The power of Sympy for programming in geometry

Ernest Scheiber*

Abstract

For didactic purposes, the capabilities of Sympy to solve synthetic geometrical problems are highlighted. There is exemplified the Apollonius problem: construct a circle tangent to other three circles. The problem is solved using inversion and code examples are provided.

2012 ACM Subject Classification: G.4, D.1.m
Key words: Sympy, programming for geometry, Apollonius problem.

1 Introduction

Teaching classical geometry takes advantages today of the computer technologies. Over the last few decades, several software programs have been developed to facilitate the study of geometry. One notable example is *Geogebra*, which is currently one of the leading interactive geometry software applications. It assists us in proving and / or verifying relations or statements, as well as generating high-quality geometric figures, among other capabilities.

We can also utilize *Sympy*, a Python package for symbolic computation. *Sympy* not only offers powerful tools for symbolic calculations but also includes classes for various geometry notions. This makes it a valuable resource for exploring and studying geometry concepts.

When comparing the usage of *Sympy* and *Geogebra*, the following pros and cons can be considered:

Pros of using *Sympy*:

- Geometric computations are based on symbolic computation (Computer Algebra System), providing more precise results compared to floating point algebra.
- Provides a straightforward way to manage resources efficiently.

*scheiber@unitbv.ro

- Allows the development of large programs, enabling complex geometric computations and analyses.
- Takes advantage of the extensive capabilities and versatility of the Python programming language.

Cons of using *Sympy*:

- Lack of an interactive interface compared to software like *Geogebra*, which may impact the ease of use and immediate visualization of geometric objects.
- Limited facilities for directly plotting geometric objects, requiring additional steps or tools for visualization.
- Steeper learning curve as it requires basic programming skills in Python, as well as knowledge of *Sympy* itself.

The purpose of this note is didactic. Its aim is to exemplify the power of *Sympy* in solving classical geometry problems, specifically the Apollonius problem of constructing a circle tangent to three given circles. The solution will be based on inversion with respect to a circle [1], [2], [4], [3]. The required properties of the inversion are recalled in the Appendix.

To plot the geometric figures, we have developed a simple plotting utility based on the *matplotlib* package.

2 The Apollonius problem

A problem is to find / draw circles tangent to three *things.* The *things* may be points, (straight) lines and circles. The required circles must be tangent to the

given circles and / or lines and pass through given points. It results 10 problems:

No.	The problem	Abbr.
1	Circle passing through three points	PPP
2	Circle tangent to three lines	LLL
3	Circle passing through two points and tangent to a line	PPL
4	Circle passing through a point and tangent to two lines	PLL
5	Circle tangent to two lines and a circle	LLC
6	Circle passing through a point and tangent to a line and a circle	PLC
7	Circle tangent to a line and two circles	LCC
8	Circle passing through two points and tangent to a circle	PPC
9	Circle passing through a point and tangent to two circles	PCC
10	Circle tangent to three circles	CCC

Let be the circles $\mathcal{C}(A, r_A), \mathcal{C}(C, r_C), \mathcal{C}(E, r_E)$. Depending on $\min\{r_A, r_C, r_E\}$ there are three cases to solve the CCC problem:

1. $r_A = r_C = r_E$;
2. $r_A > r_C = r_E$;
3. $r_A, r_C > r_E$.

The first case is simple. The circle passing through the centers of the circles is modified by adding or subtracting its radius. The result for the circles $\mathcal{C}((0,0),2), \mathcal{C}((5,5),2), \mathcal{C}((3,-3),2)$ are given in Fig. 1.

For the second case, let us suppose that the circle $\mathcal{C}(O, \rho)$ is a solution to the CCC problem (see Fig. 2). Then, the circle $\mathcal{C}(O, \rho - r_E)$ passes through the points C and E, and it is tangent to the circle $\mathcal{C}(A, r_A - r_E)$. We have obtained a PPC problem: finding the circle that passes through C and E and is tangent to the circle $\mathcal{C}(A, r_A - r_E)$. If we know a solution to this problem, denoted as $\mathcal{C}(O, r)$, then $\mathcal{C}(O, r + r_E)$ will be a solution to the initial problem. This method is known as the Viète method. It is also possible to find solutions starting with $\mathcal{C}(A, r_A + r_E)$. That is why it is crucial to check if the PPC solution also leads to a solution for CCC.

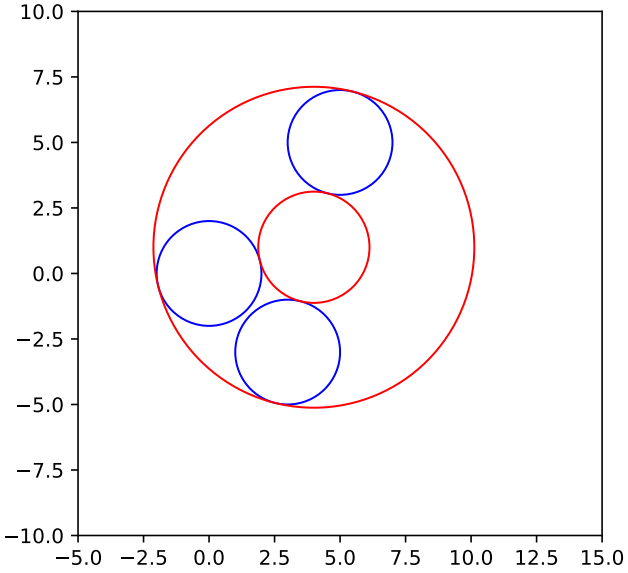


Fig. 1: Case 1.

Solving the PPC problem by inversion

Let $A(x_A, y_A)$, $B(x_B, y_B)$, $C(x_C, y_C)$, and $D(x_D, y_D)$ be the given points. The task is to construct a circle that passes through A and B and is tangent to the circle $\mathcal{C}_1 = \mathcal{C}(C, |CD|)$. Introducing an additional point E, we assume that the solution to the problem is the circle $\mathcal{C}_0$. To investigate this further, we consider the inversions with respect to the circle $\mathcal{C}(A, |AE|)$, which results in the following transformations:

- The point B and its inversion $B' = \text{Inv}(B)$.
- The circle $\mathcal{C}_1$ and its inversion $\mathcal{C}_1' = \text{Inv}(\mathcal{C}_1)$.
- The circle $\mathcal{C}_0$. Since $\mathcal{C}_0$ passes through A, which is the center of inversion, the transformed object is the line $d = \text{Inv}(\mathcal{C}_0)$.

It can be noted that d has a single common point with the circle $\mathcal{C}_1'$ and that $B' \in d$. The point at infinity is the common point between the inversions of A and of the circle $\mathcal{C}_0$. It results that

1. The line d is a tangent from B' to the circle $\mathcal{C}_0'$, and consequently

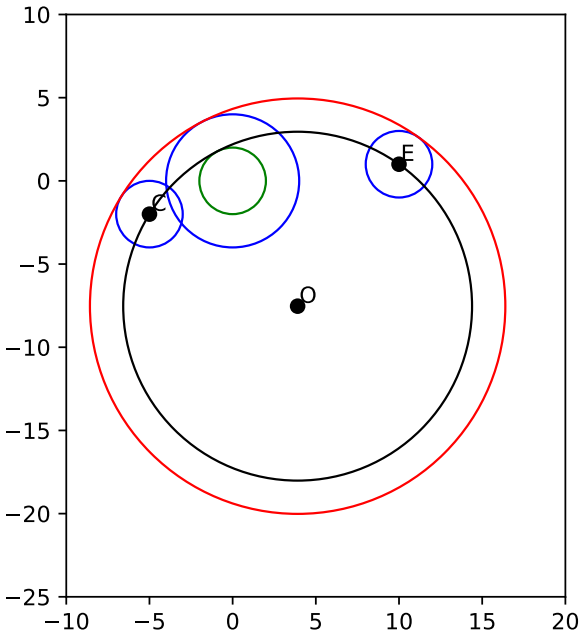


Fig. 2: Reasoning in Case 2

2. $\mathcal{C}_0$, the required circle, is the inversion of the line d.

The corresponding Python code is

```
import sympy as smp

def invPPC(xA,yA,xB,yB,xC,yC,xD,yD):
    A = smp.Point(xA,yA)
    B = smp.Point(xB,yB)
    C = smp.Point(xC,yC)
    D = smp.Point(xD,yD)
    E = smp.Point(xD+1,yD+1)
    c = smp.Circle(C,C.distance(D))
    B1 = inv(A,E,B)
    e = invC(A,E,C,D)

    rez = []
    if type(e) == smp.Circle:
        t = e.tangent_lines(B1)
        if t! = []:
            T10 = smp.intersection(e,t[0]);T1 = T10[0]
            T20 = smp.intersection(e,t[1]);T2 = T20[0]
            f = invL(A,E,B1,T2)
```

```
            g = invL(A,E,B1,T1)
            rez.append(f)
            rez.append(g)

    elif type(e) == smp.Line2D:
        g = e.parallel_line(B1)
        f = invL(A,E,g.p1,g.p2)
        rez.append(f)

    return rez
```

For the circles $\mathcal{C}((0,0),4), \mathcal{C}((10,1),2), \mathcal{C}((-5,-2),2)$ the solutions are given in Fig. 3.

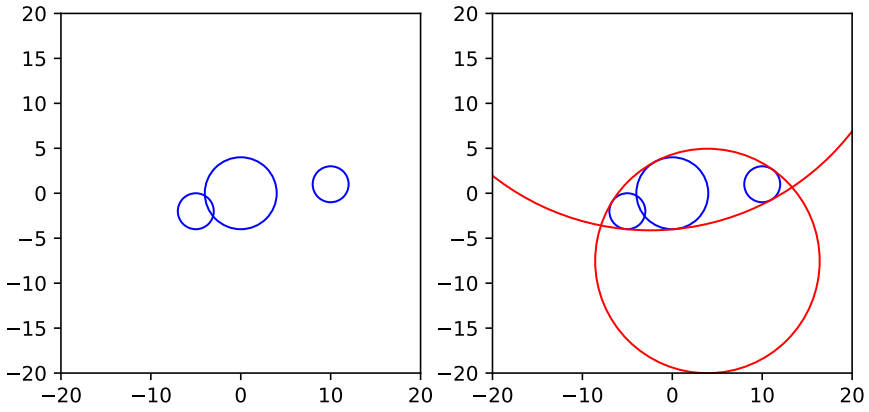


Fig. 3: Case 2.

With the Viète method the third case is reduced to a PCC problem.

Solving the PCC problem by inversion

Using similar notations, the circles are $\mathcal{C}_1 = \mathcal{C}(A, |AB|)$ and $\mathcal{C}_2 = \mathcal{C}(C, |CD|)$ and E is the point. There is required to construct a circle passing through E and being tangent to the circles $\mathcal{C}_1$, $\mathcal{C}_2$. With an additional point F and supposing that the solution of the problem is the circle $\mathcal{C}_0$ there is considered the inversions with respect to the circle $\mathcal{C}(E, |EF|)$ of

- The circles $\mathcal{C}_1, \mathcal{C}_2$: $\mathcal{C}_1' = \text{Inv}(\mathcal{C}_1), \mathcal{C}_2' = \text{Inv}(\mathcal{C}_2)$;

  It is possible that one or both of the circles $\mathcal{C}_1,\ \mathcal{C}_2$ to pass through E case in which its inversion is a line.

- The circle $\mathcal{C}_0$. Because $\mathcal{C}_0$ passes through E, the center of inversion, the result is the line $d = \text{Inv}(\mathcal{C}_0)$.

It can be noted that d has a single common point with the circles $\mathcal{C}_1'$ and $\mathcal{C}_2'$. The point at infinity is the common point between the inversions of E and of the circle $\mathcal{C}_0$. It results that

1. If $\mathcal{C}_1',\ \mathcal{C}_2'$ are circles then d is a common tangent of these circles.

   If $\mathcal{C}_1$ is a line (by example) then d is a tangent to $\mathcal{C}_2$ parallel to $\mathcal{C}_1$.

2. Consequently $\mathcal{C}_0$, the required circle, is the inversion of the line d.

We used the code

```
import sympy as smp

def invPCC(xA,yA,xB,yB,xC,yC,xD,yD,xE,yE):
    A  =  smp.Point(xA,yA)
    B  =  smp.Point(xB,yB)
    C  =  smp.Point(xC,yC)
    D  =  smp.Point(xD,yD)
    E  =  smp.Point(xE,yE)
    F  =  smp.Point(xE+1,yE+1)

    c  =  smp.Circle(A,A.distance(B))
    d  =  smp.Circle(C,C.distance(D))
    e  =  invC(E,F,A,B)
    f  =  invC(E,F,C,D)

    if type(e) == smp.Line2D and (type(f) == smp.Line2D or type(f) == smp.Segment2D):
         if e.slope == f.slope:
                print('There exists an infinity number of solutions')
                s0=c.tangent_lines(E)
            print('There exists an infinity number of solutions')
        else:
            print('The problem does not have solution')
            s0=[]

        t = []

    if (type(e) == smp.Line2D or type(e) == smp.Segment2D) and type(f) == smp.Circle:
        t = parallelTangents(f,e)[0]

    if type(e) == smp.Circle and (type(f) == smp.Line2D or type(f) == smp.Segment2D):
        t = parallelTangents(e,f)[0]

    if type(e) == smp.Circle and type(f) == smp.Circle:
        M  =  e.center
        P  =  smp.Point2D(M.x+e.radius,M.y)
```

```
        N  =  f.center
        Q  =  smp.Point2D(N.x+f.radius,N.y)
        t  =  tang(M,P,N,Q);

    rez = []
    for i in range(len(t)):
        h = t[i]
        P  =  smp.Point2D(smp.N(h.p1.x),smp.N(h.p1.y))
        Q  =  smp.Point2D(smp.N(h.p2.x),smp.N(h.p2.y))
        s  =  invL(E,F,P,Q)
        rez.append(s)

    if t == [] and len(s0) > 0:
        rez.append(s0[0])

    return rez
```

In the code provided, the function tang calculates all the common tangents of the circles $\mathcal{C}(A, |AB|)$ and $\mathcal{C}(C, |CD|)$, while the function parallelTangents calculates the tangents to the circle $\mathcal{C}(A, |AB|)$ that are parallel to the line CD. The second Appendix contains the code of the function tang.

The solutions of the CCC problem for the circles $\mathcal{C}((-3, 2), \sqrt{13}), \mathcal{C}((4, 6), 2\sqrt{2})$, $\mathcal{C}((0, -2), 1)$ are given in Fig. 4.

3 Final remarks

Sympy is a powerful tool for synthetic and analytic geometry. It allows the realization of geometric constructions, proofs, and problem solving.

When the final purpose is to plot some geometric figures, at some level of computation we turn to floating point arithmetic. The effect was a drastic reduction of the computation time.

To make the results reproducible we provide the full code at https://github.com/e-scheiber/sympy_geometry.git.

References

[1] Bakel'man I. Ya., 1974, *Inversions*. The Univ. of Chicago Press, Chicago. 1

[2] Henderson W. D., Taimina D., 2020, *Inversions in Circles* - Ch. 16. Experiencing Geometry - Project Euclid, DOI: 10.3792/euclid/9781429799850-20. 1, A

[3] Mihăileanu N.N., 1965, *Complements of synthetic geometry.* Ed. Didactică şi Pedagogică, Bucureşti (Romanian). 1

[4] Pamfilos P., Inversion. http://http://users.math.uoc.gr/~pamfilos/eGallery/problems/Inversion.pdf. 1

[5] * * *, 2023, *SymPy.* https://www.sympy.org/en/index.html.

A Inversions in circles

If $\mathcal{C}(A, R)$ is cercle with center in the point A and R is its radius the points (C, D) is an inversive pair if

1. the points are on the same ray with the starting point A;
2. $|AC| \cdot |AD| = R^2$ ($|AC|$ and $|AD|$ represent the distance between the endpoints of each segment).

The point A is named the center of inversion.

The *Sympy* function to compute the inverse of the point C with respect to the circle $\mathcal{C}(A, R)$, where $R = AB$, is

```
def inv(A,B,C):
     R = A.distance(B)
     d = A.distance(C)
     return A+(R/d)**2*(C-A)
```

We recall the following properties [2]:

Theorem A.1 *If (C, D) is an inversive pair with respect to the circle $\mathcal{C}(A, R)$ then any circle passing through the points C and D is orthogonal to the circle $\mathcal{C}(A, R)$.*

Theorem A.2 *An inversion takes a circle through the center of inversion to a line not through the center, and vice versa.*

The *Symply* function to obtain the inversion of the line CD

```
import sympy as smp

def invL(A,B,C,D):
    C1 = inv(A,B,C)
    D1 = inv(A,B,D)
    return smp.Circle(A,C1,D1)
```

Theorem A.3 *An inversion takes circles not through the center of inversion to circles not through the center.*

The *Symply* function to obtain the inversion of the circle $\mathcal{C}(C, |CD|)$ is

```
import sympy as smp

def invC(A,B,C,D):
   r1 = A.distance(B)
   r2 = C.distance(D)
   d = smp.Circle(C,r2)
   E0 = smp.intersection(d,smp.Line(A,C));E1 = E0[0];E2 = E0[1]
   s = smp.Segment(E1,E2)
   F0 = smp.intersection(d,s.perpendicular_bisector());F1 = F0[0];F2 = F0[1]
   P = inv(A,B,F1)
   Q = inv(A,B,F2)
   if(A.distance(C) == r2):
       return smp.Line(P,Q)
   else
       return smp.Circle(P,Q,self.inv(A,B,D)
```

B The common tangent of two circles

Two cases must be taken into account: the circles have or not the same radius.

```
def tang(A,B,C,D):
    c  =  smp.Circle(A,A.distance(B))
    d  =  smp.Circle(C,C.distance(D))
    r1  =  c.radius
    r2  =  d.radius
    r  =  abs(r1-r2)
    if r !=  0:
        if r1<r2:
            e  =  smp.Circle(C,r)
        else:
            e  =  smp.Circle(A,r)

        f  =  smp.Circle((A+C)/2,A.distance(C)/2)
        E0  =  smp.intersection(f,e);
        if len(E0) == 0:
            return []
        else:
            E  =  E0[0]

        if r1<r2:
            F0  =  smp.intersection(d,smp.Ray(C,E));F  =  F0[0]
            g0  =  d.tangent_lines(F);g  =  g0[0]
        else:
            F0  =  smp.intersection(c,smp.Ray(A,E));F  =  F0[0]
            g0  =  c.tangent_lines(F);g  =  g0[0]

        h  =  g.reflect(smp.Line(A,C))
        if r1+r2 < A.distance(C):
            r  =  r1/(r1+r2)
            G0  =  smp.intersection(smp.Segment(A,C),smp.Circle(A,r*A.distance(C)));G  =  G0[0]
            l  =  d.tangent_lines(G)
            return [g,h,l[0],l[1]]
        elif r1+r2 == A.distance(C):
```

```
        E0 = smp.intersection(smp.Segment(A,C),c);
        p=c.tangent_lines(E0[0])
        return [g,h,p[0]]
    else:
        return [g,h]

else:
    E0  =  smp.intersection(c,smp.Line(A,C).perpendicular_line(A))
    p  =  c.tangent_lines(E0[0])
    q  =  c.tangent_lines(E0[1])
    if r1+r2 < A.distance(C):
        r  =  r1/(r1+r2)
        G0  =  smp.intersection(smp.Segment(A,C),smp.Circle(A,r*A.distance(C)));G  =  G0[0]
        l  =  d.tangent_lines(G)
        return [p[0],q[0],l[0],l[1]]
    elif r1+r2 == A.distance(C):
        E0 = smp.intersection(smp.Segment(A,C),c);
        h=c.tangent_lines(E0[0])
        return [p[0],q[0],h[0]]
    else:
        return [p[0],q[0]]
```

Corresponding to the cases we get the images from Fig. 5 and Fig. 6.

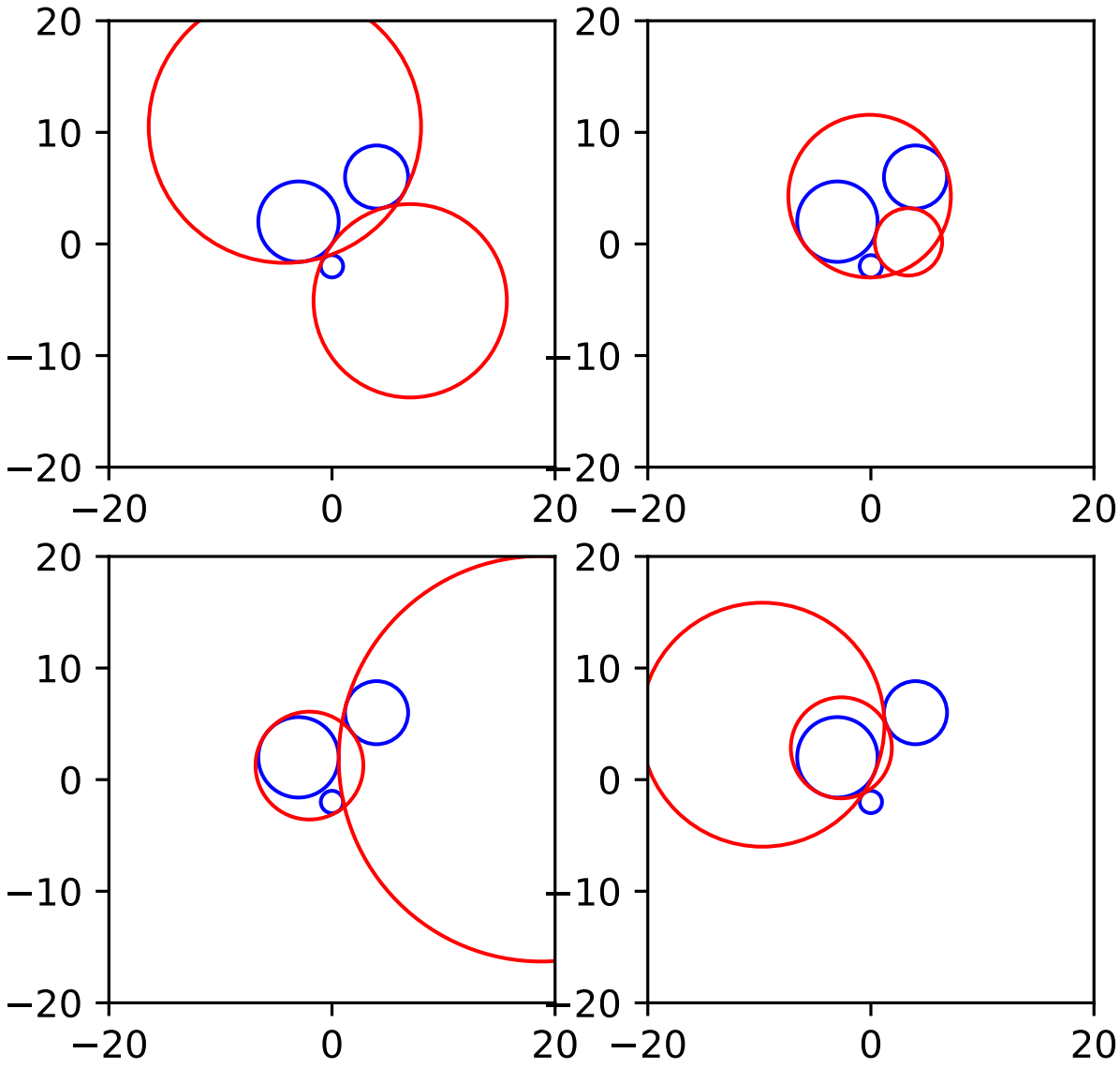
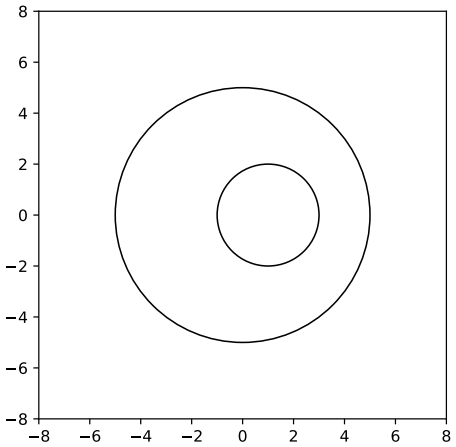
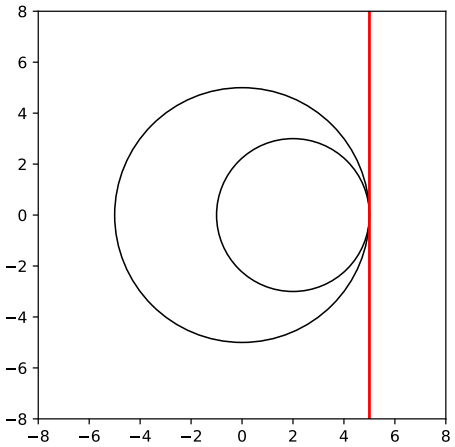


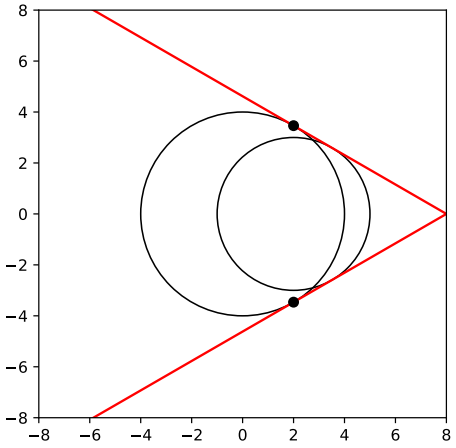
Fig. 4: Case 3.



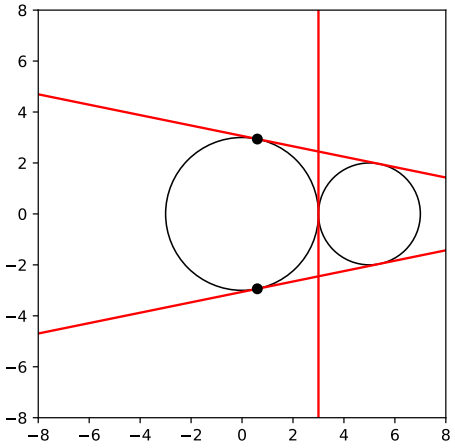
(a) Number of tangents is 0



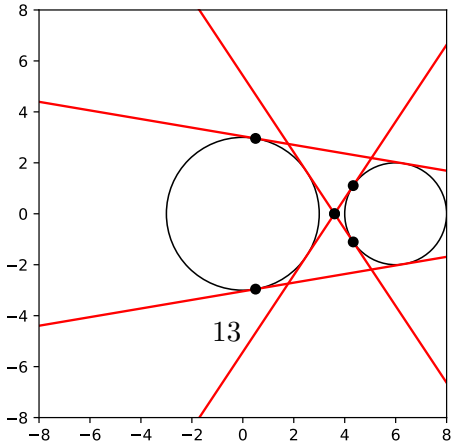
(b) Number of tangents is 1



(c) Number of tangents is 2

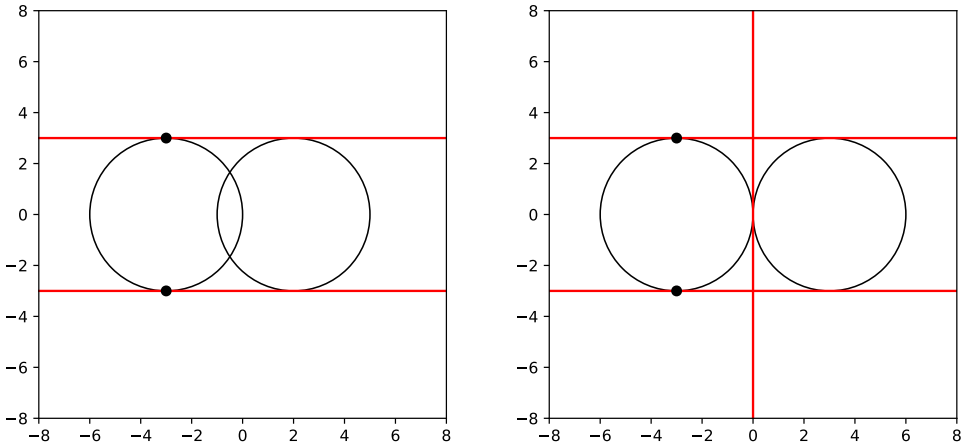


(d) Number of tangents is 3



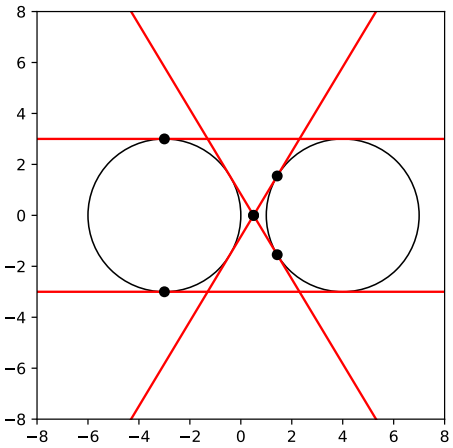
(e) Number of tangents is 4

Fig. 5: Common tangents of two circles with unequal radiuses



(a) Number of tangents is 2

(b) Number of tangents is 3



(c) Number of tangents is 4

Fig. 6: Common tangents of two circles with equal radiuses