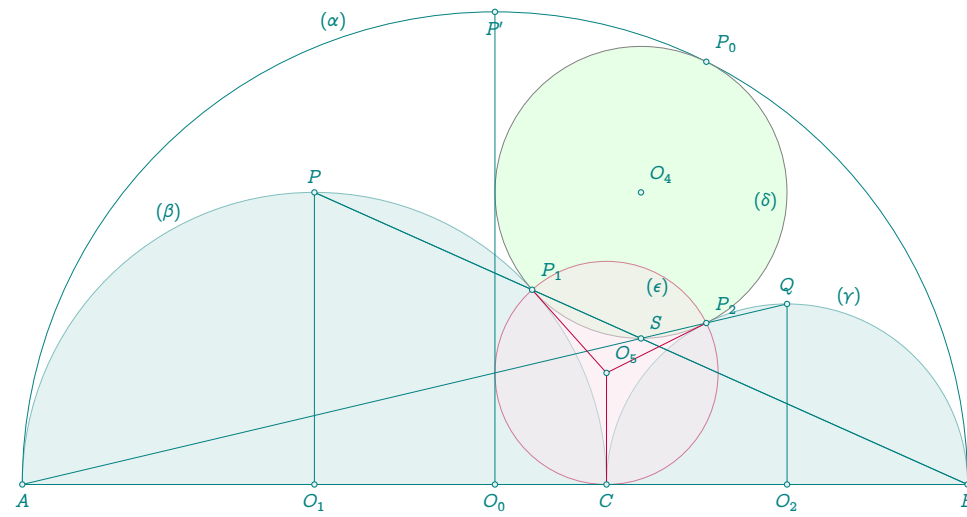
init_elements()
z.A = point(0, 0)
z.B = point(10, 0)
L.AB = line(z.A, z.B)
z.C = L.AB:gold_ratio()
L.AC = line(z.A, z.C)
L.CB = line(z.C, z.B)
z.O_0 = L.AB.mid
z.O_1 = L.AC.mid
z.O_2 = L.CB.mid
C.O0B = circle(z.O_0, z.B)
C.O1C = circle(z.O_1, z.C)

```

```

C.O2C = circle(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.O1O2 = line(z.O_1, z.O_2)
L.O0O1 = line(z.O_0, z.O_1)
L.O0O2 = line(z.O_0, z.O_2)
z.M_0 = L.O1O2:harmonic_ext(z.C)
z.M_1 = L.O0O1:harmonic_int(z.A)
z.M_2 = L.O0O2:harmonic_int(z.B)
L.BP = line(z.B, z.P)
L.AQ = line(z.A, z.Q)
z.S = intersection(L.BP, L.AQ)
L.PpO0 = line(z.Pp, z.O_0)
L.PC = line(z.P, z.C)
z.Ap = intersection(L.PpO0, L.PC)
L.CS = line(z.C, z.S)
C.M1A = circle(z.M_1, z.A)
C.M2B = circle(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(z.P_0, z.P_1, z.P_2)
z.O_4 = T.P0P1P2.circumcenter
T.CP1P2 = triangle(z.C, z.P_1, z.P_2)
z.O_5 = T.CP1P2.circumcenter

```



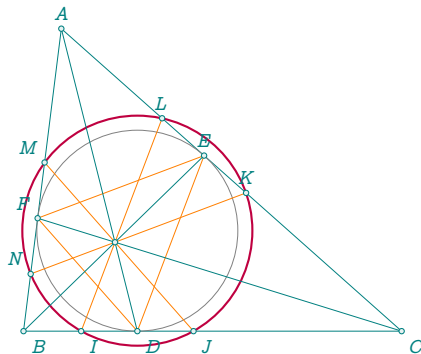
2.62 Adams's circle

Given a triangle ABC , construct the contact triangle. Now extend lines parallel to the sides of the contact triangle from the Gergonne point. These intersect the triangle ABC in the six points M, N, I, J, K, L . Adams proved in 1843 that these points are concyclic in a circle now known as the Adams' circle.

Weisstein, Eric W. "Adams' Circle." From MathWorld—A Wolfram Web Resource.

File adam.lua:

```
init_elements()
z.A = point(0.5, 4)
z.B = point(0, 0)
z.C = point(5, 0)
T.ABC = triangle(z.A, z.B, z.C)
T.ins = T.ABC:contact()
z.D, z.E, z.F = T.ins:get()
z.i = T.ABC.incenter
z.G_e = T.ABC:gergonne_point()
local function find_points(side1, side2, side3)
  local line = side1:ll_from(z.G_e)
  local p1 = intersection(line, side2)
  local p2 = intersection(line, side3)
  return p1, p2
end
z.M, z.J = find_points(T.ins.ca, T.ABC.ab, T.ABC.bc)
z.N, z.K = find_points(T.ins.bc, T.ABC.ab, T.ABC.ca)
z.I, z.L = find_points(T.ins.ab, T.ABC.bc, T.ABC.ca)
```



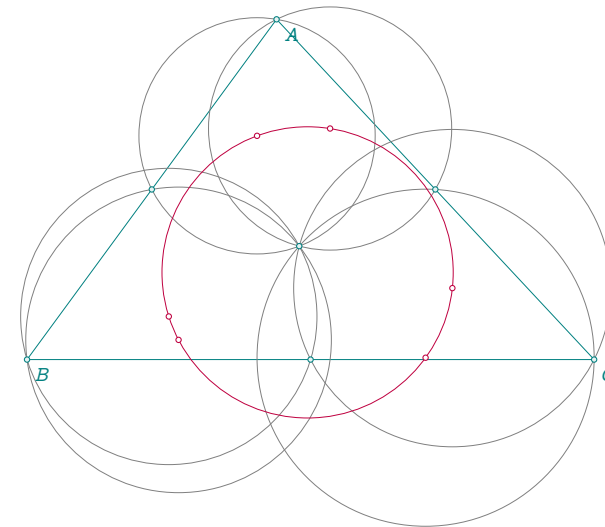
2.63 Van Lamoen's circle

Divide a triangle by its three medians into six smaller triangles. Surprisingly, the circumcenters of the six circumcircles of these smaller triangles are concyclic. Their circumcircle is known as the van Lamoen circle.

Weisstein, Eric W. "van Lamoen Circle." From MathWorld—A Wolfram Web Resource.

File lamoen.lua:

```
init_elements()
z.A = point(2.2, 3)
z.B = point(0, 0)
z.C = point(5, 0)
T.ABC = triangle(z.A, z.B, z.C)
T.med = T.ABC:medial()
z.ma, z.mb, z.mc = T.med:get()
z.G = T.ABC.centroid
z.wab = triangle(z.A, z.mb, z.G).circumcenter
z.wac = triangle(z.A, z.mc, z.G).circumcenter
z.wba = triangle(z.B, z.ma, z.G).circumcenter
z.wbc = triangle(z.B, z.mc, z.G).circumcenter
z.wca = triangle(z.C, z.ma, z.G).circumcenter
z.wcb = triangle(z.C, z.mb, z.G).circumcenter
T.lamoen = triangle(z.wab, z.wac, z.wba)
z.w = T.lamoen.circumcenter
```

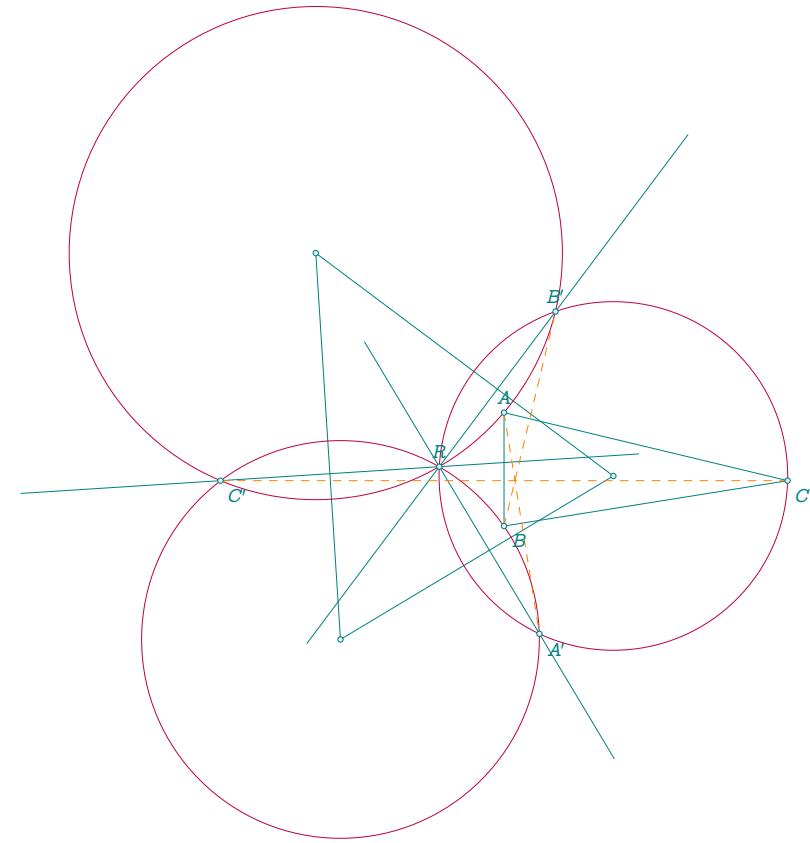


2.64 Yiu's circles variant one

The Yiu circles of a reference triangle is the circles passing through one vertex and the reflections of other vertices with respect to the opposite sides. All three circles have one thing in common: their **radical center**.

File yiu1.lua:

```
init_elements()
z.A = point(0, 2)
z.B = point(0, 0)
z.C = point(5, 0.8)
T.ABC = triangle(z.A, z.B, z.C)
z.Ap = T.ABC.bc:reflection(z.A)
z.Bp = T.ABC.ca:reflection(z.B)
z.Cp = T.ABC.ab:reflection(z.C)
T.A = triangle(z.A, z.Bp, z.Cp)
T.B = triangle(z.B, z.Ap, z.Cp)
T.C = triangle(z.C, z.Ap, z.Bp)
z.O_A = T.A.circumcenter
z.O_B = T.B.circumcenter
z.O_C = T.C.circumcenter
T.O = triangle(z.O_A, z.O_B, z.O_C)
C.A = circle(z.O_A, z.A)
C.B = circle(z.O_B, z.B)
C.C = circle(z.O_C, z.C)
z.R = C.A:radical_center(C.B,C.C)
z.a,z.b = C.A:radical_axis(C.B):get()
z.c,z.d = C.A:radical_axis(C.C):get()
z.e,z.f = C.B:radical_axis(C.C):get()
```



2.65 Yiu's circles variant two

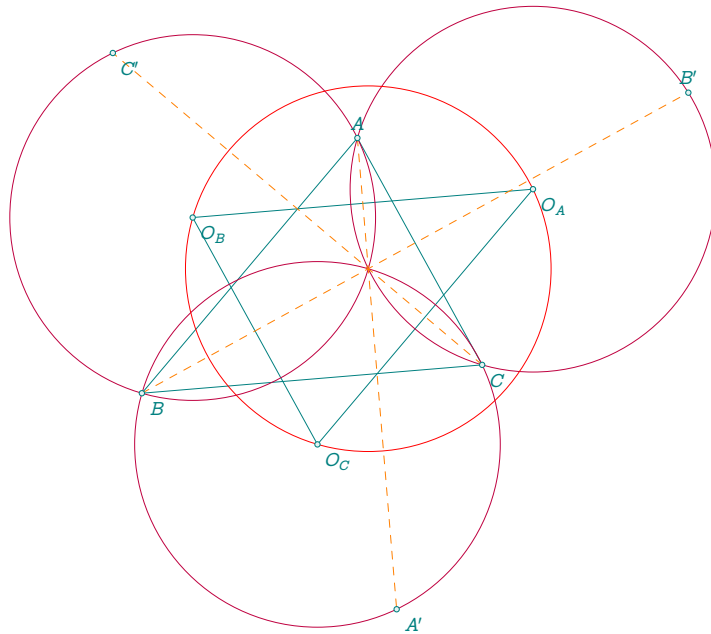
File yiu2.lua:

```
init_elements()
z.A = point(3.8, 4.5)
z.B = point(0, 0)
z.C = point(6, 0.5)
T.ABC = triangle(z.A,z.B,z.C)
z.Ap = T.ABC.bc:reflection(z.A)
z.Bp = T.ABC.ca:reflection(z.B)
z.Cp = T.ABC.ab:reflection(z.C)
T.ABpC = triangle(z.A, z.Bp, z.C)
T.BACp = triangle(z.B, z.A, z.Cp)
```

```

T.CApB = triangle(z.C, z.Ap, z.B)
z.O_A = T.ABpC.circumcenter
z.O_B = T.BACp.circumcenter
z.O_C = T.CApB.circumcenter
z.H = T.ABC.orthocenter

```



2.66 Yff's circles variant one

File yff1.lua:

```

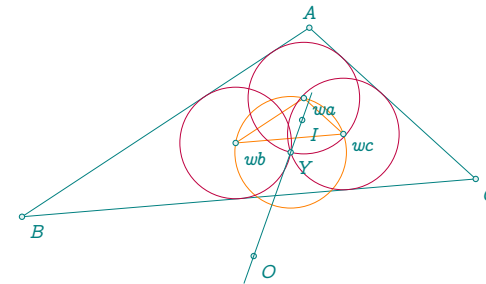
init_elements()
z.A = point(3.8, 2.5)
z.B = point(0, 0)
z.C = point(6, 0.5)
T.ABC = triangle(z.A, z.B, z.C)
z.I = T.ABC.incenter
z.O = T.ABC.circumcenter
z.Y = T.ABC:kimberling(55)
local r = T.ABC.inradius
local R = T.ABC.circumradius
local rho = (r * R) / (R + r)

```

```

C.Y = circle:new(through(z.Y, rho))
z.T = C.Y.through
local bisectors = {
  {z.A, z.I, "wa", T.ABC.ab},
  {z.B, z.I, "wb", T.ABC.bc},
  {z.C, z.I, "wc", T.ABC.ca}
}
for _, bisector in ipairs(bisectors) do
  local origin, incenter, name, side = unpack(bisector)
  local L = line:new(origin, incenter)
  local x, y = intersection(C.Y, L)
  local d = side:distance(x)
  if math.abs(d - C.Y.radius) < tkz.epsilon then
    z[name] = x
  else
    z[name] = y
  end
end
end

```



2.67 Yff's circles variant two

File yff2.lua:

```

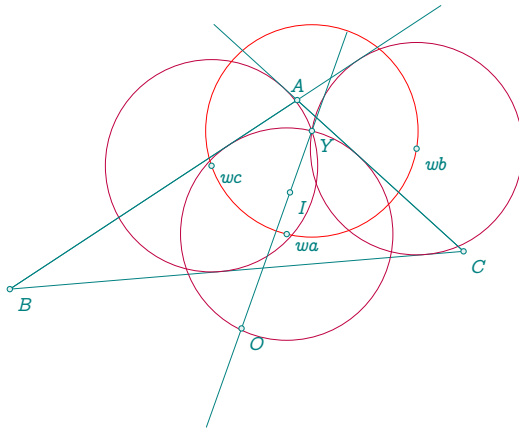
init_elements()
z.A = point(3.8, 2.5)
z.B = point(0, 0)
z.C = point(6, 0.5)
T.ABC = triangle(z.A, z.B, z.C)
z.I = T.ABC.incenter
z.O = T.ABC.circumcenter
z.Y = T.ABC:kimberling(56)
local r = T.ABC.inradius

```

```

local R = tkz.length(z.O, z.A)
local rho = (r * R) / (R - r)
local C = circle(through(z.Y, rho))
z.T = C.through
local bisectors = {
  {z.A, z.I, "wa", T.ABC.ab},
  {z.B, z.I, "wb", T.ABC.bc},
  {z.C, z.I, "wc", T.ABC.ca}
}
}
for _, bisector in ipairs(bisectors) do
  local origin, incenter, name, side = unpack(bisector)
  local L = line(origin, incenter)
  local x, y = intersection(C, L)
  local d = side:distance(x)
  if math.abs(d - C.radius) < tkz.epsilon then
    z[name] = x
  else
    z[name] = y
  end
end
end

```



```

init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.Z = point(4.5, 0)
z.F = point(2.5, 0)
C.A = circle(z.A, z.Z)
C.B = circle(z.B, z.Z)
C.Z = circle(z.Z, z.F)
z.G = z.Z:symmetry(z.F)
z.C = intersection(circle(z.A, z.G), circle(z.B, z.F))
z.Y = intersection(line(z.A, z.C), C.A, {near = z.C})
z.X = intersection(line(z.B, z.C), C.B, {near = z.C})
z.Cp = intersection(line(z.Y, z.X), line(z.A, z.B))
z.Bp = intersection(line(z.Z, z.X), line(z.A, z.C))
z.R = intersection(circle(z.Cp, z.Z), circle(z.C, z.X), {near = z.X})
z.Q = intersection(circle(z.Bp, z.Y), C.B, {near = z.X})
z.wi = intersection(line(z.B, z.Q), line(z.C, z.R))
T.ABC = triangle(z.A, z.B, z.C)
C.ins = T.ABC:in_circle()
z.wi, z.to = C.ins:inversion(circle(z.wi, z.R)):get()

```

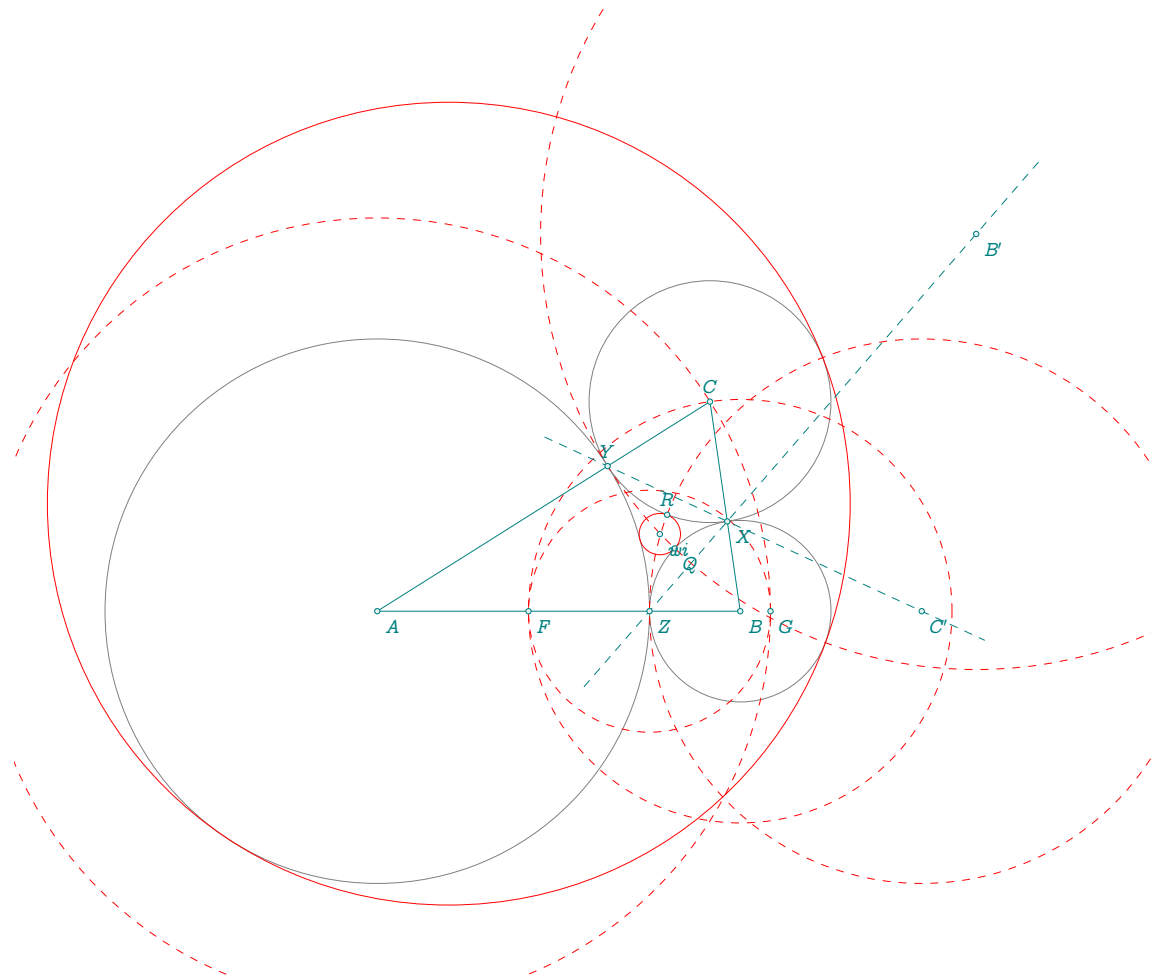
2.68 Construction of Soddy's circles

Proposition 1. Given two mutually tangent circles with centers A and B , and the line (AB) , a generic circle tangent to both circles is constructible in five moves. Articles:

Efficiently Constructing Tangent Circles Arthur Baragar & Alex Kontorovich

Mathematics Magazine

File soddy.lua:



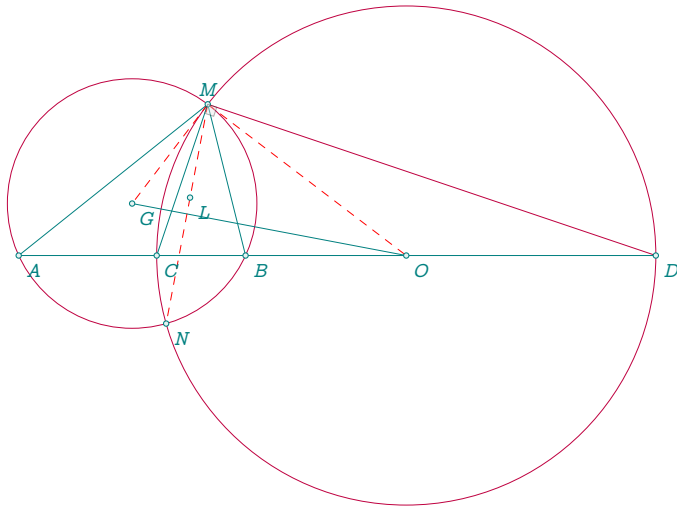
2.69 Radical axis of circumcircle and Apollonius circle

File radical.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.M = point(5, 4)
T.AMB = triangle(z.A, z.M, z.B)
L.AB = T.AMB.ca
z.I = T.AMB.incenter
L.MI = line(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(z.C, z.D)
z.O = L.CD.mid
z.G = T.AMB.circumcenter
C.GA = circle(z.G, z.A)
C.OC = circle(z.O, z.C)
_, z.N = intersection(C.GA, C.OC)
z.L = T.AMB.lemoine_point()

```



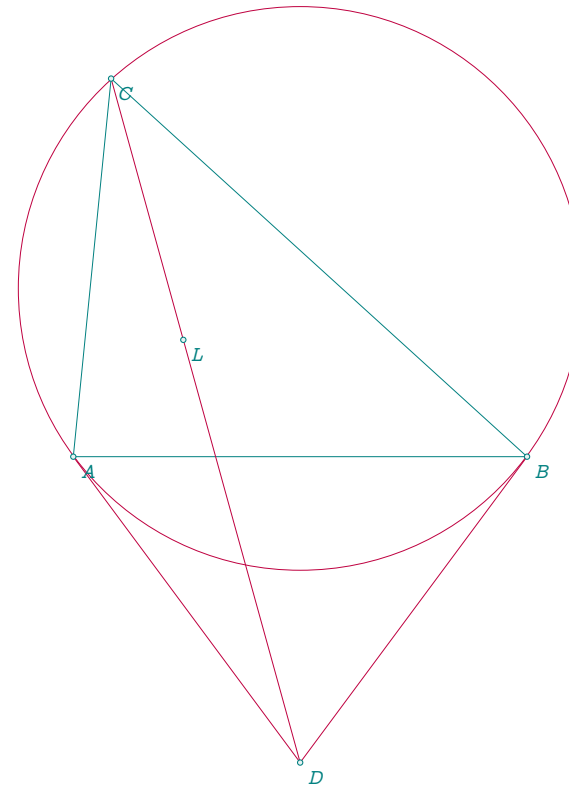
2.70 Symmedian and tangents

File symm_tgts.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(.5, 5)
T.ABC = triangle(z.A, z.B, z.C)
C.cir = T.ABC.circum_circle()
z.O = C.cir.center
L.A = C.cir:tangent_at(z.A)
L.B = C.cir:tangent_at(z.B)
z.D = intersection(L.A, L.B)
z.L = T.ABC.lemoine_point()

```



2.71 Gold Arbelos

File gold_arbelos.lua:

```

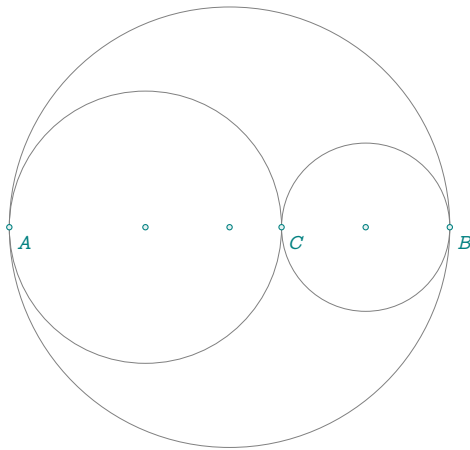
init_elements()
z.A = point(0, 0)
z.C = point(6, 0)
L.AC = line(z.A, z.C)
_, _, z.x, z.y = L.AC:square():get()
z.O_1 = L.AC.mid
C.one = circle(z.O_1, z.x)
z.B = intersection(L.AC, C.one)
L.CB = line(z.C, z.B)
z.O_2 = L.CB.mid
L.AB = line(z.A, z.B)
z.O_0 = L.AB.mid

```

```

z.B = point(5, 2)
z.C = point(1.2, 2)
T.ABC = triangle(z.A, z.B, z.C)
z.O = T.ABC.circumcenter
C.OA = circle(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.ABC.bc)
z.Q = intersection(L.tB, T.ABC.ca)
z.R = intersection(L.tC, T.ABC.ab)

```



2.72 Lemoine

Theorem 39: Lemoine_three tangents

Let ABC be a triangle and let $\mathcal{C}(ABC)$ denote its circumcircle. Let T_A , T_B , and T_C be the tangents to $\mathcal{C}(ABC)$ at A , B , and C , respectively.

Assume that the lines BA and CA , extended beyond A , meet T_C and T_B at P and Q , respectively. Assume also that T_A meets T_C at U and T_B at V .

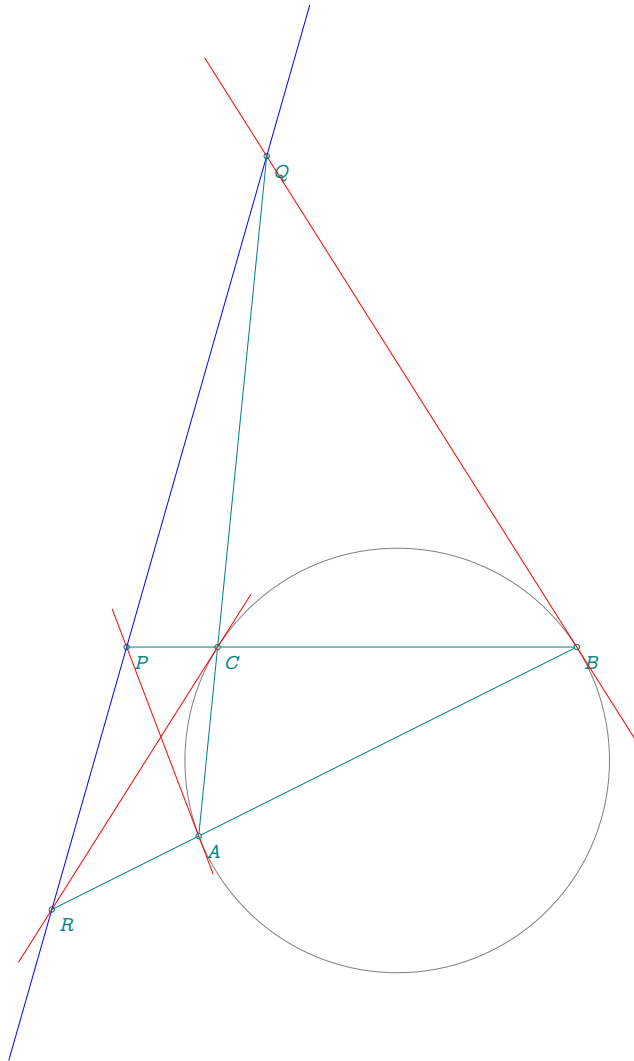
Then the three lines T_A , UV , and PQ are either concurrent or parallel.

File lemoine.lua

```

init_elements()
z.A = point(1, 0)

```



2.73 Lemoine point

Definition 12: Lemoine point

In Euclidean geometry, the Lemoine point of a triangle ABC —also known as the symmedian point or Grebe point—is the common intersection of the three symmedians, i.e., the reflections of the medians across the corresponding internal angle bisectors. Equivalently, it is the isogonal conjugate of the centroid of the triangle.

1. Let L be the Lemoine point, and let L_c be the foot of the symmedian issued from C . Then

$$\frac{CL}{CL_c} = \frac{a^2 + b^2}{a^2 + b^2 + c^2}.$$

2. In Kimberling's Encyclopedia of Triangle Centers, the Lemoine point is labelled X(6).
3. It is the isogonal conjugate of the centroid (center of gravity).
4. It is the unique point that minimizes the sum of the squares of the distances to the three sides of the triangle.
5. It is the barycentre of the weighted points $(A, a^2), (B, b^2), (C, c^2)$.
6. Its barycentric coordinates are proportional to

$$a^2 : b^2 : c^2.$$

File symmedian.lua:

```

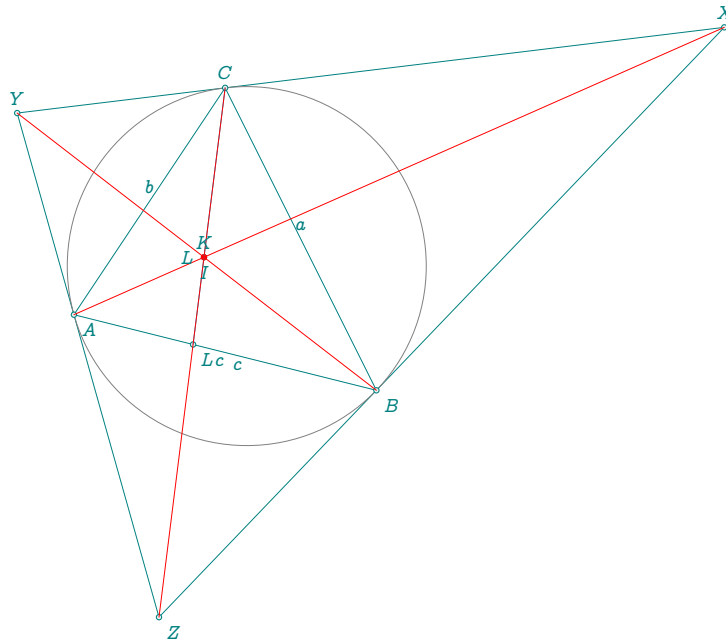
init_elements()
z.A = point(1, 2)
z.B = point(5, 1)
z.C = point(3, 5)
T.ABC = triangle(z.A, z.B, z.C)
T.tgt = T.ABC:tangential()
z.X,z.Y,z.Z = T.tgt:get()
z.O = T.ABC:circumcenter
T.SY = T.ABC:symmedian()
z.La, z.Lb, z.Lc = T.SY:get()
k = (T.ABC.a * T.ABC.a + T.ABC.b * T.ABC.b)
  / (T.ABC.a * T.ABC.a + T.ABC.b * T.ABC.b +
    T.ABC.c * T.ABC.c)
L.SY = line(z.C, z.Lc)
z.L = L.SY:point(k)

```

```

z.K = T.ABC:kimberling(6)
z.G = T.ABC.centroid
z.I = T.ABC:isogonal(z.G)

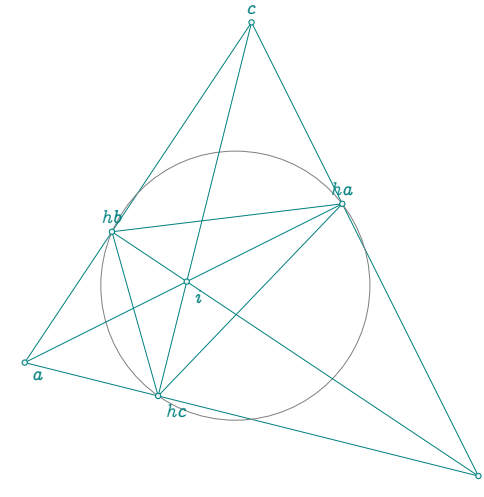
```



```

z.b = point(5, 1)
z.c = point(3, 5)
T.abc = triangle(z.a, z.b, z.c)
z.i = T.abc.orthocenter
T.cevian = T.abc:cevian(z.i)
z.ha, z.hb, z.hc = T.cevian:get()
C.cev = T.abc:cevian_circle(z.i)
z.w = C.cev.center

```



2.74 Example: Cevian circle with orthocenter

Definition 13: Cevian circle

Given a triangle ABC and a point P , the Cevian circle of P is the circumcircle of the Cevian triangle of P , that is, the triangle formed by the three intersections of the cevians AP , BP , and CP with the opposite sides.

Definition 14: Special case

If P is the orthocenter of ABC , then the Cevian circle of P coincides with the nine-point circle.

File `cevian.lua`:

```

init_elements()
z.a = point(1, 2)

```

2.75 Excircles

Definition 15: Excircles

An excircle (or escribed circle) of a triangle is a circle lying outside the triangle, tangent to one of its sides and tangent to the extensions of the other two sides. Every triangle has three distinct excircles, each tangent to a different side of the triangle.

File `excircles.lua`:

```

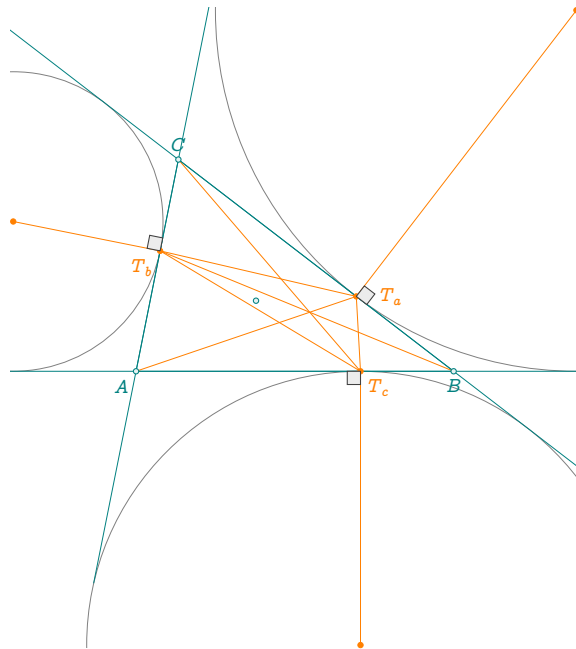
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(0.8, 4)
T.ABC = triangle(z.A, z.B, z.C)
z.K = T.ABC.centroid
z.J_a, z.J_b, z.J_c = T.ABC:excentral():get()

```

```

z.T_a, z.T_b, z.T_c = T.ABC:extouch():get()
la = line(z.A, z.T_a)
lb = line(z.B, z.T_b)
z.G = intersection(la, lb)

```



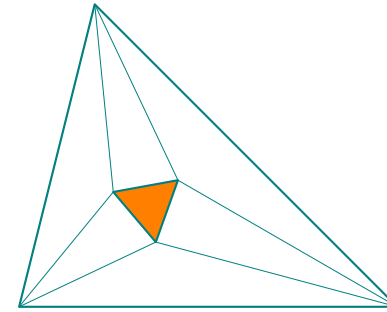
2.76 Morley triangle

File morley.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
z.C = point(1, 4)
T.ABC = triangle(z.A, z.B, z.C)
z.a1, z.a2 = tkz.trisector(z.A, z.B, z.C)
z.b1, z.b2 = tkz.trisector(z.B, z.C, z.A)
z.c1, z.c2 = tkz.trisector(z.C, z.A, z.B)
z.D = intersection_ll_(z.A, z.a1, z.B, z.b2)
z.E = intersection_ll_(z.B, z.b1, z.C, z.c2)
z.F = intersection_ll_(z.C, z.c1, z.A, z.a2)

```



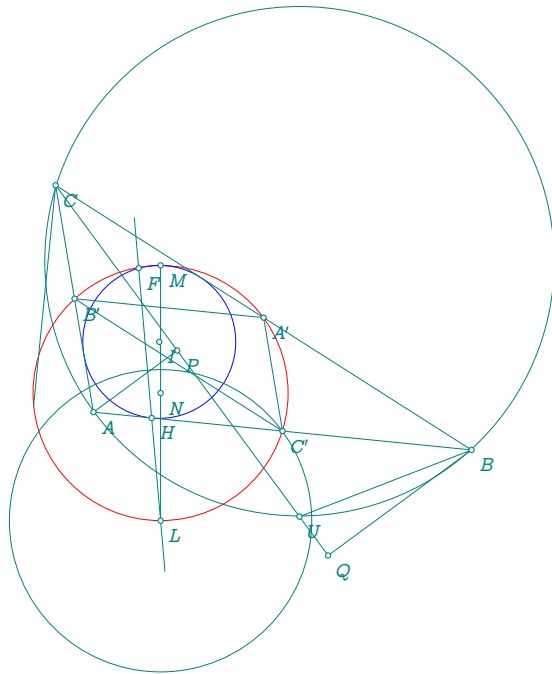
2.77 Feuerbach theorem

File feuerbach.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(5, -0.5)
z.C = point(-0.5, 3)
T.ABC = triangle(z.A, z.B, z.C)
z.O = T.ABC.circumcenter
z.N = T.ABC.eulercenter
z.I, z.K = T.ABC.in_circle():get()
z.H = T.ABC.ab:projection(z.I)
z.Ap, z.Bp, z.Cp = T.ABC.medial():get()
C.IH = circle(z.I, z.H)
C.NAp = circle(z.N, z.Ap)
C.OA = circle(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(z.C, z.U)
L.ML = line(z.M, z.L)
z.P = L.CU:projection(z.A)
z.Q = L.CU:projection(z.B)
L.LH = line(z.L, z.H)
z.F = intersection(L.LH, C.IH)

```



2.78 Orthopole and Simson line

File orthopole.lua:

```

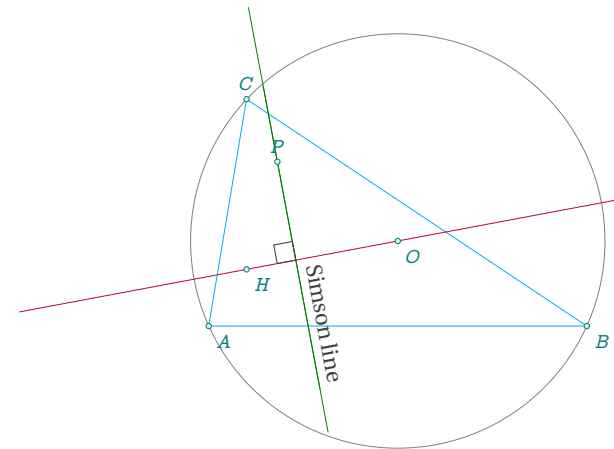
init_elements()
z.A = point:new(0, 0)
z.B = point:new(5, 0)
z.C = point:new(0.5, 3)
T.ABC = triangle:new(z.A, z.B, z.C)
z.O = T.ABC.circumcenter
C.circum = circle(z.O, z.A)
z.H = T.ABC.orthocenter
L.OH = line(z.O, z.H)
z.P = T.ABC.orthopole(L.OH)
L.ortho = L.OH:orthogonal_from(z.P)
z.p = L.ortho.pb
z.Q = intersection(T.ABC.ab, L.ortho)
L.perp = T.ABC.ab:orthogonal_from(z.Q)
z.q = L.perp.pb

```

```

z.x, z.y = intersection(C.circum, L.perp)
L.simson = T.ABC:simson_line(z.x)
z.sa, z.sb = L.simson:get()
z.h = intersection(L.simson, L.OH)

```



2.79 Gold Arbelos properties

File gap.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(10, 0)
L.AB = line(z.A, z.B)
z.C = L.AB:gold_ratio()
z.O_1 = L.AB.mid
L.AC = line(z.A, z.C)
z.O_2 = L.AC.mid
L.CB = line(z.C, z.B)
z.O_3 = L.CB.mid
C.one = circle(z.O_1, z.B)
C.two = circle(z.O_2, z.C)
C.three = circle(z.O_3, z.B)
z.Q = C.two.north
z.P = C.three.north
L.O23 = line(z.O_2, z.O_3)
z.M_0 = L.O23:harmonic_ext(z.C)
L.O12 = line(z.O_1, z.O_2)

```

```

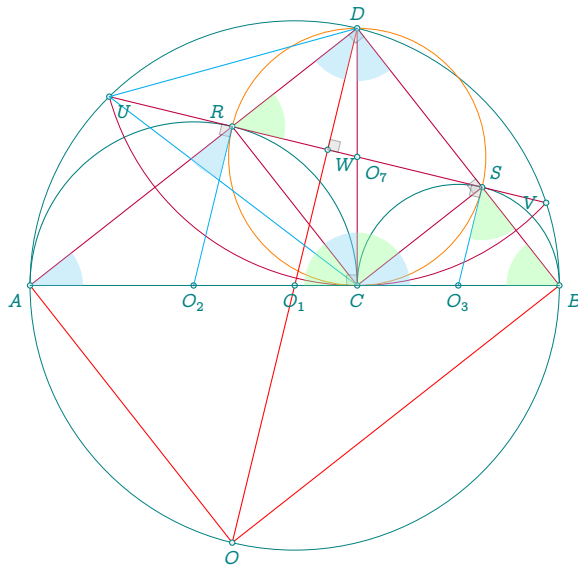
z.M_1 = L.O12:harmonic_int(z.A)
L.O13 = line(z.O_1, z.O_3)
z.M_2 = L.O13:harmonic_int(z.B)
L.bq = line(z.B, z.Q)
L.ap = line(z.A, z.P)
z.S = intersection(L.bq, L.ap)
z.x = z.C:north()
L.Cx = line(z.C, z.x)
z.D, _ = intersection(L.Cx, C.one)
L.CD = line(z.C, z.D)
z.O_7 = L.CD.mid
C.DC = circle(z.D, z.C)
z.U, z.V = intersection(C.DC, C.one)
L.UV = line(z.U, z.V)
z.R, z.S = L.UV:projection(z.O_2, z.O_3)
L.O1D = line(z.O_1, z.D)
z.W = intersection(L.UV, L.O1D)
z.O = C.DC:inversion(z.W)

```

```

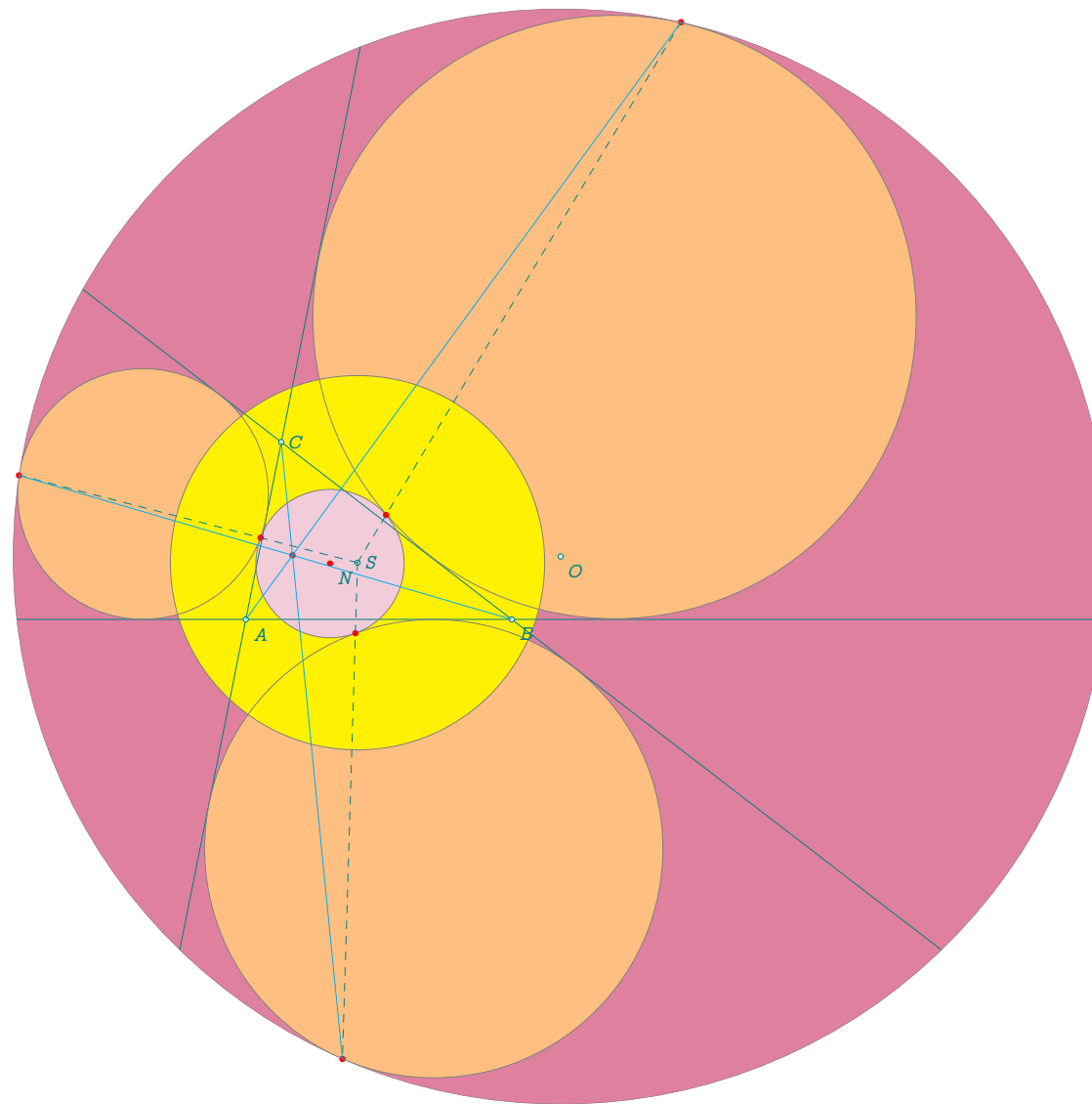
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(0.8, 4)
T.ABC = triangle(z.A, z.B, z.C)
z.N = T.ABC.eulercenter
z.Ea, z.Eb, z.Ec = T.ABC:feuerbach():get()
z.Ja, z.Jb, z.Jc = T.ABC:excentral():get()
z.S = T.ABC:spieker_center()
C.JaEa = circle(z.Ja, z.Ea)
C.ortho = C.JaEa:orthogonal_from(z.S)
z.a = C.ortho.south
C.euler = T.ABC:euler_circle()
z.O = C.ortho:inversion(C.euler).center
z.xa, z.xb, z.xc = C.ortho:set_inversion(z.Ea, z.Eb, z.Ec)
z.apo = T.ABC:kimberling(181)
z.W, z.T = T.ABC:feuerbach_apollonius_k181():get()

```



2.80 Apollonius circle v1 with inversion

File apo_inv.lua:



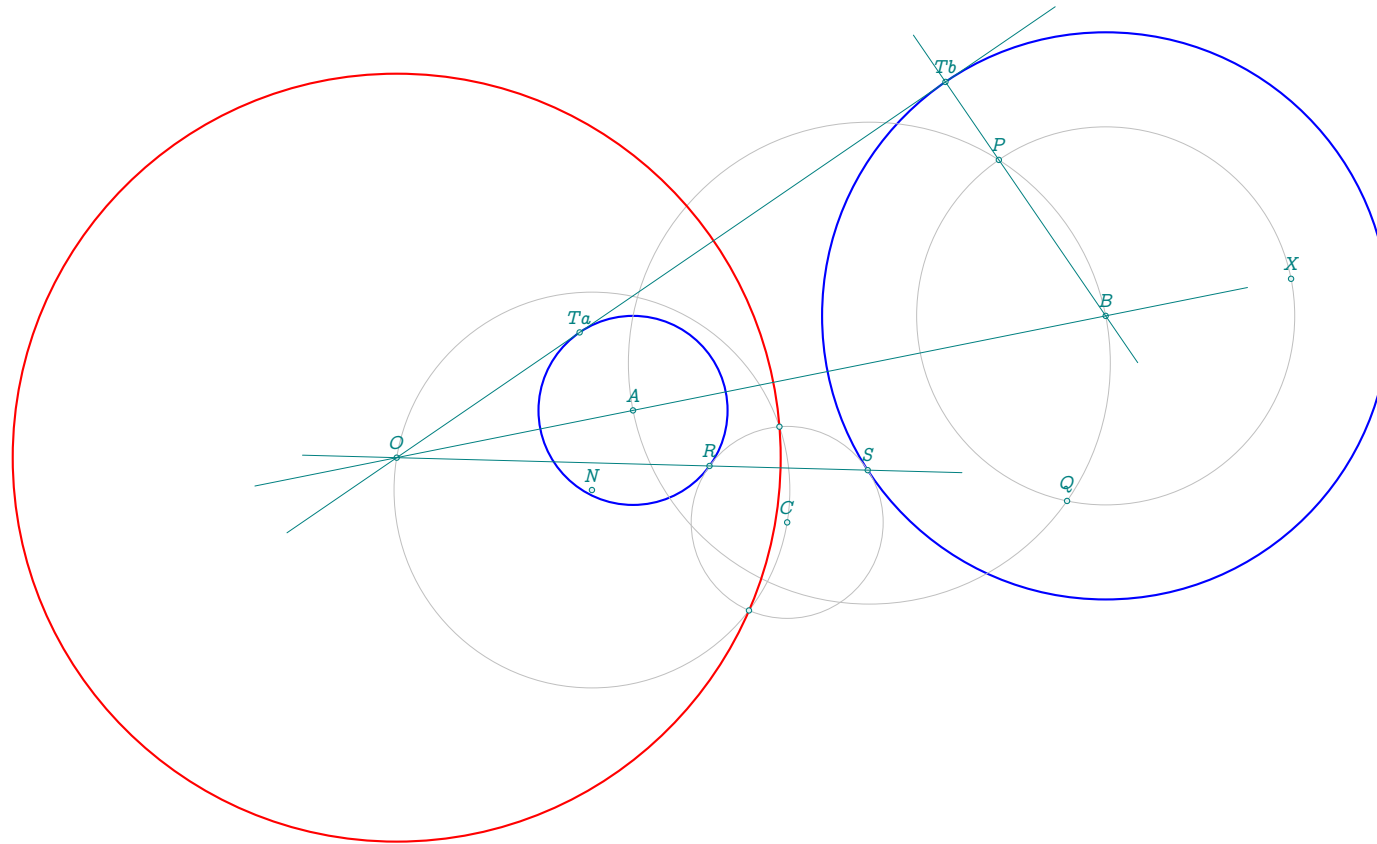
2.81 Construction of the midcircle of two disjoint circles

File mid_circle.lua:

```

init_elements()
z.A = point(0, 0)
z.a = point(1, 0)
z.B = point(5, 1)
z.b = point(8, 1)
C.A = circle(z.A, z.a)
C.B = circle(z.B, z.b)
L.common = C.A:common_tangent(C.B)
z.Ta, z.Tb = L.common:get()
z.O = C.A:external_similitude(C.B)
L.AB = line(z.A, z.B)
ra = C.A.radius
rb = C.B.radius
z.X = L.AB:report(rb-ra, z.B)
z.M = L.AB.mid
C.MA = circle(z.M, z.A)
C.BX = circle(z.B, z.X)
z.P, z.Q = intersection(C.MA, C.BX)
z.R = C.A:point(-0.1)
L.OR = line(z.O, z.R)
z.S = intersection(C.B, L.OR, {near=z.R})
L.AR = line(z.A, z.R)
L.BS = line(z.B, z.S)
z.C = intersection(L.AR, L.BS)
z.N = tkz.midpoint(z.O, z.C)
C.NO = circle(z.N, z.O)
C.CR = circle(z.C, z.R)
z.ta, z.tb = intersection(C.NO, C.CR)

```



2.82 Midcircles and Arbelos

File mid_circle_arbelos.lua:

```

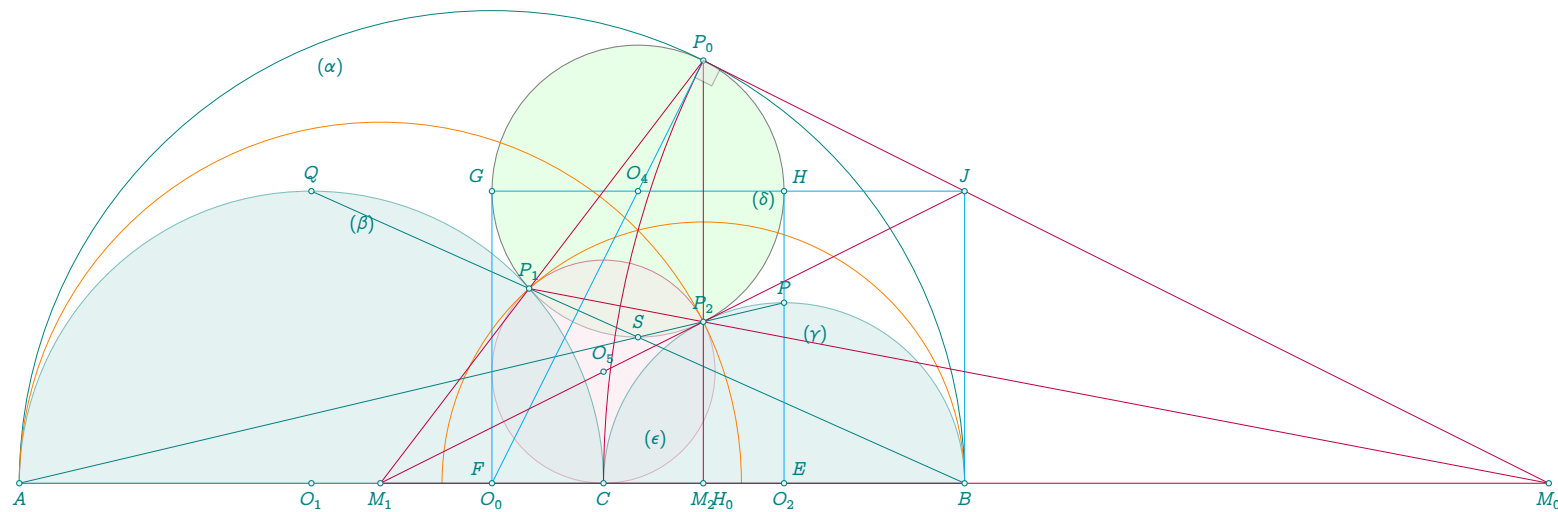
init_elements()
z.A = point(0, 0)
z.B = point(10, 0)
L.AB = line(z.A, z.B)
z.C = L.AB:gold_ratio()
L.AC = line(z.A, z.C)
L.CB = line(z.C, z.B)
z.O_0 = L.AB.mid
z.O_1 = L.AC.mid
z.O_2 = L.CB.mid
C.O0B = circle(z.O_0, z.B)
C.O1C = circle(z.O_1, z.C)
C.O2C = circle(z.O_2, z.B)
z.Q = C.O1C:midarc(z.C, z.A)
z.P = C.O2C:midarc(z.B, z.C)
L.O1O2 = line(z.O_1, z.O_2)
L.O0O1 = line(z.O_0, z.O_1)
L.O0O2 = line(z.O_0, z.O_2)
z.M_0 = L.O1O2:harmonic_ext(z.C)
z.M_1 = L.O0O1:harmonic_int(z.A)
z.M_2 = L.O0O2:harmonic_int(z.B)
L.BQ = line(z.B, z.Q)
L.AP = line(z.A, z.P)
z.S = intersection(L.BQ, L.AP)
L.CS = line(z.C, z.S)
C.M1A = circle(z.M_1, z.A)
C.M2B = circle(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.P012 = triangle(z.P_0, z.P_1, z.P_2)
z.O_4 = T.P012.circumcenter
T.CP12 = triangle(z.C, z.P_1, z.P_2)
z.O_5 = T.CP12.circumcenter
z.BN = z.B:north()
L.BBN = line(z.B, z.BN)
L.M1P2 = line(z.M_1, z.P_2)
z.J = intersection(L.BBN, L.M1P2)

```

```

L.AP0 = line(z.A, z.P_0)
L.BP0 = line(z.B, z.P_0)
C.O4P0 = circle(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)

```



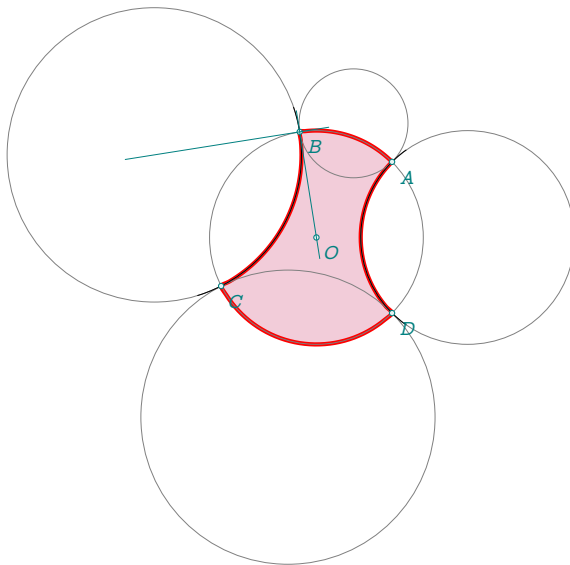
2.83 Circle and path

File path_circle.lua:

```

init_elements()
z.O = point(0, 0)
z.A = point(1, 1)
C.OA = circle(z.O, z.A)
z.B = C.OA:point(0.15)
z.C = C.OA:point(0.45)
z.D = C.OA:point(0.75)
C.RBC = C.OA:orthogonal_through(z.B, z.C)
z.R = C.RBC.center
C.SDA = C.OA:orthogonal_through(z.D, z.A)
z.S = C.SDA.center
C.TAB = C.OA:orthogonal_through(z.B, z.A)
z.T = C.TAB.center
C.UCD = C.OA:orthogonal_through(z.C, z.D)
z.U = C.UCD.center
PA.AB = C.OA:path(z.A, z.B, 20)
PA.CD = C.OA:path(z.C, z.D, 20)
PA.CB = C.RBC:path(z.C, z.B, 20)
PA.AD = C.SDA:path(z.A, z.D, 20)
PA.zone = PA.AB - PA.CB + PA.CD - PA.AD

```



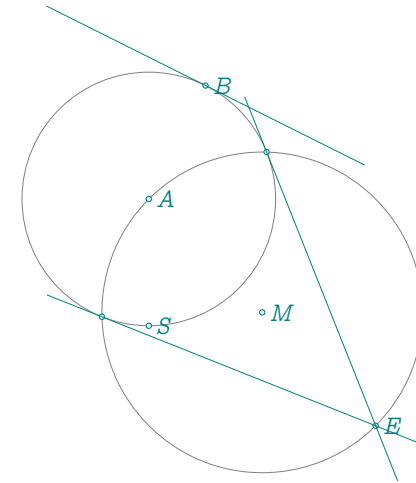
2.84 Tangent and circle

File tangent.lua:

```

init_elements()
z.A = point(1, 0)
z.B = point(2, 2)
z.E = point(5, -4)
L.AE = line(z.A, z.E)
C.AB = circle(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 = C.AB:tangent_at(z.B):get()

```



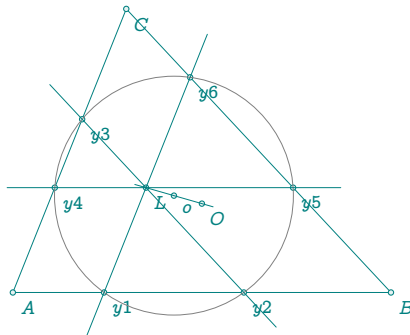
2.85 First Lemoine circle

Draw lines through the symmedian point L and parallel to the sides of the triangle. The points where the parallel lines intersect the sides of the triangle then lie on a circle known as the first Lemoine circle. It has center at the Brocard midpoint, i.e., the midpoint of $[OL]$, where O is the circumcenter and K is the symmedian point

Weisstein, Eric W. "First Lemoine Circle." From MathWorld—A Wolfram Web Resource.

File `first_lemoine.lua`:

```
init_elements()
z.A = point(1, 1)
z.B = point(5, 1)
z.C = point(2.2, 4)
T.ABC = triangle(z.A, z.B, z.C)
z.O = T.ABC.circumcenter
C.first_lemoine = T.ABC:first_lemoine_circle()
z.o, z.w = C.first_lemoine:get()
z.Ar, z.Al = C.first_lemoine:tangent_at(z.A):get()
z.Br, z.Bl = C.first_lemoine:tangent_at(z.B):get()
z.Cr, z.Cl = C.first_lemoine:tangent_at(z.C):get()
z.y1, z.y2 = intersection(T.ABC.ab, C.first_lemoine)
z.y5, z.y6 = intersection(T.ABC.bc, C.first_lemoine)
z.y3, z.y4 = intersection(T.ABC.ca, C.first_lemoine)
z.L = T.ABC:lemoine_point()
```



2.86 First and second Lemoine circles

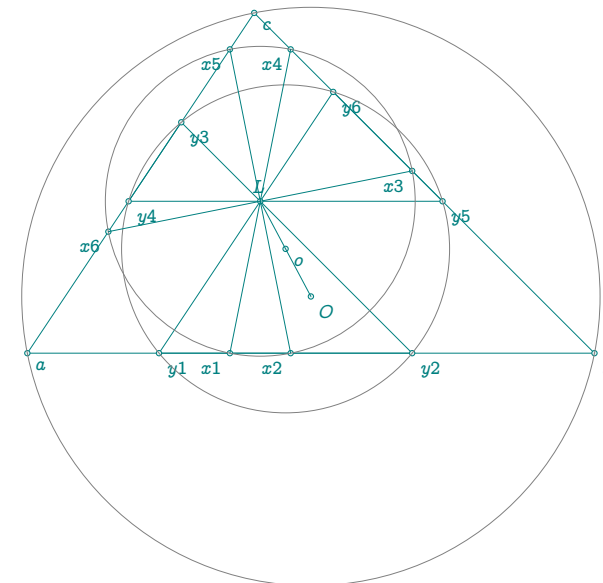
Draw antiparallels through the symmedian point L . The points where these lines intersect the sides then lie on a circle, known as the cosine circle (or sometimes the second Lemoine circle).

Weisstein, Eric W. "Cosine Circle." From MathWorld—A Wolfram Web Resource.

File `second_lemoine.lua`:

```
init_elements()
z.a = point(0, 0)
z.b = point(5, 0)
```

```
z.c = point(2, 3)
T.abc = triangle(z.a, z.b, z.c)
z.O = T.abc.circumcenter
z.o, z.p = T.abc:first_lemoine_circle():get()
L.ab = line(z.a, z.b)
L.ca = line(z.c, z.a)
L.bc = line(z.b, z.c)
z.L, z.x = T.abc:second_lemoine_circle():get()
C.first_lemoine = circle(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(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(z.y1, z.y6)
L.y4y5 = line(z.y4, z.y5)
L.y2y3 = line(z.y2, z.y3)
```



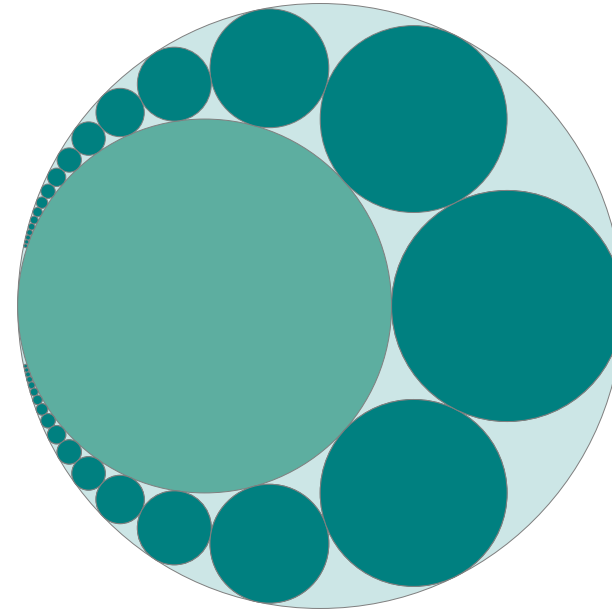
2.87 Pappus chain

Definition 16: Pappus chain

Starting with the circle P_1 tangent to the three semicircles forming the arbelos, construct a chain of tangent circles P_i , all tangent to one of the two small interior circles and to the large exterior one. This chain is called the Pappus chain

File pappus.lua:

```
init_elements()
xC, nc = 10, 16
xB = xC * tkz.invpphi
xD = (xC * xC) / xB
xJ = (xC + xD) / 2
r = xD - xJ
z.A = point(0, 0)
z.B = point(xB, 0)
z.C = point(xC, 0)
L.AC = line(z.A, z.C)
z.i = L.AC.mid
L.AB = line(z.A, z.B)
z.j = L.AB.mid
z.D = point(xD, 0)
C.AC = circle(z.A, z.C)
for i = -nc, nc do
  z["J" .. i] = point(xJ, 2 * r * i)
  z["H" .. i] = point(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(z.A, z["J" .. i])
  C.JH = circle(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(z["S" .. i .. "p"], z["T" .. i .. "p"])
  z["I" .. i] = L.SpTp.mid
end
```

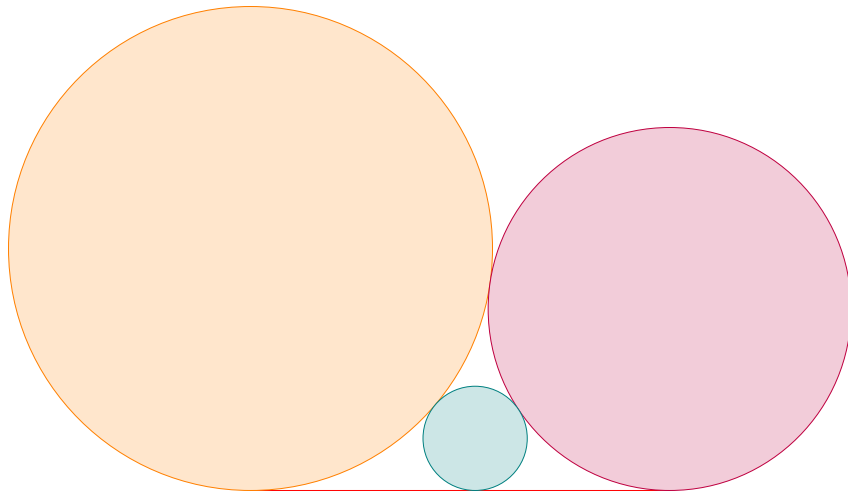


2.88 Three Circles

Construct a circle tangent to two circles tangent to each other and to a straight line.

File three_circles.lua:

```
init_elements()
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(2 * xk, r2)
  z[h2] = point(2 * xk, 0)
  z[c1] = point(0, r1)
  z[h1] = point(0, 0)
  L.h1h2 = line(z[h1], z[h2])
  z[c3] = point(cx, cy)
  z[h3] = L.h1h2:projection(z[c3])
end
threecircles("A", 4, "B", 3, "C", "E", "G", "F")
```



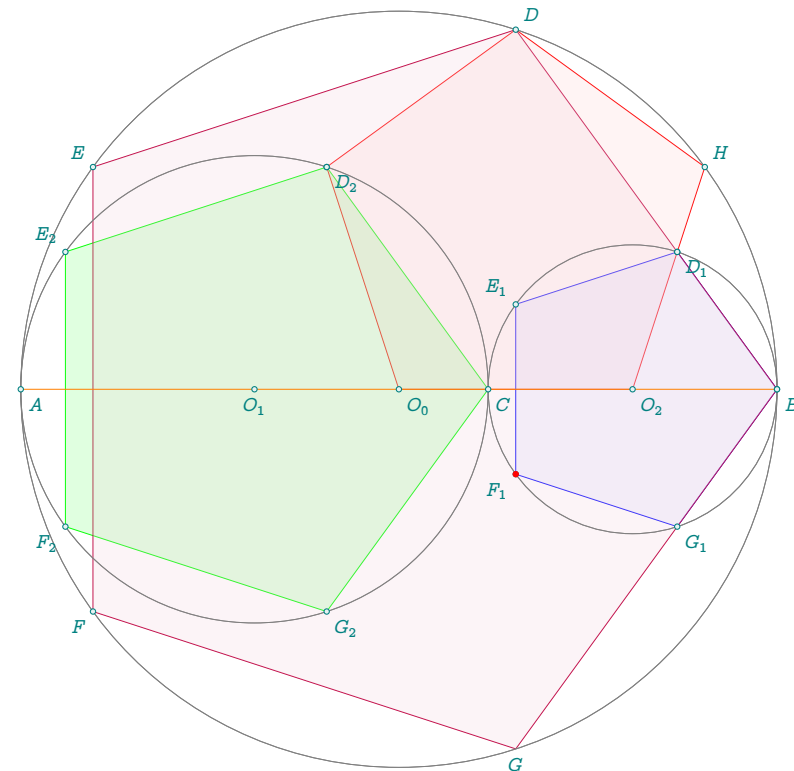
2.89 Pentagons in a golden arbelos

HIROSHI OKUMURA: a note on regular pentagons arising from the golden arbelos.

File pentagons.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(10, 0)
L.AB = line(z.A, z.B)
z.C = L.AB:gold_ratio()
L.AC = line(z.A, z.C)
L.CB = line(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(z.O_0, z.B)
C.O1C = circle(z.O_1, z.C)
C.O2B = circle(z.O_2, z.B)
z.M_0 = C.O1C:external_similitude(C.O2B)
L.O0C = line(z.O_0, z.C)
T.golden = L.O0C:golden()
z.L = T.golden.pc
L.O0L = line(z.O_0, z.L)
z.D = intersection(L.O0L, C.O0B)
L.DB = line(z.D, z.B)
```

```
_, z.Z = intersection(L.DB, C.O2B)
L.DA = line(z.D, z.A)
z.I = intersection(L.DA, C.O1C)
L.O2Z = line(z.O_2, z.Z)
z.H = intersection(L.O2Z, C.O0B)
C.BD = circle(z.B, z.D)
C.DB = circle(z.D, z.B)
_, z.G = intersection(C.BD, C.O0B)
z.E = intersection(C.DB, C.O0B)
C.GB = circle(z.G, z.B)
_, z.F = intersection(C.GB, C.O0B)
k = 1 / tkz.phi ^ 2
kk = tkz.phi
z.D_1, z.E_1, z.F_1, z.G_1 = z.B:homothety(k, z.D, z.E, z.F, z.G)
z.D_2, z.E_2, z.F_2, z.G_2 = z.M_0:homothety(kk, z.D_1, z.E_1, z.F_1, z.G_1)
```



2.90 Polar and Pascal's theorem

Example 5: Polar and Pascal's theorem

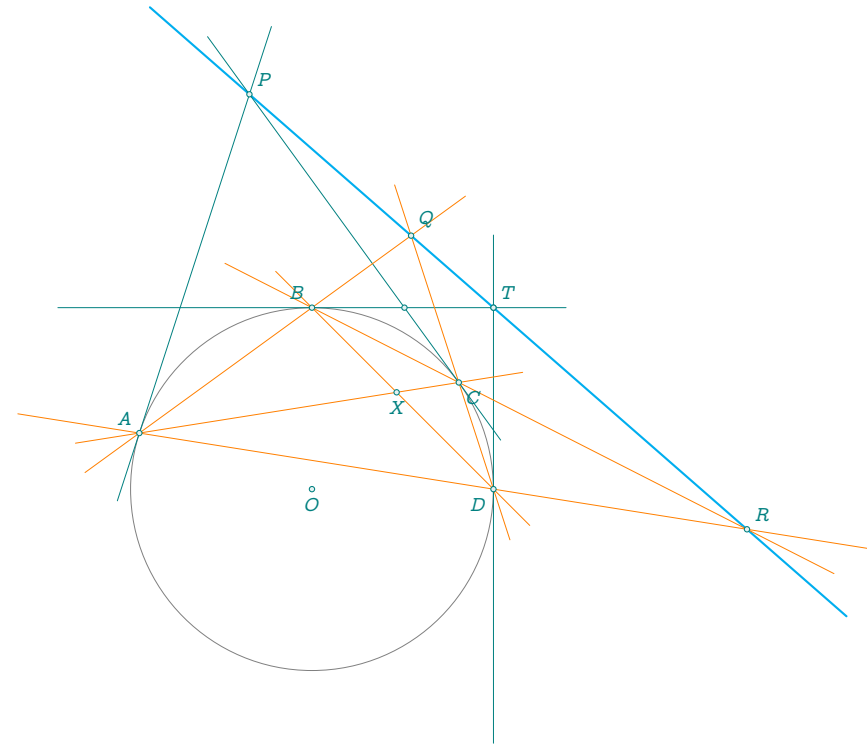
Given a cyclic quadrilateral $ABCD$ if the intersection of (AC) and (BD) is P , the intersection of (AB) and (CD) is Q and the intersection of (AD) and (BC) is R , prove the polar of P with respect of $\mathcal{C}(O, A)$ passes through Q and R .

File polar.lua:

```

init_elements()
z.O = point(0, 0)
z.D = point(3, 0)
C.OD = circle(z.O, z.D)
z.B = C.OD:point(0.25)
z.A = C.OD:point(0.45)
z.C = C.OD:point(0.10)
L.AC = line(z.A, z.C)
L.AB = line(z.A, z.B)
L.CD = line(z.C, z.D)
L.AD = line(z.A, z.D)
L.BD = line(z.B, z.D)
L.BC = line(z.B, z.C)
z.X = intersection(L.AC, L.BD)
z.Q = intersection(L.AB, L.CD)
z.R = intersection(L.AD, L.BC)
L.QR = line(z.Q, z.R)
L.Ta = C.OD:tangent_at(z.A)
L.Tb = C.OD:tangent_at(z.B)
L.Tc = C.OD:tangent_at(z.C)
L.Td = C.OD:tangent_at(z.D)
z.Ax, z.Ay = L.Ta:get()
z.Bx, z.By = L.Tb:get()
z.Cx, z.Cy = L.Tc:get()
z.Dx, z.Dy = L.Td:get()
z.Ibd = intersection(L.Tb, L.Td)
z.P = intersection(L.Ta, L.Tc)
z.T = intersection(L.Tb, L.Td)
z.Ibc = intersection(L.Tb, L.Tc)

```



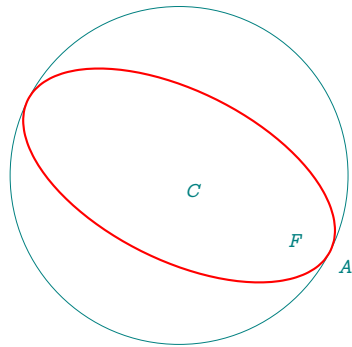
2.91 Ellipse

File ellipse.lua:

```

init_elements()
z.C = point(3, 2)
z.A = point(5, 1)
L.CA = line(z.C, z.A)
z.b = L.CA:north_pa
L.Cb = line(z.C, z.b)
z.B = L.Cb:point(0.5)
CO.EL = conic(EL_points(z.C, z.A, z.B))
PA.E = CO.EL:points(0, 1, 50)
z.F = CO.EL.Fa

```



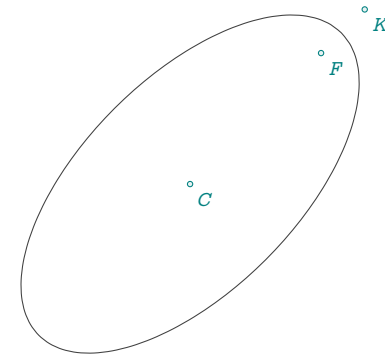
2.92 Ellipse with radii

Example 6: Ellipse with radii

In this example, K is the projection of the focus F on the directrix.

File radii.lua:

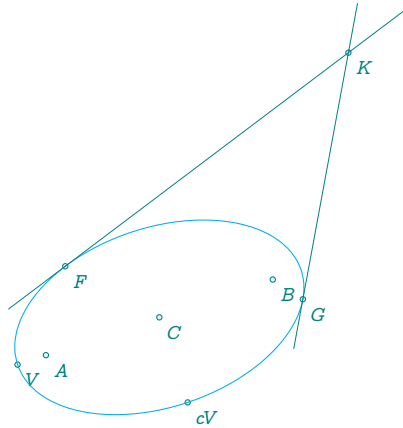
```
init_elements()
z.C = point(0, 4)
local b = math.sqrt(8)
local a = math.sqrt(32)
local c = math.sqrt(a ^ 2 - b ^ 2)
local e = c / a
z.F = z.C:rotation(math.pi / 4, z.C + point(c, 0))
z.K = z.C:rotation(math.pi / 4, z.C + point(a ^ 2 / c, 0))
z.Kp = (z.K - z.C):orthogonal(1):at(z.K)
L.dir = line(z.K, z.Kp)
CO.EL = conic(z.F, L.dir, e)
PA.curve = CO.EL:points(0, 1, 50)
```



2.93 Ellipse_with_foci

File focii.lua:

```
init_elements()
local e = 0.8
z.A = point(2, 3)
z.B = point(5, 4)
z.K = point(6, 7)
L.AB = line(z.A, z.B)
z.C = L.AB.mid
local c = point.abs(z.B - z.C)
local a = c / e
CO.EL = conic(EL_bifocal(z.A, z.B, a))
PA.curve = CO.EL:points(0, 1, 50)
z.cV = CO.EL.covertex
z.V = CO.EL.vertex
L.ta, L.tb = CO.EL:tangent_from(z.K)
z.F = L.ta.pb
z.G = L.tb.pb
```

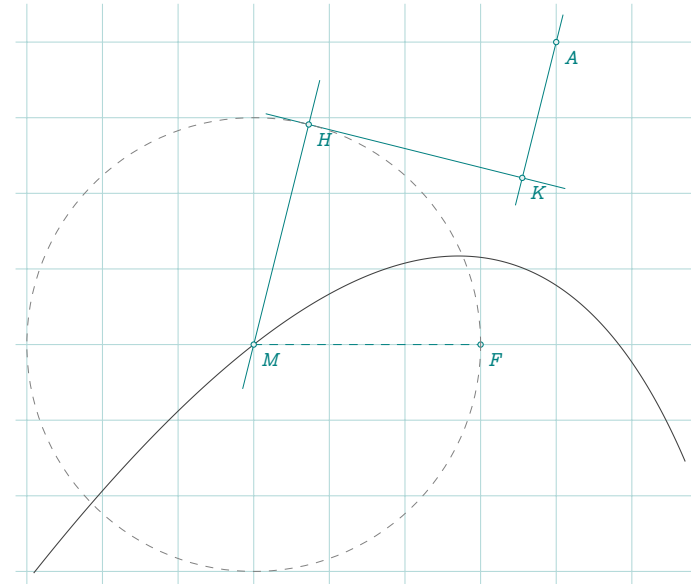


2.94 Parabola with Focus, Axis of Symmetry, and a Point on the Curve

This construction illustrates how to define a parabola from its focus, its axis of symmetry, and a single point on the curve. The axis is determined as the line orthogonal to the segment joining the focus to the given point, passing through a specific intersection derived from a circle construction. This method is particularly useful when the directrix is not known explicitly.

File `parabola1.lua`:

```
init_elements()
z.F = point(2, 0)
z.A = point(3, 4)
L.FA = line(z.A, z.F)
z.M = point(-1, 0)
C.MF = circle(z.M, z.F)
L.l1 = L.FA:ll_from(z.M)
z.H = intersection(C.MF, L.l1)
L.dir = L.FA:ortho_from(z.H)
z.K = intersection(L.dir, L.FA)
CO.PA = conic(z.F, L.dir, 1)
PA.curve = CO.PA:points(-5, 3, 20)
```



2.95 Ellipse with Center, Vertex, and Covertex

This construction defines an ellipse from its center, a vertex, and a covertex. From these three points, the semi-axes lengths a and b are computed, allowing the eccentricity e to be derived. The foci and the direction of the directrix are then constructed accordingly, making it possible to define the ellipse using the `conic` method.

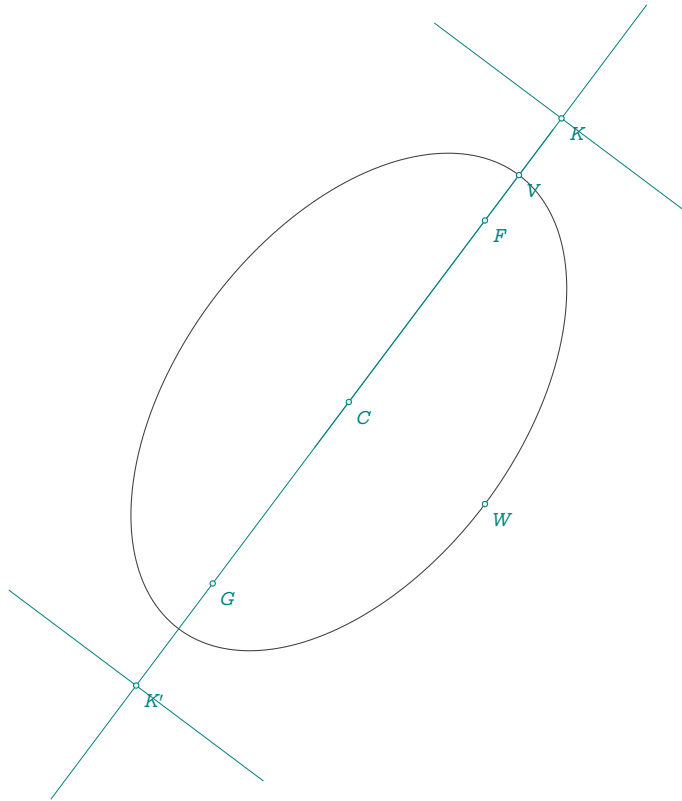
File `ellipse2.lua`:

```
init_elements()
z.C = point(1, -1)
z.V = point(4, 3)
z.W = (z.C - z.V):orthogonal(3):at(z.C)
local a = tkz.length(z.C, z.V)
local b = tkz.length(z.C, z.W)
local c = math.sqrt(a ^ 2 - b ^ 2)
local e = c / a
axis = line(z.C, z.V)
z.F = axis:report(c, z.C)
z.G = z.C:symmetry(z.F)
z.K = axis:report(b ^ 2 / c, z.F)
z.Kp = axis:report(-b ^ 2 / c, z.G)
z.u = (z.C - z.K):orthogonal(2):at(z.K)
```

```

z.v = (z.C - z.K):orthogonal(-2):at(z.K)
L.dir = line(z.u, z.v)
z.r = (z.C - z.Kp):orthogonal(2):at(z.Kp)
z.s = (z.C - z.Kp):orthogonal(-2):at(z.Kp)
CO.EL = conic(z.F, L.dir, e)
PA.curve = CO.EL:points(0, 1, 100)

```



2.96 Ellipse with Foci and a Point

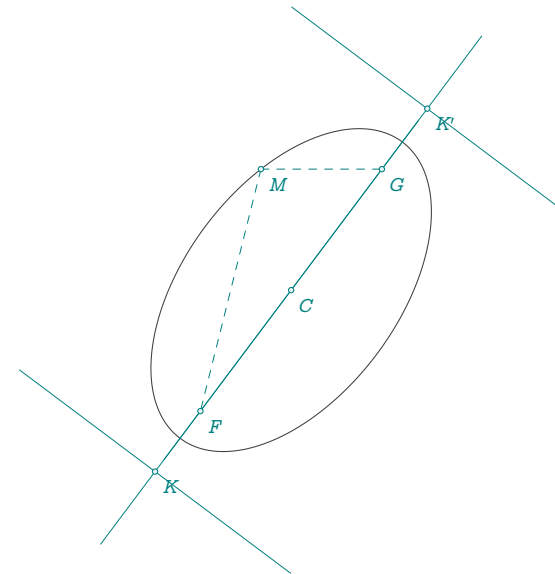
This construction defines an ellipse given its two foci and a single point on the curve. The fundamental property $MF + MG = 2a$, where F and G are the foci and M a point on the ellipse, allows the computation of the semi-major axis a . From this, the eccentricity and the direction of the directrix can be derived.

File ellipse3.lua:

```

init_elements()
z.F = point(1, -1)
z.G = point(4, 3)
z.M = point(2, 3)
z.C = tkz.midpoint(z.F, z.G)
local a = (tkz.length(z.F, z.M) + tkz.length(z.G, z.M)) / 2
local c = tkz.length(z.F, z.G) / 2
local b = math.sqrt(a ^ 2 - c ^ 2)
local e = c / a
axis = line(z.G, z.F)
z.K = axis:report(b ^ 2 / c, z.F)
z.Kp = axis:report(-b ^ 2 / c, z.G)
z.u = (z.C - z.K):orthogonal(2):at(z.K)
z.v = (z.C - z.K):orthogonal(-2):at(z.K)
L.dir = line(z.u, z.v)
z.r = (z.C - z.Kp):orthogonal(2):at(z.Kp)
z.s = (z.C - z.Kp):orthogonal(-2):at(z.Kp)
CO.EL = conic(z.F, L.dir, e)
PA.curve = CO.EL:points(0, 1, 100)

```



2.97 Orthic_inellipse with search_ellipse

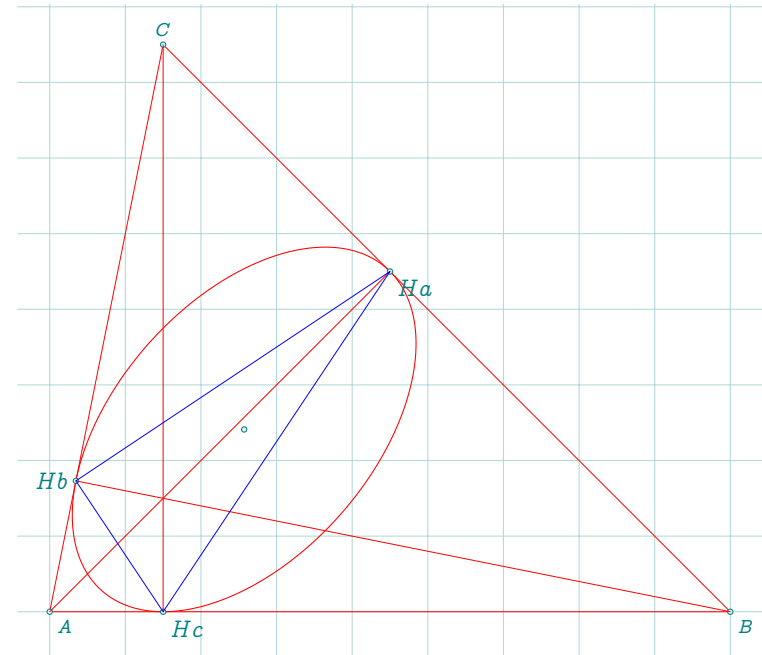
Definition 17: T

e orthic inconic of a triangle is the inconic with inconic parameters $x : y : z = \cos(A) : \cos(B) : \cos(C)$.

It is an ellipse for acute triangles and a hyperbola for obtuse triangles.

File inellipse.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(1, 5)
T.ABC = triangle(z.A, z.B, z.C)
z.H = T.ABC.orthocenter
z.O = T.ABC.circumcenter
T.orthic = T.ABC.orthic()
z.K = T.ABC:symmedian_point()
z.Ha, z.Hb, z.Hc = T.orthic:get()
z.a, z.b, z.c = T.ABC:tangential():get()
z.p, z.q, z.r = T.ABC:circumcevian(z.H):get()
z.Sa, z.Sb = z.K:symmetry(z.Ha, z.Hb)
local coefficients = search_ellipse("Ha", "Hb", "Hc", "Sa", "Sb")
local center, ra, rb, angle = ellipse_axes_angle(coefficients)
CO.EL = conic(EL_radii(z.K, ra, rb, angle))
PA.curve = CO.EL:points(0, 1, 100)
```



2.98 Kiepert hyperbola

Definition 18: Kiepert hyperbola

In triangle geometry, the Kiepert conics are two special conics associated with the reference triangle. One of them is a hyperbola, called the Kiepert hyperbola and the other is a parabola, called the Kiepert parabola.

It has been proved that the Kiepert hyperbola is the hyperbola passing through the vertices, the centroid and the orthocenter of the reference triangle [Wikipedia]

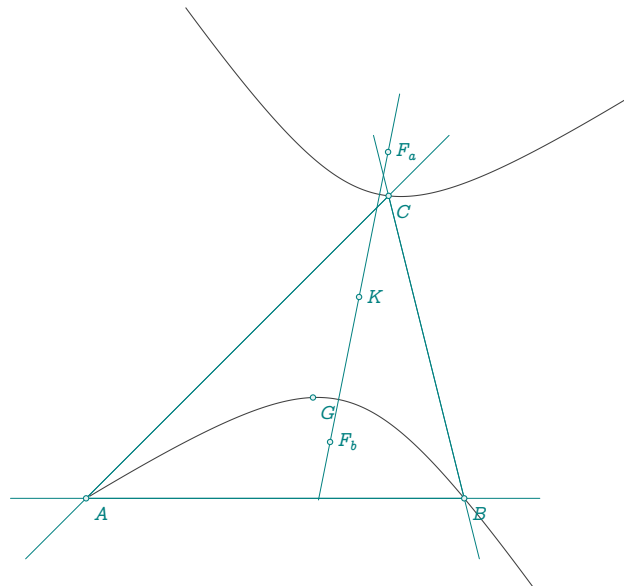
File kiepert_hyperbola.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
z.C = point(4, 4)
T = triangle(z.A, z.B, z.C)
z.K = T:kimberling(115)
z.circumcenter = T.circumcenter
z.G = T.centroid
```

```

L.brocard = T:brocard_axis()
C.circum = circle(z.circumcenter, z.A)
z.M, z.N = intersection(L.brocard, C.circum)
L.asx = T:simson_line(z.M)
L.asy = T:simson_line(z.N)
z.ux, z.uy = get_points(L.asx)
z.vx, z.vy = get_points(L.asy)
HY = T:kiepert_hyperbola()
curve = HY:points(-3, 3, 50)
curveb = HY:points(-3, 3, 50, "swap")
z.F_a, z.F_b = HY.Fa, HY.Fb

```



2.99 Kiepert parabola

Definition 19: Kiepert parabola

The Euler line of a triangle is the conic section directrix of the Kiepert parabola. In fact, the directrices of all parabolas inscribed in a triangle pass through the orthocenter. The triangle formed by the points of contact is called the Steiner triangle.

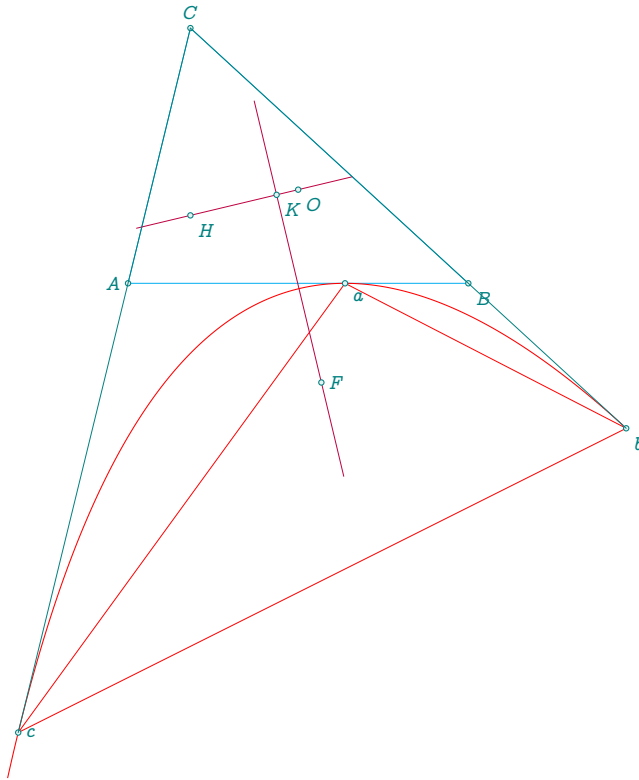
The Kiepert parabola is tangent to the sides of the triangle (or their extensions), the line at infinity, and the Lemoine axis. The focus of the parabola has is Kimberling center X_1 (110).

File kiepert_parabola.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(1.1, 4.5)
T.ABC = triangle(z.A, z.B, z.C)
z.H = T.ABC.orthocenter
z.O = T.ABC.circumcenter
CO.kiepert = T.ABC:kiepert_parabola()
PA.kiepert = CO.kiepert:points(-5, 7, 50)
z.F = CO.kiepert.Fa
z.S = CO.kiepert.vertex
z.K = CO.kiepert.K
z.a = intersection(CO.kiepert, T.ABC.ab)
z.b = intersection(CO.kiepert, T.ABC.bc)
z.c = intersection(CO.kiepert, T.ABC.ca)

```



2.100 Euler ellipse

Definition 20: Euler ellipse

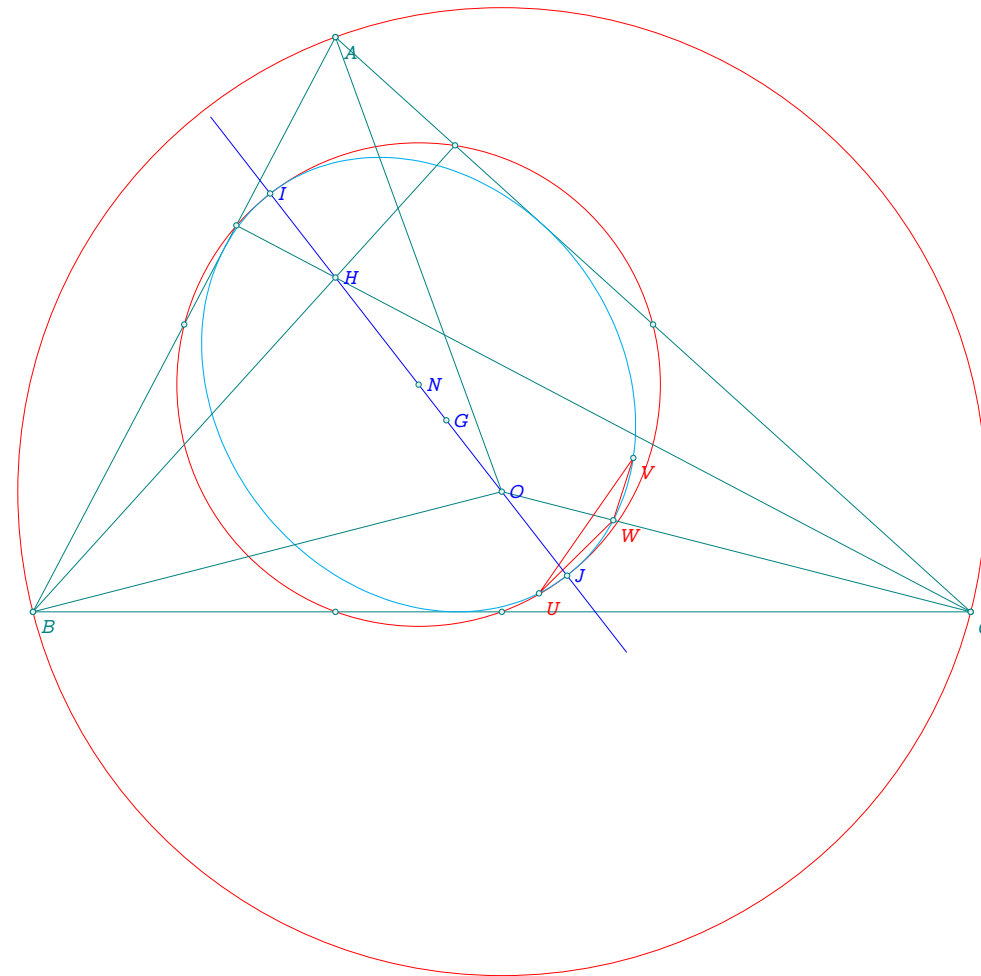
Euler's ellipse is a conic section, tangent to the three sides of a triangle, with foci at the orthocentre and the centre of the circumscribed circle.

File euler_ellipse.lua:

```
init_elements()
z.A = point(2, 3.8)
z.B = point(0, 0)
z.C = point(6.2, 0)
L.AB = line(z.A, z.B)
T.ABC = triangle(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 = T.ABC:medial():get()
z.Ha, z.Hb, z.Hc = T.ABC:orthic():get()
z.Ea, z.Eb, z.Ec = T.ABC:extouch():get()
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.circum)
local a = 0.5 * tkz.length(z.I, z.J)
CO.E = conic(EL_bifocal(z.O, z.H, a))
PA.E = CO.E:points(0, 1, 50)
L.AH = line(z.A, z.H)
L.BH = line(z.B, z.H)
L.CH = line(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(z.B, z.C)
L.XO = line(z.X, z.O)
L.YO = line(z.Y, z.O)
L.ZO = line(z.Z, z.O)
z.x = intersection(L.BC, L.XO)
z.U = intersection(L.XO, CO.E)
_, z.V = intersection(L.YO, CO.E)
_, z.W = intersection(L.ZO, CO.E)
```

Let ABC be an acute-angled triangle, neither right-angled nor equilateral.



2.101 Related circles from Leon Bankoff

The shaded circles are equal and the unshaded circles are equal.

File bankoff.lua:

```

init_elements()
local n = 0
  PA.c = path:new()
  PA.t = path:new()
local function push(o, t)
  if o and t then
    PA.c:add_point(o)
    PA.t:add_point(t)
    n = n + 1
  end
end
local R = 8
local r = R / 8
z.O = point(0, 0)
z.A = point(R, 0)
C.A0 = circle(z.A, z.O)
z.B = point(R, R)
L.OB = line(z.O, z.B)
T.OAB = triangle(z.O, z.A, z.B)
z.A2 = T.OAB.ab.mid
z.B2 = T.OAB.bc.mid
z.M = L.OB.mid
z.a1 = z.A2 + point( r,  r)
z.a2 = z.M + point( r, -r)
z.a3 = z.A + point(-r,  r)
z.a4 = z.B2 + point(-r, -r)
for i, A in ipairs{z.a1, z.a2, z.a3, z.a4} do
  local B
  if i <= 2 then B = tkz.midpoint(z.a1, z.a2)
  else          B = tkz.midpoint(z.a3, z.a4)
  end
  push(A, B)
end
T.OA2M = triangle(z.O, z.A2, z.M)
T.MBB2 = triangle(z.M, z.B, z.B2)
z.m1, z.th1 = T.OA2M:in_circle():get()
z.m2, z.th2 = T.MBB2:in_circle():get()

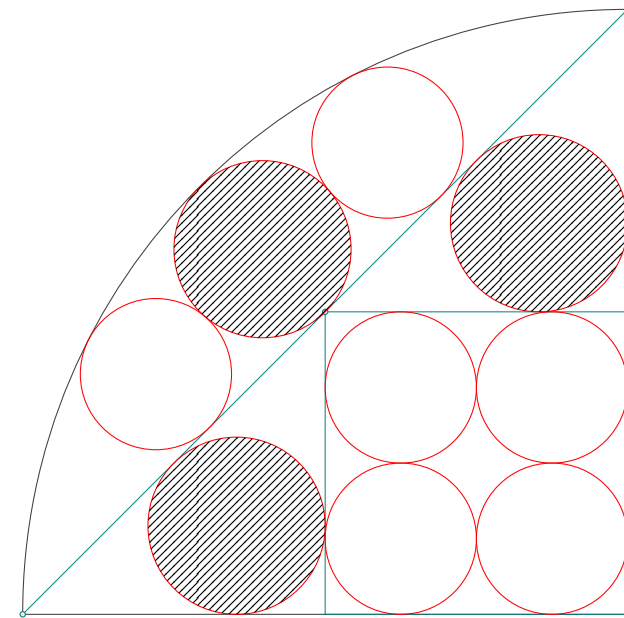
```

```

push(z.m1, z.th1)
push(z.m2, z.th2)
local pw, pt = C.A0:CLP(L.OB, z.M)
z.w1 = pw:get(1)
z.t1 = pt:get(1)
push(z.w1, z.t1)
C.w1 = circle(z.w1, z.t1)
local pw, pt = C.A0:CCL(C.w1, L.OB)
z.c1 = pw:get(1)
z.h1 = pt:get(1)
z.c2 = pw:get(2)
z.h2 = pt:get(2)
push(z.c1, z.h1)
push(z.c2, z.h2)

```

Let ABC be an acute-angled triangle, neither right-angled nor equilateral.



3 Apollonius' problem

Overview

The classical Apollonius' problem asks for one or several circles tangent to three given geometric objects, each of which may be a circle (**C**), a line (**L**) or a point (**P**). Each case is denoted by a three-letter code such as **CCP**, **LLL**, or **PLC**, according to the type of the three given objects.

Historically, there are ten distinct configurations:

PPP, LPP, CPP, LLP, LLL, CLP, CLL, CCP, CCL, CCC.

Viète solved all ten cases using only compass and straightedge constructions, building complex solutions from simpler ones.

Note (work in progress): This section is under active development. Some *degenerate* or *very special* configurations may not be fully handled yet.

Unified API (paths everywhere)

Each case is implemented as a method attached to the class of the *first* object (e.g. `point:PPP`, `line:LLL`, `circle:CLL`). All methods are being unified to return *paths* so that drawing and node extraction are consistent across cases.

Syntax: `PA.center`, `PA.through`, `n = <obj>:XYZ(args)`

Returned values:

- `PA.center`: a path collecting the centers of the solution circles;
- `PA.through`: a path collecting the corresponding through-points;
- `n`: the number of valid solutions (can be 0).

Remarks:

- All results can be directly drawn using `\tkzDrawCirclesFromPaths(PA.center, PA.through)`.
- Nodes can be exported to TikZ with `tkz.nodes_from_paths(PA.center, PA.through)`.
- Empty paths correspond to degenerate configurations.
- Some older methods still return `paths`, `n`; they will be progressively updated to the unified format.

Using `tkz.nodes_from_paths` instead of `\tkzDrawCirclesFromPaths`

If you wish to draw *all* the solution circles at once, the macro `\tkzDrawCirclesFromPaths(PA.ce, PA.th)` is the most convenient approach. However, if you need to highlight or draw *only one specific circle*, it is often preferable to use the Lua helper `tkz.nodes_from_paths`. In this case, you may keep the solutions in local variables to avoid polluting the global table:

```
local pc, pt, n = C.OC:CPP(z.A, z.B)
tkz.nodes_from_paths(pc, pt)
% creates z.w1, z.t1, z.w2, z.t2,...
```

By default, this generates nodes `(w1,t1)`, `(w2,t2)`, ... in the table `z`, which can then be used to draw individual circles or to label / style them differently.

Combining both approaches: You can also combine the two techniques: first use `\tkzDrawCirclesFromPaths` to display all solution circles, and then use `tkz.nodes_from_paths` to extract the corresponding nodes and emphasize only some of them (change colour, add labels, draw radii, etc.). This is particularly useful in demonstrations or pedagogical figures.

Example idea:

- Compute the solutions:
`PA.center, PA.through, n = C.OC:CPP(z.A, z.B)`.
- Draw all circles with `\tkzDrawCirclesFromPaths`.
- Call `tkz.nodes_from_paths(PA.center, PA.through)`.
- Then, for instance: `\draw[red,thick] (z.w1) circle[radius];` to highlight only the first one.

Thus, the macro gives a quick visual result, while `tkz.nodes_from_paths` offers fine control over individual solutions.

Global `PA.xxx` vs Local Variables:

When using contact methods such as:

```
PA.center, PA.through, n = C.OC:CPP(z.A, z.B)
```

the two paths `PA.center` and `PA.through` are stored as *global* variables in the `z/PA` namespace. This has two important advantages:

- They can be used directly in TikZ with macros such as:
`\tkzDrawCirclesFromPaths(PA.center, PA.through)`.
- They are automatically cleared when calling `init_elements()`, which avoids name conflicts between different figures or compilations.

Since TikZ macros work purely on the TeX side, the paths must exist globally to be accessible outside the `\directlua` block.

Using Local Paths with `tkz.nodes_from_paths`:

If you do not want to pollute the global namespace, you may store the results locally:

```
local pc, pt, n = C.OC:CPP(z.A, z.B)
tkz.nodes_from_paths(pc, pt) -- creates z.w1, z.t1,...
```

Here, `pc` and `pt` remain Lua-local, but the function `tkz.nodes_from_paths` exports the corresponding nodes (`w1, t1, w2, t2, ...`) into the global table `z` so they can be used in TikZ.

Which One to Use?

- Use `PA.center`, `PA.through` (global) if you plan to call:
`\tkzDrawCirclesFromPaths(PA.center, PA.through)`

because this macro expects global paths accessible from TeX.

- Use `tkz.nodes_from_paths` if you want to select or highlight only some of the solution circles. It also works with either global or local paths.
- A combined approach is often the most flexible: draw all circles with the macro, then export nodes to highlight specific ones.

Example Workflow:

1. Compute solutions globally:
`PA.center, PA.through, n = C.OC:CPP(z.A, z.B)`
2. Draw all circles:
`\tkzDrawCirclesFromPaths(PA.center, PA.through)`
3. Export the same centers and points to nodes:
`\directlua{ tkz.nodes_from_paths(PA.center, PA.through) }`
4. Now highlight one circle, for instance:
`\draw[red,thick] (z.w1) -- (z.t1);`

In summary, `PA.xxx` must be *global* for TikZ macros, while `tkz.nodes_from_paths` lets you create node names (`w1, t1, ...`) from either global or local paths depending on how much control you need.

3.1 Class point: method PPP(p, p)

Purpose:

The method `PPP` constructs the *circumcircle* passing through three distinct points: the current point (`self`) and the two given points `a` and `b`.

If the three points are collinear or not distinct, no circle can be defined and the method returns empty paths with a counter equal to 0.

Syntax: `PA.center, PA.through, n = z.A:PPP(z.B, z.C)`

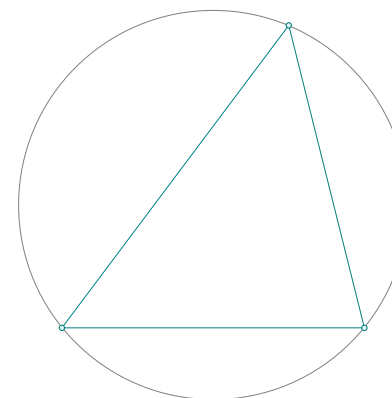
Returns:

- **PA.center**: a **path** containing the center(s) of the circumcircle(s);
- **PA.through**: a **path** containing the corresponding point(s) on each circle;
- **n**: the number of valid circles (1 if a circumcircle exists, 0 otherwise).

Example usage:

File `apo_PPP.lua`:

```
init_elements()
z.A = point(0, 0)
z.B = point(4, 0)
z.C = point(3, 4)
PA.center, PA.through, _ = z.A:PPP(z.B, z.C)
tkz.nodes_from_paths(PA.center, PA.through)
-- w and t by default
```



Remarks: This is the simplest case of the Apollonius family. If `n = 0`, the three points are collinear and no circle is drawn.

3.2 Class line: method LPP(p, p)

Purpose: Given a running line L (the current object **self**) and two points **a** and **b** lying on the *same side* of L , this method constructs the circle(s) tangent to L and passing through both **a** and **b**.

Syntax: `local PA.center, PA.through, n = L:LPP(a, b)`

Parameters:

- **a, b**: two distinct points.
- *Precondition*: **a** and **b** are on the same side of the running line **self**.

The method is designed to handle a variety of geometric configurations, including the following special cases:

- The segment $[MN]$ is perpendicular to the line (AB) ;
- The segment $[MN]$ is parallel to the line (AB) ;
- The points M and N lie on opposite sides of the line (AB) ;
- One of the points (either M or N) lies on the line (AB) .

This construction is useful in problems involving constrained circle placements or Apollonius-type configurations.

Returns:

- `PA.center` and `PA.through`: two paths containing respectively the centers and the points of tangency of the solution circles.
- `n`: the number of existing solutions (0, 1 or 2).

Remarks:

- If `n = 0`, both paths are empty and no circle can be drawn.
- In degenerate cases with a single solution, the same circle is recorded once (`n = 1`).
- No `tex.error` is raised; the method always returns valid paths.
- The results can be transferred and drawn using the macros: `\tkzNodesFromPaths` and `\tkzDrawCirclesFromPaths`.

Existence & number of solutions

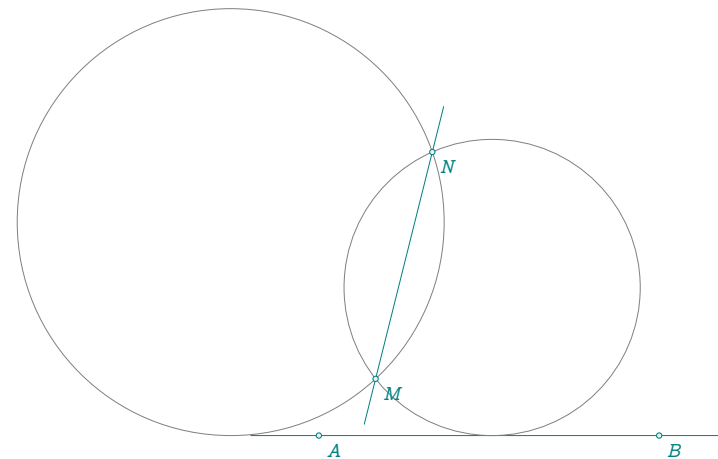
- No solution (0). If the intersection point $i = L \cap (ab)$ exists and lies on the segment $[a, b]$, then no circle tangent to L can pass through a and b .
- Single solution (1).

- If exactly one of the points **a** or **b** belongs to L (point of tangency), the unique solution has its center at the intersection of the mediator of $[a, b]$ with the perpendicular to L at that point.
- If $L \parallel (ab)$, the unique solution has its center at the circumcenter of the triangle (a, b, p) where p is the orthogonal projection of $\text{mid}(a, b)$ on L .
- Two solutions (2).
 - If $L \perp (ab)$ and $i = L \cap (ab)$ is defined (and not on $[a, b]$), two symmetric solutions exist.
 - In the general oblique case (neither parallel nor orthogonal), two solutions exist.

Example usage:

File `apo_LPP.lua`:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.M = point(1, 1)
z.N = point(2, 5)
L.AB = line(z.A, z.B)
PA.center,
PA.through, n = L.AB:LPP(z.M, z.N)
tkz.nodes_from_paths(PA.center, PA.through, "O", "T")
```



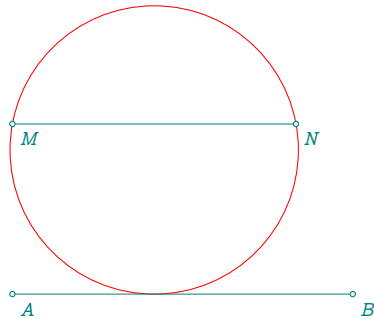
Let's look at the case where the line (MN) is parallel to the initial line.

File apo_LPP2.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.M = point(0, 3)
z.N = point(5, 3)
L.AB = line(z.A, z.B)
PA.center, PA.through, n = L.AB:LPP(z.M, z.N)

```



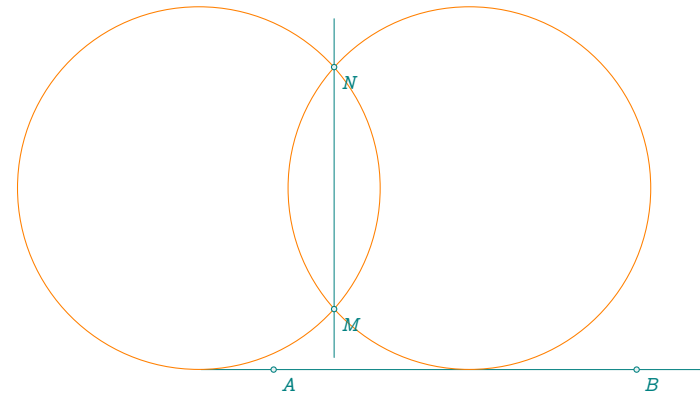
Where the line is perpendicular to the initial line.

File apo_LPP3.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.M = point(1, 1)
z.N = point(1, 5)
L.AB = line(z.A, z.B)
PA.center,
PA.through, n = L.AB:LPP(z.M, z.N)

```



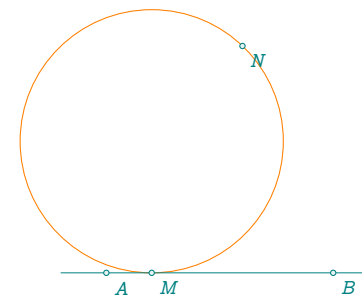
The last special case is when one of the points is on the initial line. In this case, there's only one solution.

File apo_LPP4.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
z.M = point(1, 0)
z.N = point(3, 5)
L.AB = line(z.A, z.B)
PA.center,
PA.through, n = L.AB:LPP(z.M, z.N)
PA.center, PA.through,
n = L.AB:LPP(z.M, z.N)

```



3.3 Class line: method LLP(L, p)

Purpose: Given the running line D (**self**), another line L , and a point p , this method constructs the circle(s) passing through p and tangent to *both* lines D and L .

Syntax: `PA.center, PA.through, n = L.AB:LLP(L, p)`

Let us consider two straight lines, (AB) and (CD) , and a point P not lying on either line. The problem is to determine whether there exists a circle passing through P and tangent to both lines.

Parameters:

- L : a line distinct from the running line **self** (they may be parallel or intersecting).
- p : a point in the plane.

Returns:

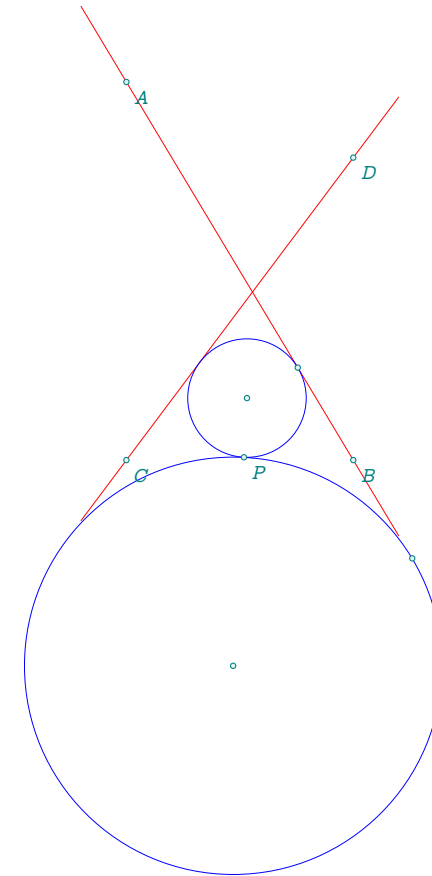
- **PA.center, PA.through**: two paths containing the centers and tangent points of the solution circles.
- **n**: the number of solutions (0, 1, or 2).
- If no solution exists, **n = 0** and both paths are empty.

Remarks: This method generalizes Apollonius' problem in the case of two lines and one point. It handles both parallel and intersecting lines, as well as special configurations where the point lies on a bisector or one of the lines.

General case

File apo_LL2.lua:

```
init_elements()
z.A = point(-1, 2)
z.B = point(2, -3)
z.C = point(-1, -3)
z.D = point(2, 1)
L.AB = line(z.A, z.B)
L.CD = line(z.C, z.D)
z.S = intersection(L.AB, L.CD)
L.SI = tkz.bisector(z.S, z.A, z.D)
z.P = L.SI:point(-1)
local centers,
  throughs, n = L.AB:LLP(L.CD, z.P)
tkz.nodes_from_paths(centers, throughs)
```

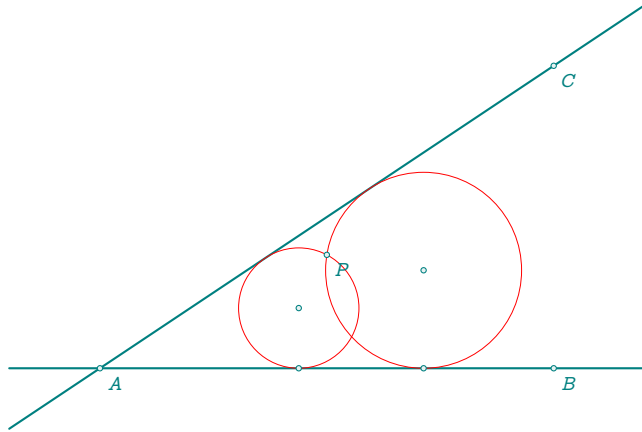


Point inside the angle

File apo_LL2.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
L.AB = line(z.A, z.B)
z.C = point(6, 4)
L.AC = line(z.A, z.C)
z.P = point(3, 1.5)
local centers,
```

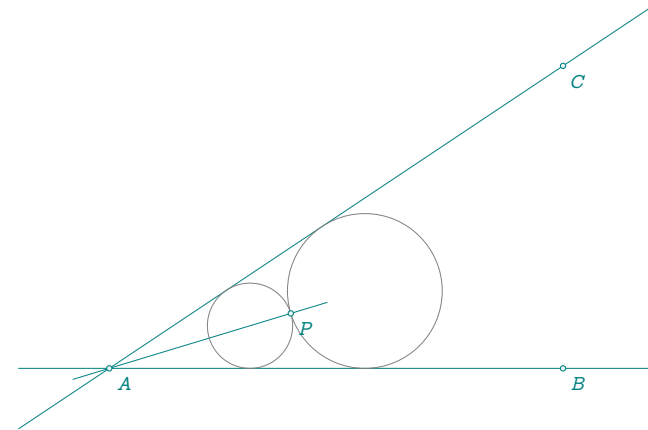
```
throughs, n = L.AB:LLP(L.AC, z.P)
tkz.nodes_from_paths(centers, throughs)
```



Point on a bisector

File apo_LL3.lua:

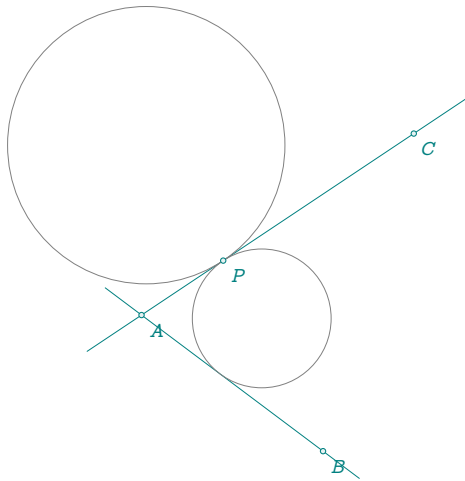
```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
L.AB = line(z.A, z.B)
z.C = point(6, 4)
L.AC = line(z.A, z.C)
L.bi = tkz.bisector(z.A, z.B, z.C)
z.P = L.bi:point(0.4)
local centers,
  throughs, n = L.AB:LLP(L.AC, z.P)
tkz.nodes_from_paths(centers, throughs)
```



Point on one of the lines

File apo_LL4.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(4, -3)
L.AB = line(z.A, z.B)
z.C = point(6, 4)
L.AC = line(z.A, z.C)
z.P = L.AC:point(.3)
local centers,
  throughs, n = L.AB:LLP(L.AC, z.P)
tkz.nodes_from_paths(centers, throughs)
```



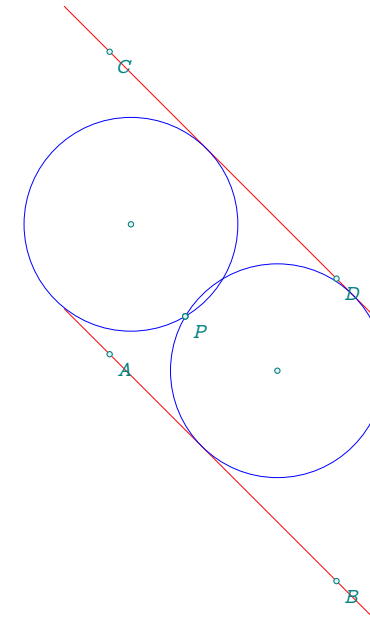
Parallel lines

File apo_LL5.lua:

```

init_elements()
z.A = point(-2, 2)
z.B = point(1, -1)
z.C = point(-2, 6)
z.D = point(1, 3)
L.AB = line(z.A, z.B)
L.CD = line(z.C, z.D)
z.P = point(-1, 2.5)
local centers,
  throughs, n = L.AB:LLP(L.CD, z.P)
tkz.nodes_from_paths(centers, throughs)

```



3.4 Class line: method LLL(L, L)

This method completes the family of contact configurations. It handles the case of a circle tangent to three lines.

Syntax:

```

C_in = L.AB:LLL(L.BC, L.CA, "in")
C_exA = L.AB:LLL(L.BC, L.CA, "pa")
C_in, C_ea, C_eb, C_ec = L.AB:LLL(L.BC, L.CA)

```

Purpose:

The method **LLL** constructs the circles tangent to three lines.

- If the three lines are *non-parallel*, they define a triangle, and **LLL** returns the *incircle* and the three *excircles*, equivalent to **triangle:c_lll("all")**.
- If exactly two lines are parallel and the third one is transversal, two circles are tangent to the three lines; these two solutions are returned.

Returns:

- In the triangle case:

- "all" : four circles (incircle and three excircles),
- "in" : incircle only,
- "pa", "pb", "pc" : the corresponding excircle.
- In the parallel case: two circles tangent to the three lines.

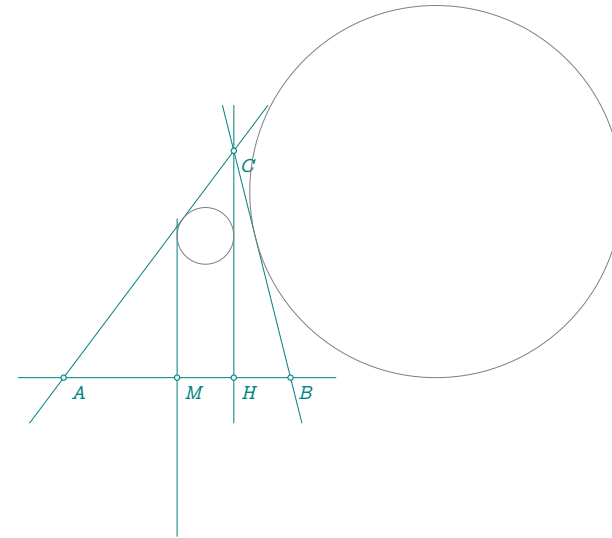
Errors:

If at least two pairs of lines are parallel, the configuration is degenerate.

Example usage:

File apo_LLL1.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(4.5, 6)
T.ABC = triangle(z.A, z.B, z.C)
L.AB = line(z.A, z.B)
L.AC = line(z.A, z.C)
L.med = L.AB : mediator ()
z.M = L.AB.mid
z.x, z.y = get_points(L.med)
z.H = L.AB:projection(z.C)
L.ortho = L.AB:orthogonal_from(z.C)
PA.center, PA.through = L.AC:LLL(L.med, L.ortho)
z.w = PA.center:get(2)
z.t = PA.through:get(2)
PA.center, PA.through = L.AB:LLL(L.AC, T.ABC.bc)
z.o = PA.center:get(4)
z.h = PA.through:get(4)
```



3.5 Class: method CLL(L, L)

Alias: c_cll

Syntax:

PA.center, PA.through, n = circle:CLL(L1, L2, choice, inside)

Purpose:

Constructs the circles that are tangent to:

- the current circle (self),
- and the two lines L1 and L2.

The method returns two **path** objects containing all the solutions found:

- PA.center – centers of the solution circles,
- PA.through – corresponding through-points (point of tangency),
- n – number of solutions.

Options:

- choice – Optional. Integer from 1 to 4 selecting one of the angular sectors formed by the two lines at their intersection. By default, choice = "all" which means all sectors are explored and all valid solutions are returned.
- inside – Optional. If set to "inside", the circle is tangent internally to the reference circle (its radius is considered negative in the auxiliary construction). If omitted, tangency is external.

Explanation:

- The intersection of L1 and L2 defines four sectors numbered from 1 to 4 in the direct (counterclockwise) order starting from L1.
- For each chosen sector, up to two circles can be tangent to the two lines and to the reference circle.
- With the default option `choice = "all"`, the method tries all four sectors and appends every valid solution to the paths.

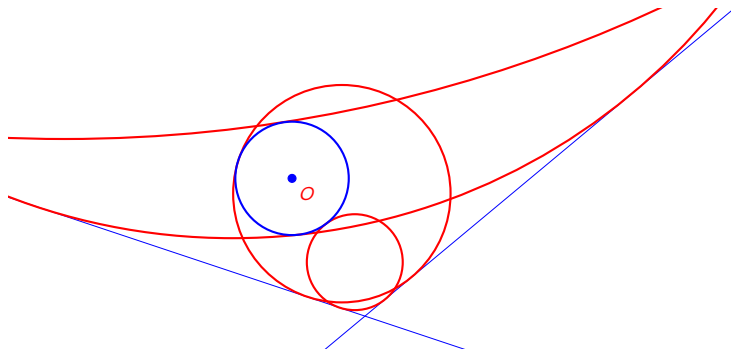
Remarks:

- If the lines are parallel, this method does not apply.
- The user can draw the resulting circles using `\tkzDrawCirclesFromPaths(PA.center, PA.through)`.
- The value of `n` gives the total number of circles stored in the paths.

Examples usage:

File `apo_CLL1.lua`:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, -2)
L.AB = line(z.A, z.B)
z.C = point(-3, -4)
z.D = point(3, 1)
L.CD = line(z.C, z.D)
z.O = z.D + point(-3,1)
z.X = z.O + point(0,1)
C.OX = circle(z.O, z.X)
PA.center, PA.through = C.OX:CLL(L.AB, L.CD,"all")
tkz.nodes_from_paths(PA.center, PA.through)
```

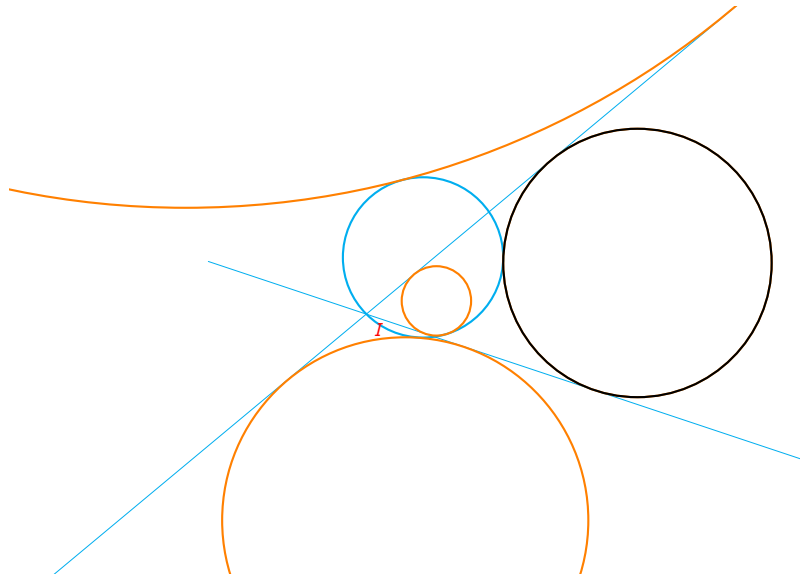


Some explanations: The given circle lies within sectors 1 and 2. In this case, we are looking for the solution circles located in these two sectors. There is a method that determines how a circle is positioned with respect to two lines: `circle:lines_position(L1, L2)`. If you do not know how many solutions exist, the following example shows how to proceed.

A special case: The intersection point of the two lines belongs to the given circle. In this example, the `CLL_all` method determines all solutions. The points that define the solution circles are stored in objects of type `path`.

File `apo_CLL2.lua`:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, -2)
L.AB = line(z.A, z.B)
z.C = point(-3, -4)
z.D = point(3, 1)
L.CD = line(z.C, z.D)
z.I = intersection(L.AB, L.CD)
z.O = z.I + point(1,1)
z.X = z.I
C.OX = circle(z.O, z.X)
PA.center, PA.through = C.OX:CLL_all(L.AB, L.CD)
z.w1 = PA.center:get(1)
z.t1 = PA.through:get(1)
```



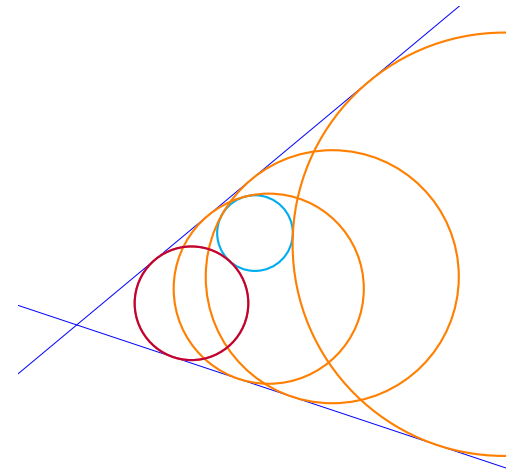
Another example

File apo_CLL3.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(6, -2)
L.AB = line(z.A, z.B)
z.C = point(-3, -4)
z.D = point(3, 1)
L.CD = line(z.C, z.D)
z.O = z.D + point(3,1)
z.X = z.O + point(0,1)
C.OX = circle(z.O, z.X)
PA.center, PA.through = C.OX:CLL_all(L.AB, L.CD)
z.w1 = PA.center:get(2)
z.t1 = PA.through:get(2)

```



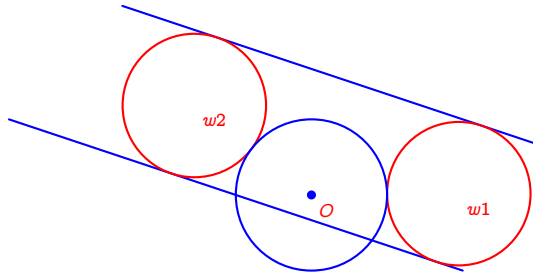
The last case: the lines are parallel

File apo_CLL4.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(6, -2)
L.AB = line(z.A, z.B)
z.C = point(-3, -3)
z.D = point(3, -5)
L.CD = line(z.C, z.D)
z.O = point(2, -4)
z.X = z.O + point(0,2)
C.OX = circle(z.O, z.X)
PA.center, PA.through, n = C.OX:CLL(L.AB, L.CD)
tkz.nodes_from_paths(PA.center, PA.through)

```



3.6 Class circle: method CPP(p, p)

Circle tangent to a given circle and passing through two points.

Purpose:

Construct the circle(s) tangent to a given circle (the receiver `self`) and passing through two given points `a` and `b`.

The name CPP stands for: *circle tangent to one circle and passing through two points*.

Remarks: This method returns one or two solution circles, depending on the configuration. A valid solution exists only if both points are on the *same side* of the given circle (both inside or both outside).

Parameters:

- `a, b` — two distinct points through which the solution circles must pass.

Syntax: `local PA.center, PA.through, n = C.OC:CPP(a, b)`

Returns:

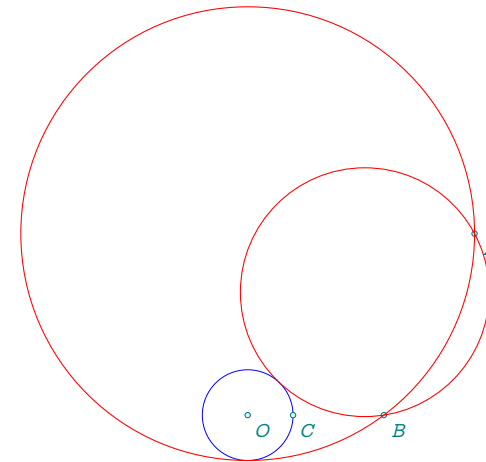
- `PA.center` — a **path** containing the centers of the solution circles;
- `PA.through` — a **path** containing the corresponding points on each circle;
- `n` — the number of solutions (0, 1, or 2).

Usage: After calling this method, the Lua function `tkz.nodes_from_paths(PA_center, PA_through)` creates the corresponding nodes `z.w1, z.w2, ...` and `z.t1, z.t2, ...` for TeX drawing. You can then draw all circles within a `tikzpicture` environment using:
`\tkzDrawCirclesFromPaths(PA.center, PA.through).`

Existence condition: A solution exists only if both points `a` and `b` are on the same side of the given circle (either both inside or both outside its disk). If one point lies inside and the other outside, no solution is returned (`n = 0`).

File `apo_CPP1.lua`:

```
init_elements()
z.A = point(5,4)
z.B = point(3,0)
z.O = point(0,0)
z.C = point(1,0)
C.OC = circle(z.O, z.C)
PA.center, PA.through,
n = C.OC:CPP(z.A, z.B)
```



Special cases:

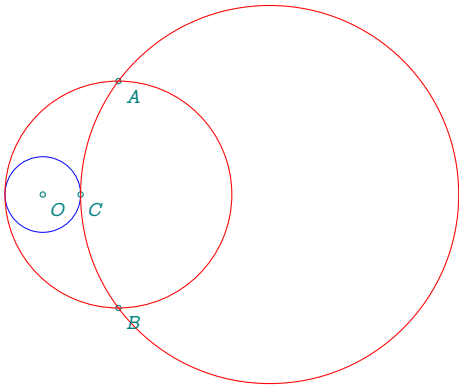
Equidistant points from the center:

File `apo_CPP2.lua`:

```

init_elements()
z.A = point(2,3)
z.B = point(2,-3)
z.O = point(0,0)
z.C = point(1,0)
C.OC = circle(z.O, z.C)
PA.center,
PA.through,
n = C.OC:CPP(z.A, z.B)

```



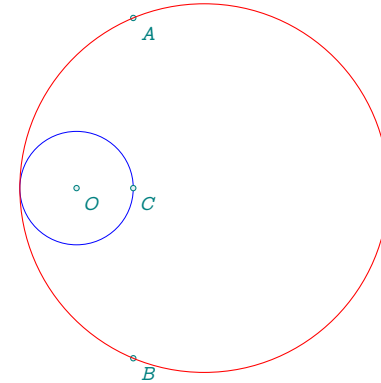
Tangent case: When the line (AB) is tangent to the initial circle, there may be a single (degenerate) solution. It can even happen that (AB) is tangent and both points are equidistant from the center.

File apo_CPP3.lua:

```

init_elements()
z.A = point(1,3)
z.B = point(1,-3)
z.O = point(0,0)
z.C = point(1,0)
C.OC = circle(z.O, z.C)
PA.center, PA.through,
n = C.OC:CPP(z.A, z.B)

```



3.7 Class circle: method CLP(L, p, mode)

Purpose: Circle tangent to a given line and to the current circle, passing through a given point.

Syntax: PA.center, PA.through, n = circle:CLP(L, p [, mode])

- L — a line object.
- p — a point through which the solution circle(s) must pass.
- mode — optional string in "all", "external", "internal". Default is "all".

Returns: three values:

1. PA.center — path of centers,
2. PA.through — path of through-points,
3. n — number of solutions (0...4).

These paths can be drawn directly with:

`\tkzDrawCirclesFromPaths(PA.center, PA.through).`

Existence and number of solutions

- Generic case (point not on line nor circle): up to 4 circles.
- Special cases (only 2 or 0 solutions):
 - p on the circle : 2 solutions,
 - p on the line : 2 solutions,
 - line tangent to the circle : one family may collapse,
 - p at line-circle intersection : no non-trivial solution.

Options

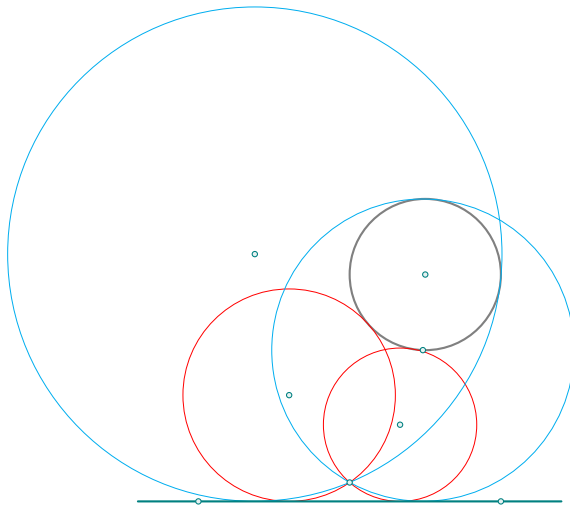
- "all" (default) — internal and external families (up to 4),

- "external" — only externally tangent solutions,
- "internal" — only internally tangent solutions ("inside" accepted).

Examples usage

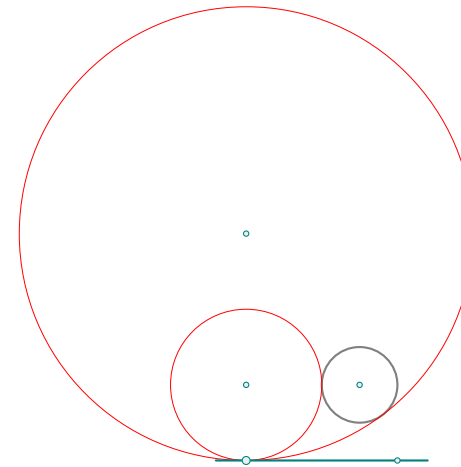
File apo_CLP1.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(4, 0)
L.AB = line (z.A, z.B)
z.O = point(3, 3)
z.T = point(3, 2)
z.P = point(2, .25)
C.OT = circle(z.O, z.T)
PA.center, PA.through = C.OT:CLP(L.AB, z.P)
z.O1 = PA.center:get(1)
z.O2 = PA.center:get(2)
PA.center,
PA.through = C.OT:CLP(L.AB, z.P, 'internal')
z.O3 = PA.center:get(1)
z.O4 = PA.center:get(2)
```



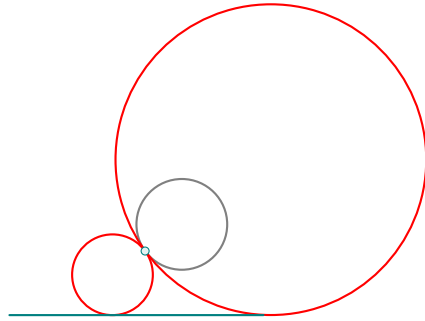
File apo_CLP2.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(4, 0)
L.AB = line (z.A, z.B)
z.O = point(3, 2)
z.T = point(3, 1)
C.OT = circle(z.O, z.T)
PA.center,
PA.through = C.OT:CLP(L.AB, z.A, "internal")
z.O1 = PA.center:get(1)
z.O2 = PA.center:get(2)
```



File apo_CLP3.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(4, 0)
L.AB = line(z.A, z.B)
z.O = point(3, 2)
z.T = point(2, 2)
C.OT = circle(z.O, z.T)
z.P = C.OT:point(0.1)
C.OT = circle (z.O , z.T)
PA.center,
PA.through = C.OT:CLP(L.AB, z.P)
```



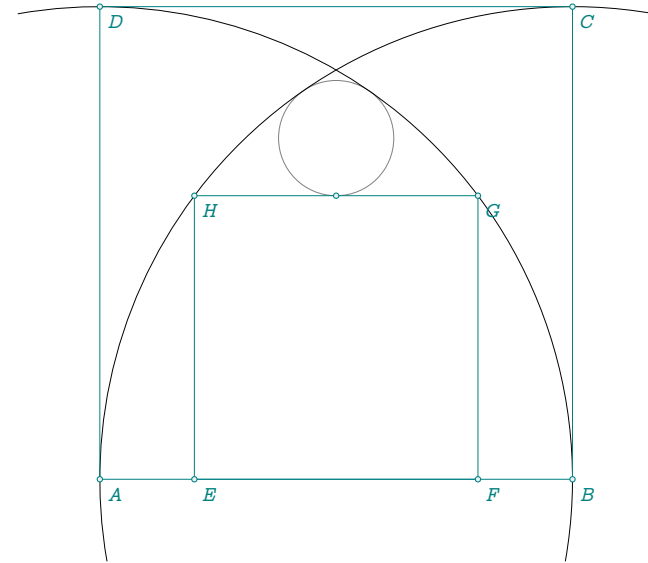
Application:

File apo_CLP4.lua:

```

init_elements()
xB = 5
z.A = point(0, 0)
z.B = point(xB, 0)
L.AB = line(z.A, z.B)
C.AB = circle(z.A, z.B)
C.BA = circle(z.B, z.A)
S.AB = L.AB:square()
_, _, z.C, z.D = S.AB:get()
z.E = point(1, 0)
z.F = point(4, 0)
L.EF = line(z.E, z.F)
S.EF = L.EF:square()
_, _, z.G, z.H = S.EF:get()
z.M = S.EF.cd.mid
PA.center, PA.through = C.AB:CLP(S.EF.cd, z.M)
z.w = PA.center:get(2)
z.t = PA.through:get(2)

```



3.8 Class circle: method CCP(C, p, mode)

Purpose: Circle tangent to two given circles and passing through a point.
This method constructs 0, 2, or 4 circles depending on the configuration, passing through a given point p and tangent to two given circles.

The name **CCP** stands for:

circle tangent to two circles and passing through a point.

medskip Syntax: $C1, C2 = C.A:CCP(C.B, p)$ -- external solutions
 $C3, C4 = C.A:CCP(C.B, p, "internal")$ -- internal solutions

The general problem may admit up to four solutions: two circles tangent along the *external* common tangents, and two circles tangent along the *internal* common tangents. For convenience, the method accepts an option:

- **"external"** (default): returns the two solutions associated with the external tangents,
- **"internal"**: returns the two solutions associated with the internal tangents.

Special cases:

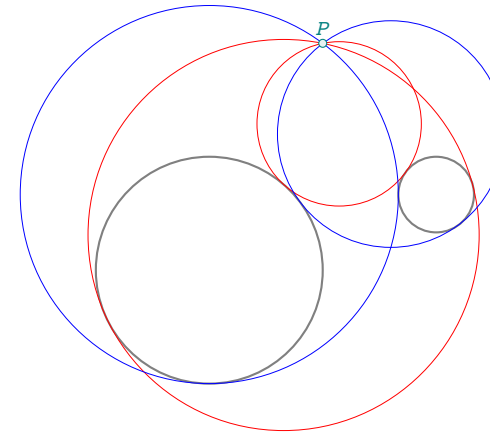
- If the point p lies strictly inside or outside both disks, up to four solutions exist in total (two external, two internal).

- If p lies on the circumference of one of the given circles, then p is already a contact point; only two solutions remain.
- If p lies simultaneously on both circles (intersection points), the configuration is degenerate and usually no non-trivial solution exists.

Examples usage:

File apo_CCP1.lua:

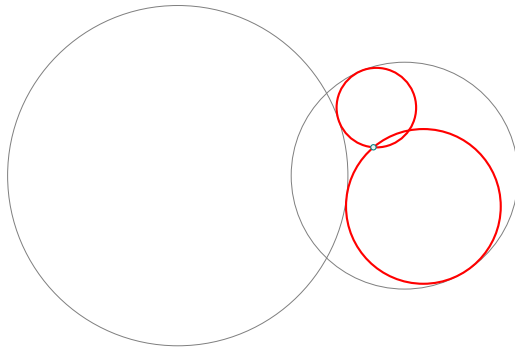
```
init_elements()
z.A = point(0, 0)
z.TA = point(3, 0)
z.B = point(6, 2)
z.TB = point(6, 1)
z.P = point(3, 6)
C.A = circle(z.A, z.TA)
C.B = circle(z.B, z.TB)
PA.center, PA.through = C.A:CCP(C.B,
    z.P,"external")
z.O1 = PA.center:get(1)
z.O2 = PA.center:get(2)
z.T1 = PA.through:get(1)
z.T2 = PA.through:get(2)
PA.center, PA.through = C.A:CCP(C.B,
    z.P,"internal")
z.O3 = PA.center:get(1)
z.O4 = PA.center:get(2)
z.T3 = PA.through:get(1)
z.T4 = PA.through:get(2)
```



The point is inside a circle

File apo_CCP2.lua:

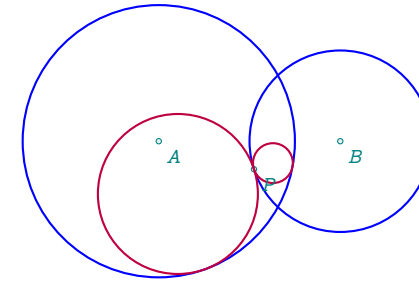
```
init_elements()
z.A = point(0, 0)
z.TA = point(3, 0)
z.B = point(4, 0)
L.AB = line(z.A, z.B)
z.TB = point(6, 0)
C.A = circle(z.A, z.TA)
C.B = circle(z.B, z.TB)
z.P = point(3.45, 0.5)
z.X, z.Y = intersection(C.A,C.B)
PA.center, PA.through = C.A:CCP(C.B, z.P)
tkz.nodes_from_paths(PA.center, PA.through)
```



The point is on a circle

File apo_CCP3.lua:

```
init_elements()
z.A = point(0, 0)
z.a = point(3, 0)
z.B = point(4, 0)
z.b = z.B + point(0,2)
L.AB = line(z.A, z.B)
C.Aa = circle(z.A, z.a)
C.Bb = circle(z.B, z.b)
z.P = C.Bb:point(0.3)
PA.center,
PA.through,n = C.Aa:CCP(C.Bb, z.P)
```



3.9 Class circle: method CCL(C2, L)

Purpose: Find all circles that are tangent to:

- the current circle **self** (called **C1**),
- another circle **C2**,
- and a line **L**.

This method returns all possible solution circles (up to four). Syntax:

```
PA.center, PA.through, n = C1:CCL(C2, L)
```

- **C1** — the circle on which the method is applied (**self**),
- **C2** — another circle to which the solution circle must be tangent,
- **L** — a line object.

Returns:

- **pc** — a **path** containing the centers of the solution circles,
- **pt** — a **path** containing the corresponding tangency points on **C1**,
- **n** — the number of solutions (from 0 to 4).

Special cases.

- If one of the circles is already tangent to **L** at a point **p**, any solution circle must also pass through **p**.
- If the line **L** intersects **C1**, no circle can be tangent to both **C1** and **L** on that side.
- If all three objects are mutually tangent, the configuration is degenerate and may admit a single or no solution.

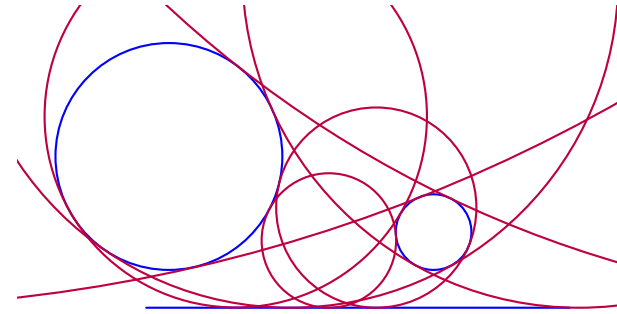
This method works with:

- `tkz.nodes_from_paths(pc, pt)` to name the solution points (**w1,t1**), (**w2,t2**), etc.,
- `\tkzDrawCirclesFromPaths(pc, pt)` to draw all the solution circles.

Example A:

File apo_CCLA.lua:

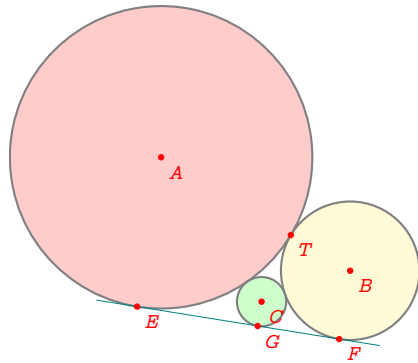
```
init_elements()
z.A = point(-4, 4)
z.B = z.A + point(3, 0)
C.AB = circle(z.A, z.B)
z.C = point(3, 2)
z.D = z.C + point(1, 0)
C.CD = circle(z.C, z.D)
z.E = point(-3, 0)
z.F = point(5, 0)
L.EF = line(z.E, z.F)
PA.center, PA.through, n = C.AB:CCL(C.CD, L.EF)
tkz.nodes_from_paths(PA.center, PA.through)
```



Example B:

File apo_CCLB.lua:

```
init_elements()
z.A = point(1, 5)
z.B = point(6, 2)
C.A = circle(from_radius(z.A, 4))
L.AB = line(z.A, z.B)
z.T = intersection(L.AB, C.A, {near = z.B})
C.B = circle(z.B, z.T)
L.TG1, L.TG2 = C.A:common_tangent(C.B, "external")
z.E, z.F = L.TG2:get()
PA.center, _, _ = C.A:CCL(C.B, L.TG2)
z.C = PA.center:get(1)
z.G = L.TG2:projection(z.C)
```



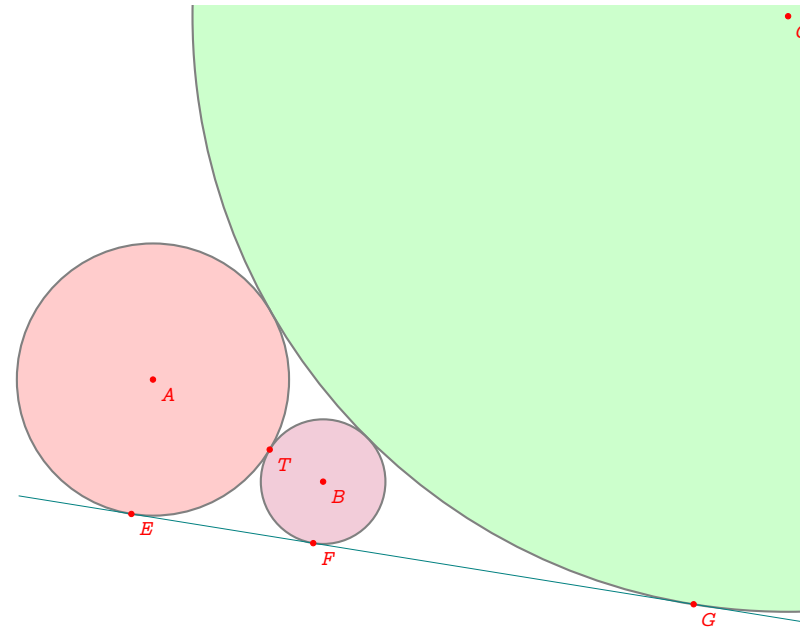
Example C:

File apo_CCLC.lua:

```

init_elements()
z.A = point(1, 5)
z.B = point(6, 2)
C.A = circle(from_radius(z.A, 4))
L.AB = line(z.A, z.B)
z.T = intersection(L.AB, C.A, {near = z.B})
C.B = circle(z.B, z.T)
L.TG1, L.TG2 = C.A:common_tangent(C.B, "external")
z.E, z.F = L.TG2:get()
PA.center, _, _ = C.A:CCL(C.B, L.TG2)
z.C = PA.center:get(2)
z.G = L.TG2:projection(z.C)

```



3.10 Case circle: method CCC(C2, C3 [, opts])

Purpose: Circles tangent to three given circles $C_1 = \text{self}$, C_2 , C_3 (up to 8 solutions).
Implementation follows Viète's reduction.

Syntax: pc, pt, n = C1:CCC(C2, C3 [, opts])

- **C1** — the current circle (self).
- **C2, C3** — other circle objects.
- **opts** — optional table of tolerances:
 - opts.abs (default 1e-4) — absolute tolerance.
 - opts.rel (default 1e-6) — relative tolerance.

Returns:

1. pc — path of solution centers,
2. pt — path of corresponding “through” points,
3. n — number of solutions found (0 to 8).

Result use in TikZ.

- With global paths: store `PA.center`, `PA.through`, then draw all circles using:
`\tkzDrawCirclesFromPaths(PA.center, PA.through).`
- With local paths: use `\tkz.nodes_from_paths(pc, pt)` to create nodes `w1,t1`, `w2,t2`, ...

Example A - Using global paths

```
\directlua{
  PA.center, PA.through, tkzNbCircles = C.A:CCC(C.B, C.C)
}
\tkzDrawCirclesFromPaths(PA.center, PA.through)
```

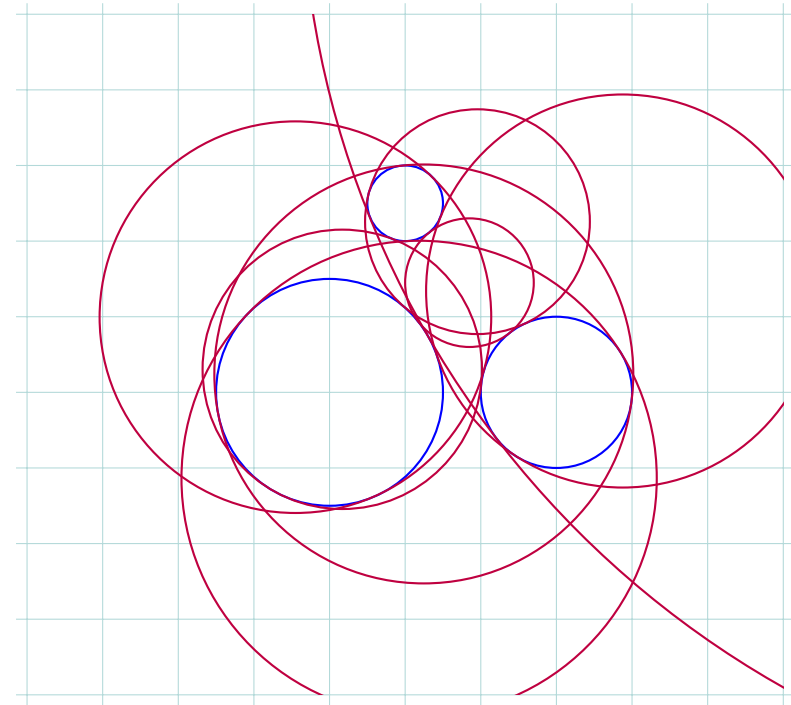
Example B - Using local paths

```
\directlua{
  local pc, pt, n = C.A:CCC(C.B, C.C)
  tkz.nodes_from_paths(pc, pt)
}
% circles are now (w1,t1), (w2,t2), ...
```

Example C

File apo_CCCC.lua:

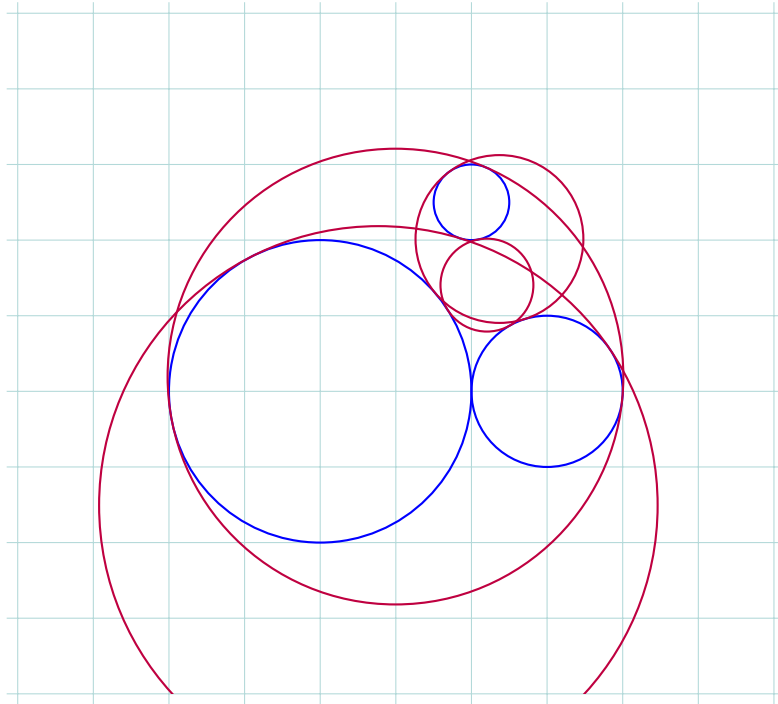
```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(2, 5)
z.a = z.A + point(3, 0)
z.b = z.B + point(2, 0)
z.c = z.C + point(1, 0)
C.Aa = circle(z.A, z.a)
C.Bb = circle(z.B, z.b)
C.Cc = circle(z.C, z.c)
PA.center, PA.through, n = C.Aa:CCC(C.Bb, C.Cc)
```



Example D

File apo_CCCD.lua:

```
init_elements()
z.A = point(0, 0)
z.B = point(6, 0)
z.C = point(4, 5)
z.a = z.A + point(4, 0)
z.b = z.B + point(2, 0)
z.c = z.C + point(1, 0)
C.Aa = circle(z.A, z.a)
C.Bb = circle(z.B, z.b)
C.Cc = circle(z.C, z.c)
pcenter, pthrough, n = C.Aa:CCC(C.Bb, C.Cc)
tkz.nodes_from_paths(pcenter, pthrough)
```



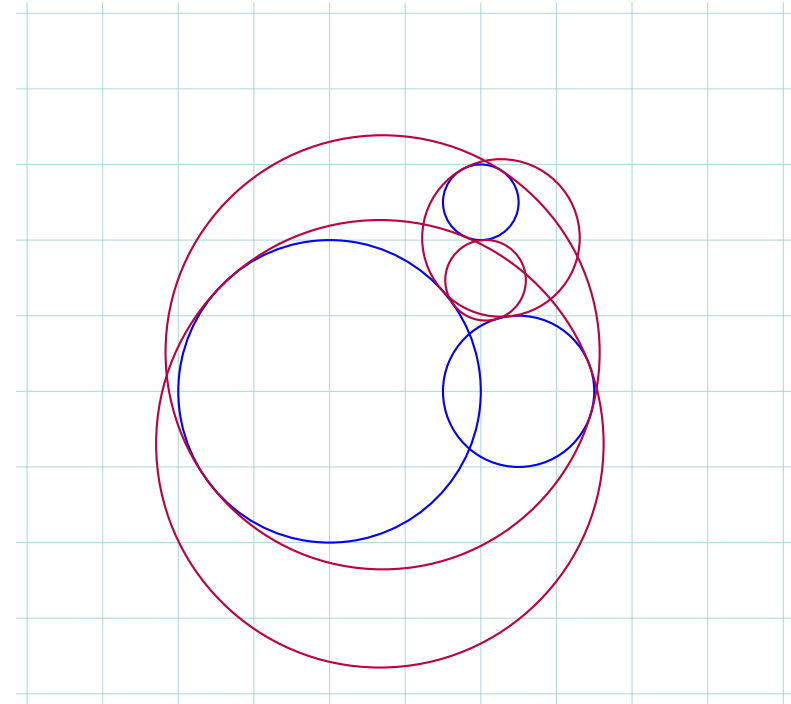
Example E

File apo_CCCE.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
z.C = point(4, 5)
z.a = z.A + point(4, 0)
z.b = z.B + point(2, 0)
z.c = z.C + point(1, 0)
C.Aa = circle(z.A, z.a)
C.Bb = circle(z.B, z.b)
C.Cc = circle(z.C, z.c)
PA.center, PA.through, n = C.Aa:CCC(C.Bb, C.Cc)

```



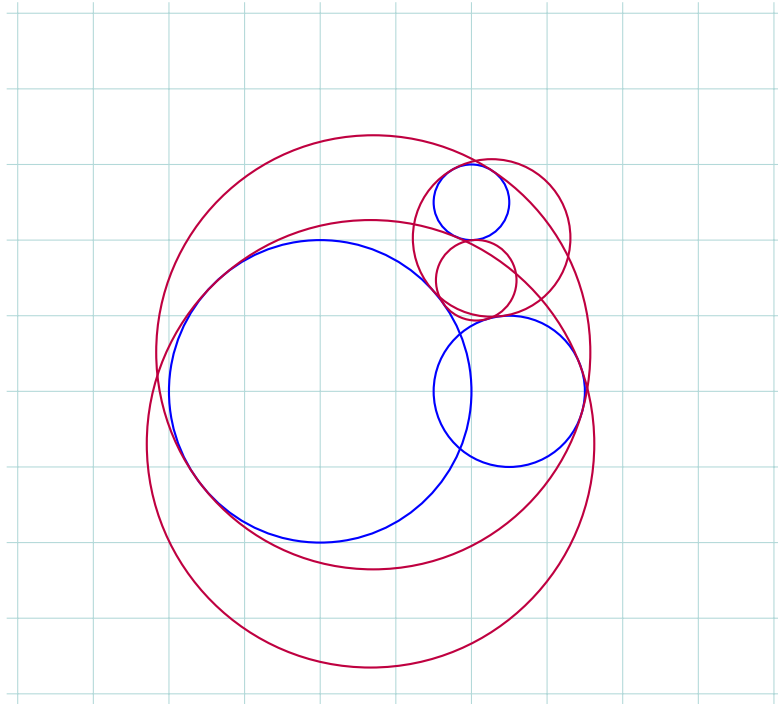
Example F

File apo_CCCE.lua:

```

init_elements()
z.A = point(0, 0)
z.B = point(5, 0)
z.C = point(4, 5)
z.a = z.A + point(4, 0)
z.b = z.B + point(2, 0)
z.c = z.C + point(1, 0)
C.Aa = circle(z.A, z.a)
C.Bb = circle(z.B, z.b)
C.Cc = circle(z.C, z.c)
PA.center, PA.through, n = C.Aa:CCC(C.Bb, C.Cc)

```



3.11 Class circle: method CCC_gergonne(C_2, C_3)

Purpose: Construct all circles tangent to three given circles using the classical Gergonne construction: centers of similitude, radical axis, poles, and contact points.
This method is only valid for circles that are disjoint in pairs.

Syntax:

`PA.center, PA.through, n = C1:CCC_gergonne(C2, C3 [, opts])`

Arguments:

- **C1** — the calling circle (self).
- **C2, C3** — two other circles.
- **opts.abs** — optional absolute tolerance for tangency tests (default = `tkz.epsilon`).

Returns:

- **PA.center** — a path storing the centers of all solution circles,
- **PA.through** — a path storing one contact point on each solution circle,

- **n** — the number of solutions found (may range from 0 to 8).
Each circle solution is represented by the pair:

$(PA.center[i], PA.through[i]), \quad i = 1, \dots, n.$

It can be drawn using

`\tkzDrawCirclesFromPaths(PA.center, PA.through).`

Geometric principle: The construction follows Gergonne:

1. Compute the external and internal centers of similitude of the three circles: J_{12}, J_{13}, J_{23} (external), I_{12}, I_{13}, I_{23} (internal).
2. These six points are aligned on four special lines: one line through the three external centers ("JJJ"), and three lines passing through one external and the two internal centers ("JII").
3. The radical center R of the three circles is computed (intersection of two radical axes).
4. For each of the four lines, compute its pole with respect to each circle. From R , draw lines to these poles; each such line intersects each circle (in general) in two points: these are candidate contact points.
5. For each triple of contact points (one on each circle), compute the circumcenter. This is a potential solution. Keep it only if it is tangent to all three circles (checked by **circle:act**).

Remarks:

- This method is purely geometric and works well when the circles are disjoint or only tangent. If the three circles intersect (secant configuration), some poles or intersections may be undefined, and no solution is returned.
- The method complements more robust algebraic approaches (Viète-based solvers).
- The returned paths can be directly used in a TikZ picture.

Example:

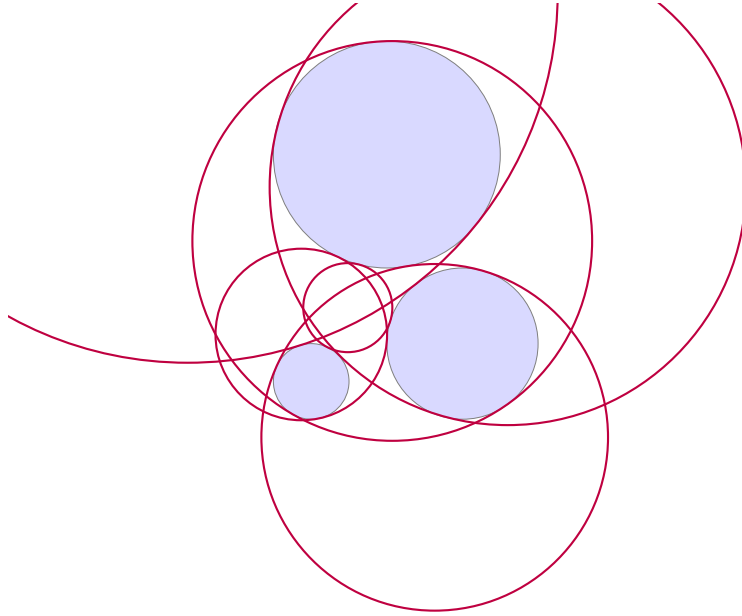
File `apo_gergonne.lua`:

```
init_elements()
z.A = point(0, 0)
z.a = z.A + point(1, 0)
z.B = point(4, 1)
z.b = z.B + point(2, 0)
z.C = point(2, 6)
```

```

z.c = z.C + point (3, 0)
C.Aa = circle(z.A, z.a)
C.Bb = circle(z.B, z.b)
C.Cc = circle(z.C,z.c)
PA.center, PA.through = C.Aa:CCC_gergonne(C.Bb, C.Cc)

```



```

C.Cc = circle(z.C,z.c)
L.rAB = C.Aa: radical_axis (C.Bb)
L.rAC = C.Aa: radical_axis (C.Cc)
L.rBC = C.Bb: radical_axis (C.Cc)
z.w = C.Aa :radical_center(C.Bb, C.Cc)
z.x1, z.y1 = L.rAB:get()
z.x2, z.y2 = L.rAC:get()
z.x3, z.y3 = L.rBC:get()
z.t = C.Aa:radical_circle(C.Bb,C.Cc).through
z.I = C.Aa:external_similitude(C.Bb)
z.J = C.Aa:external_similitude(C.Cc)
z.K = C.Bb:external_similitude(C.Cc)
L.IK = line(z.I, z.K)
z.P = C.Aa:pole(L.IK)
z.Q = C.Bb:pole(L.IK)
z.R = C.Cc:pole(L.IK)
z.a1, z.a2 = intersection_lc_(z.w, z.P, z.A, z.a)
z.b1, z.b2 = intersection_lc_(z.w, z.Q, z.B, z.b)
z.c1, z.c2 = intersection_lc_(z.w, z.R, z.C, z.c)
z.w1 = circum_center_(z.a1, z.b1, z.c2)
z.w2 = circum_center_(z.a2, z.b2, z.c1)

```

Explanations:

File apo_gergonne2.lua:

```

init_elements()
z.A = point(0, 0)
z.a = z.A + point (1, 0)
z.B = point (4, 1)
z.b = z.B + point (2, 0)
z.C = point (2, 6)
z.c = z.C + point (3, 0)
C.Aa = circle(z.A, z.a)
C.Bb = circle(z.B, z.b)

```

