Catam 2.3 Non-Euclidean Geometry

April 2023

1 Basic Geometry

1.1 Spherical Geometry

A spherical line in $\mathbb{C}_\infty$ is defined as either a straight line through the origin or $\{z \in \mathbb{C} \mid |z-a|^2 = |a|^2 + 1,\ a \in \mathbb{C}\}$. Let $z_i = x_i + iy_i$ and $a = b + ci$. We find though some algebra that the following system of equations is equivalent to a real valued linear problem.

$$\begin{aligned} |z_1 - a|^2 &= |a|^2 + 1 \\ |z_2 - a|^2 &= |a|^2 + 1 \end{aligned}$$

$$\begin{aligned} |z_1|^2 - 1 &= 2x_1 b + 2y_1 c \\ |z_2|^2 - 1 &= 2x_2 b + 2y_2 c \end{aligned}$$

We may write this as $B = MA$ with A the $\mathbb{R}^2$ form of the complex number a, $A = (b, c)^T$. For $\det(M) \neq 0$ the problem is solved with $A = M^{-1}B$. If $\det(M) = 0$ then that implies the vectors $(x_1, y_1)^T$ and $(x_2, y_2)^T$ are colinear. Or in other words $z_2 = \lambda z_1$ for $\lambda \in \mathbb{R}$ or vice versa. It can then be shown that there are three cases:

- $\lambda = 1$ this case can be safely ignored as we assume z_1, z_2 distinct.
- $\lambda = \frac{1}{|z_1|^2}$. This corresponds to selecting z_1 and z_2 antipodal points so has infinite solutions. We will use the obvious straight line solution in this case.
- λ neither. This has no solution for $a \in \mathbb{C}$ however we have an unique spherical line being the straight line through z_1, z_2.

A program called "SphericalTriangles.py" has been created and can be found in the Programs section which draws spherical triangles.

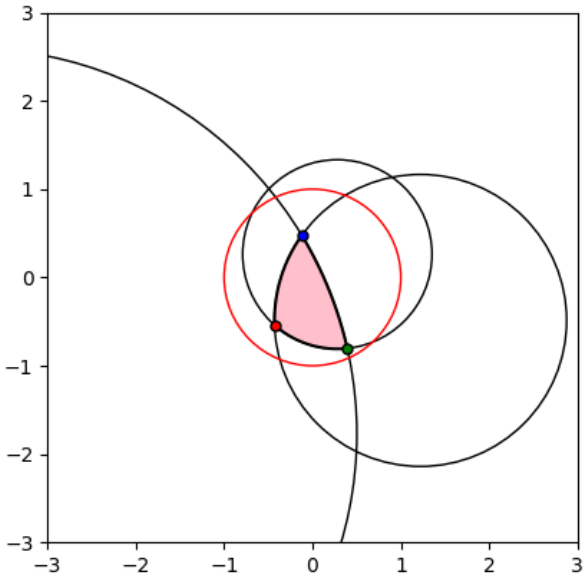


Figure 1: A random Spherical Triangle

A quick inspection of the diagram shows us that all the spherical lines (black circles) intersect the unit circle at antipodal points.

We can justify this by considering the points $z = \pm ia/|a|$. Taking $a = re^{i\theta}$

$$|z - a|^2 = |(i - r)e^{i\theta}| = 1 + r^2 = 1 + |a|^2$$

If any other point on the spherical lies on the unit circle then the spherical line is in fact the unit disc.

This makes complete sense when thinking in $3D$ as spherical lines are just projections of great circles which intersect the equator (the unit circle) at antipodal points.

1.2 Hyperbolic Geometry

The computations for hyperbolic lines are very similar to those for spherical lines, also as points in the hyperbolic disc are limited to the interior of the unit circle the only case for degeneracy is when the points are colinear to the origin, where there is exactly one solution, being the straight line between them.

A program called "HyperbolicTriangles.py" has been created and can be found in the Programs section which draws spherical triangles.

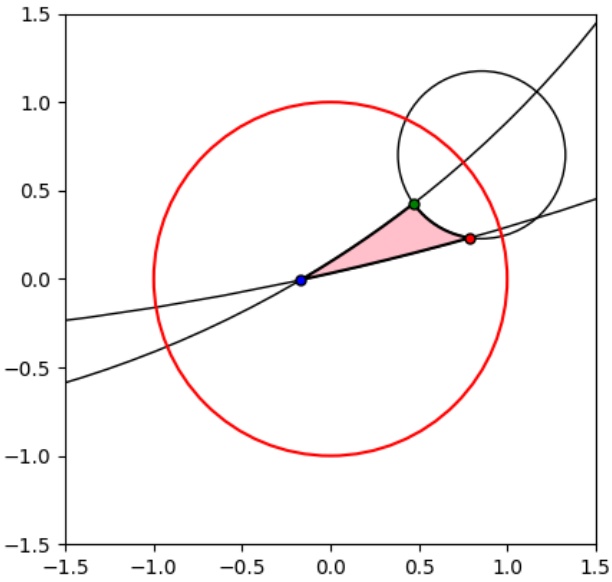
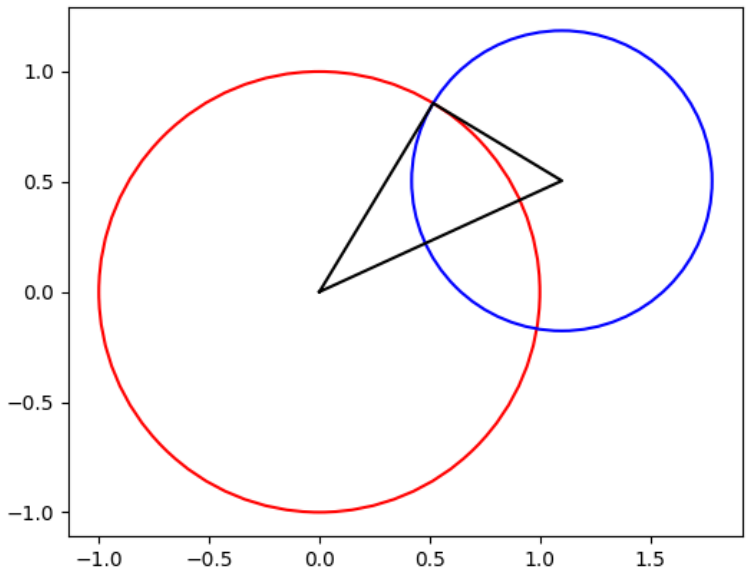


Figure 2: A random Hyperbolic Triangle

By looking at the hyperbolic lines on the preceding diagram we see they intersect $|z| = 1$ perpendicular to it.

To produce a proof for the hyperbolic lines being perpendicular to the unit circle let us construct a simple diagram.



We note the red circle as the unit circle and the blue as a hyperbolic line.

We assume the centre of the blue circle is a distance r away from the origin. And by definition of a hyperbolic line, the blue circle has radius $\sqrt{r^2-1}$

Now using the cosine rule:

$$r^2 = r^2 - 1 + 1 - 2\sqrt{r^2-1}cos(\theta)$$
$$\implies cos(\theta) = 0 \implies \theta = \frac{\pi}{2}$$

1.3 Regular Polygons

Recall the area of a spherical triangle is given by $A = \alpha+\beta+\gamma-\pi$ Using a central point at the origin in a regular n-gon of internal angle $2\pi/3$ we may construct $2n$ spherical triangles with internal angles $\pi/2, \pi/3, \pi/n$, demonstrated in the below diagram. It is then clear that the total area of the n-gon is $2n(\pi/2 + \pi/3 + \pi/n - \pi) = \pi(6-n)/3$. Clearly this means there are only regular n-gons of this form for $n = 3, 4, 5$.

To calculate r we use the following formula, with $a = 2tan^{-1}(r)$ and the triangle with angles $\pi/2, \pi/3, \pi/n$.

$$cos(\alpha) + cos(\beta)cos(\gamma) = sin(\beta)sin(\gamma)cos(a)$$

$$r = tan(\frac{1}{2}arccos(cot(\pi/3)cot(\pi/n)))$$

If we instead consider hyperbolic triangles in the same way, we instead use the formula $A = \pi - \alpha - \beta - \gamma$ which gives total area $\pi(n-6)/3$. Clearly this means we have regular hyperbolic n-gons for $n > 6$. We also get an analagous formula for r.

$$r = tanh(\frac{1}{2}arcosh(cot(\pi/3)cot(\pi/n)))$$

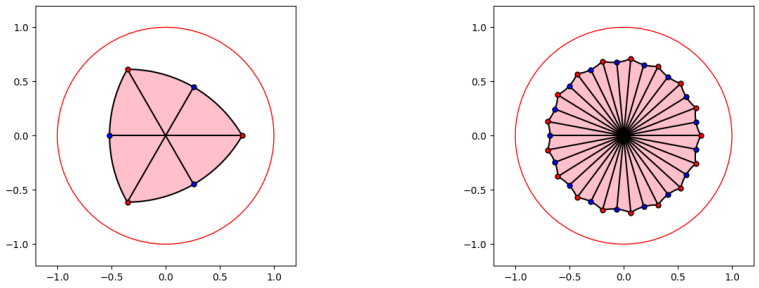


Figure 3: Regular triangle and 17-gon

The above trianges were produced by programs "SphericalRoots.py" and "HyperbolicRoots.py" respectively. They are found in the Programs section.

2 Symmetry Groups

2.1 Elementary Transformations

Allow us to first create a program which draws out triangles after transforming them via one of the three elementary transformations. A program called "TriangleInversion.py" can be found in the Programs section which does just this. Now, iterating each of these transformations and drawing it out, it is very clear that like the group elements, they have orders of $2, 3$, and p respectively.

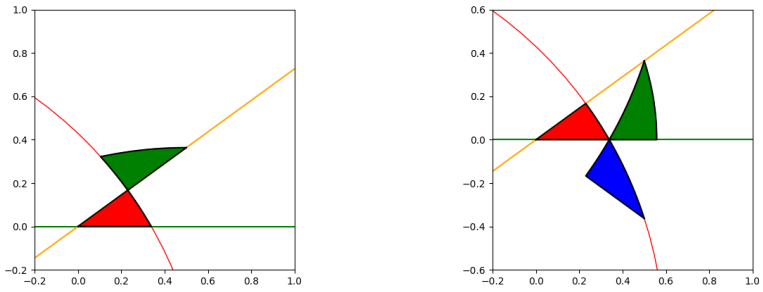


Figure 4: S_1 and S_2

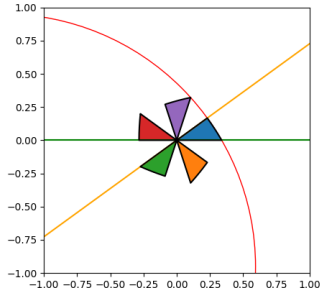


Figure 5: S_3

This becomes more obvious when we consider S_3 specifically, noting it is simply a rotation by $2\pi/p$ radians about the origin. By considering a Mobius transformation fixing the spherical and hyperbolic lines we invert in to lines through the origin we note an important result from $O(2)$ giving a composition of two reflections is a rotation. Thus geometrically all of our transformations are essentially rotations about points with angles $\pi, 2\pi/3$, and $2\pi/p$, this finally yields us a proper justification as to why these transformations have their appropriate orders.

2.2 Admissible Triangles

By using a method of generating the group $PSL(2,p)$ and then using the first found composition of generators for each element, it is possible to construct an admissible triangle corresponding to any group element. This is done in the program "SymmetryGroups.py" found in the Programs section.

Before starting let us first call the group of transformations generated by S_i as $G(p)$. By definition of admissible triangles this group acts faithfully and transitively on the set of admissible triangles.

2.2.1 p = 3

A plot of the all the triangles corresponding to a group element have been plotted below alongside a plot of a randomly chosen triangle.

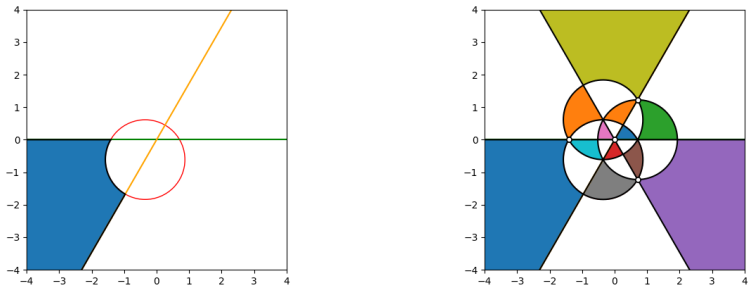


Figure 6: $p = 3$ Triangles and guidelines

We note there are 12 elements of the group $PSL(2,3)$ and 12 triangles each clearly with no overlap. Moreover under each of the three reflections, the admissible triangles are all sent to the empty space on the diagram, that is triangles with an inverse orientation to the original. It is most clear under R1, and R2 but a close inspection sees R3 follows the same pattern.

This means that as we cannot hit any other triangles under the action of $PSL(2,3)$ and as we already cover half of the sphere, $12(\pi/3 - \pi/6) = 2\pi$.

All this discussion hints to the underlying statement that the group homomorphism between $PSL(2,3)$ and $G(3)$ is in fact an isomorphism.

$$\phi : G(3) \to PSL(2,3)$$

$$\phi(S_i) \to \sigma_i$$

This follows immediately by noting we have 12 elements of both groups and the map is surjective. Then by the First Isomorphism Theorem and Lagrange's Theorem we get

$$G(3) \cong PSL(2,3)$$

2.2.2 p = 5

Again a plot of all the triangles corresponding to a group element are plotted below.

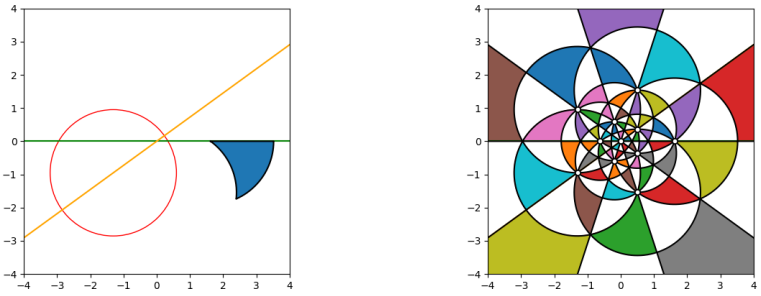
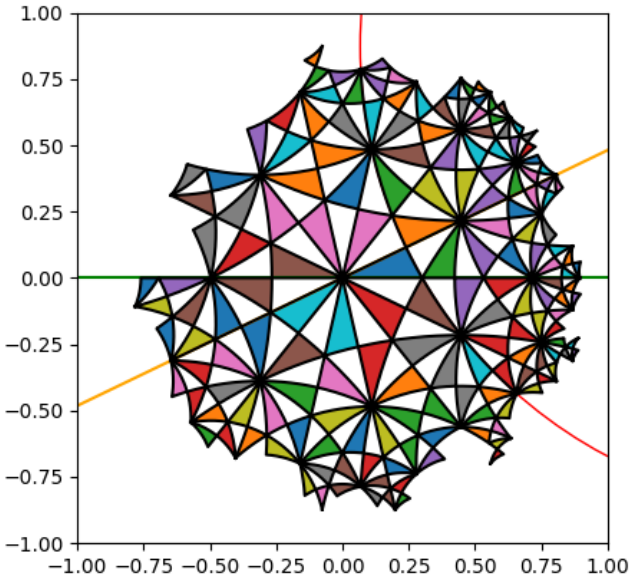


Figure 7: $p = 3$ Triangles and guidelines

The only real difference between this case and $p = 3$ is that we have 60 elements and 60 triangles. All the other considerations we made are completely analagous if not outright the same as before.

2.2.3 p = 7

Here we get a divergence from the previous spherical cases, first by noting that the hyperbolic disc has infinite area, but our triangles each have area $\pi/6 - \pi/7 = \pi/42$. So unlike before we can never tile the hyperbolic plane with a finite number of triangles. Thus it is clear we could always generate more triangles.



Depicted above it is clear that the 168 elements of $PSL(2,7)$ do not tile the hyperbolic disc.

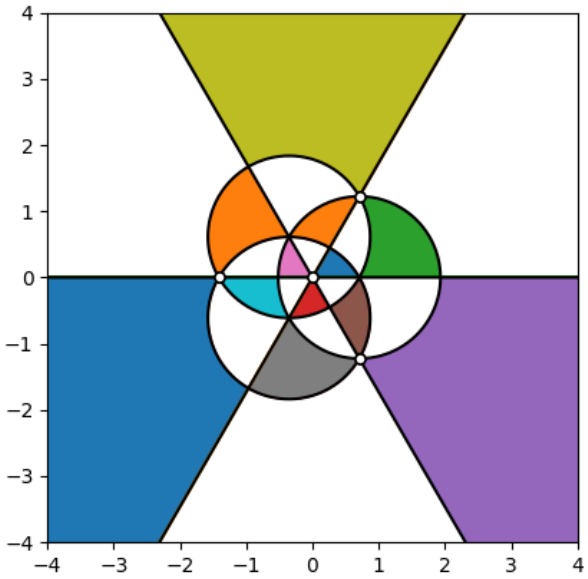
Though we lose a direct isomorphism we still have some relationship between our two groups.

$$G(7)/Ker(\phi) \cong PSL(2,7)$$

2.3 Platonic Solids

2.3.1 $p = 3$

We can note immediate similarities to the group of rotations of the tetrahedron, A_4.



If we look at the diagram and consider the white circles, that is where the vertex at the origin is sent to. We see the rotations act transitively upon the white circles. Associate these vertices with the vertices of a tetrahedron.

Now to fully specify a rotation of a tetrahedron we need to specify just two distinct vertices. One of the vertices we will assume to be the origin. We then see that wherever the triangle's origin lands there is a choice of exactly three admissible triangles. Each one of these fixing the remaining vertices. It is now clear to see that each element of $G(3)$ corresponds uniquely to a rotation of a tetrahedron, and so:

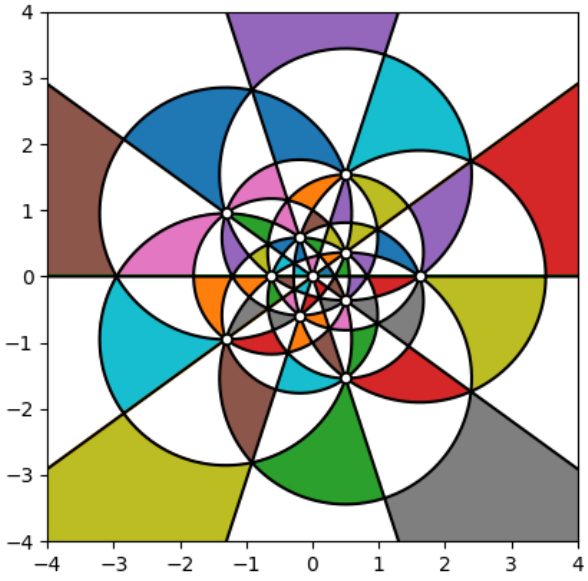
$$PSL(2,3) \cong G(3) \cong A_4$$

Moreover the rotations of the sphere in $3D$ under A_4 are actually the exact same maps as in $G(3)$ just under the stereographic projection.

2.3.2 $p = 5$

By a completely analogous argument, (noting only that there is an extra white dot at infinity). By examining the following picture we can also associate the symmetry group of rotations of an icosahedron (or dodecahedron) A_5 to $G(5)$

$$PSL(2,5) \cong A_5$$



3 Programs

3.1 SphericalTriangles.py

```
import matplotlib.pyplot as plt
import numpy
import random


def CrossProd (x1, y1, x2, y2):
    return x1*y2 - x2*y1


def ComputeCircle (x1, y1, x2, y2):

    if (x1 == None or x2 == None or y1 == None or y2 == None): return None, None, None

    b1 = x1*x1 + y1*y1 - 1
    b2 = x2*x2 + y2*y2 - 1

    det = 4*CrossProd(x1, y1, x2, y2)

    if (det == 0): return None, None, None

    a1 = (2*y2*b1 - 2*y1*b2)/det
    a2 = (-2*x2*b1 + 2*x1*b2)/det

    r = numpy.sqrt(1 + a1*a1 + a2*a2)

    return a1, a2, r


def ComputeArgFromCentre (x, y, b, c):
    X = x - b
    Y = y - c

    if (X == 0):
        if (Y == 0): return 0
        if (Y > 0): return numpy.pi/2
        if (Y < 0): return numpy.pi/2

    t = numpy.arctan(Y/X)

    if (X < 0):
        if (Y >= 0): t += numpy.pi
        else: t -= numpy.pi
```

```
    return t


def GenerateLinePoints (x1, y1, x2, y2):
    n = 100

    if (x1 != None and x2 != None and y1 != None and y2 != None):
        x = [x1 + (x2 - x1)*i/n for i in range(n+1)]
        y = [y1 + (y2 - y1)*i/n for i in range(n+1)]
    else:
        if (x1 != None):
            x = [x1*(1+i) for i in range(n+1)]
            y = [y1*(1+i) for i in range(n+1)]
        else:
            x = [x2*(n+1-i) for i in range(n+1)]
            y = [y2*(n+1-i) for i in range(n+1)]

    return x, y


def GenerateCirclePoints (b, c, r, x1, y1, x2, y2):

    # If Line then do line computations
    if (b == None): return GenerateLinePoints(x1, y1, x2, y2)

    # Generate points on the circle from z1 to z2
    a1 = ComputeArgFromCentre(x1, y1, b, c)
    a2 = ComputeArgFromCentre(x2, y2, b, c)

    # Use the cross product to determine which way to go, clock or anticlock.

    d = CrossProd(x1, y1, x2, y2)
    flag = numpy.sign(d)
    if (a2 < a1):
        a2 += 2*numpy.pi
    if (flag == -1):
        a2 -= 2*numpy.pi

    n = 100

    t = [a1 + (a2-a1)*i/n for i in range(n+1)]
    x = [b + r*numpy.cos(t[i]) for i in range(n+1)]
    y = [c + r*numpy.sin(t[i]) for i in range(n+1)]

    return x, y
```

```
def DrawSphericalTriangle (x1, y1, x2, y2, x3, y3):
    # If both are null we assume that means infinity

    # Generate Appropriate data of circles
    a1, a2, r1 = ComputeCircle(x1, y1, x2, y2)
    b1, b2, r2 = ComputeCircle(x2, y2, x3, y3)
    c1, c2, r3 = ComputeCircle(x3, y3, x1, y1)

    fig = plt.figure()
    ax = fig.add_subplot(aspect = 'equal')


    circle1 = plt.Circle((a1, a2), r1, edgecolor = 'black', facecolor = "none")
    circle2 = plt.Circle((b1, b2), r2, edgecolor = 'black', facecolor = "none")
    circle3 = plt.Circle((c1, c2), r3, edgecolor = 'black', facecolor = "none")

    ax.add_patch(circle1)
    ax.add_patch(circle2)
    ax.add_patch(circle3)


    # Generate Points
    p1, q1 = GenerateCirclePoints(a1, a2, r1, x1, y1, x2, y2)
    p2, q2 = GenerateCirclePoints(b1, b2, r2, x2, y2, x3, y3)
    p3, q3 = GenerateCirclePoints(c1, c2, r3, x3, y3, x1, y1)

    x = p1 + p2 + p3
    y = q1 + q2 + q3

    plt.plot(x, y, color = "black")
    ax.fill(x,y, color = "pink")

    # One error about filling inside not outside when shape includes infinity
    # This is only thing that really needs to be finished up

    circle = plt.Circle((0, 0), 1, edgecolor = 'red', facecolor = "none")
    ax.add_patch(circle)

    # This is very bad practise however the bodge reigns supreme.
    try:
        plt.plot(x1, y1, marker="o", markersize = 5, markerfacecolor = "red", markeredgecolo
    except:
        pass
    try:
```

```
        plt.plot(x2, y2, marker="o", markersize = 5, markerfacecolor = "green", markeredgeco
    except:
        pass
    try:
        plt.plot(x3, y3, marker="o", markersize = 5, markerfacecolor = "blue", markeredgecol
    except:
        pass

    ax.set_xlim((-3, 3))
    ax.set_ylim((-3, 3))

    plt.show()


def GenerateRandomTriangle():
    x = [random.uniform(-1, 1) for i in range(6)]
    DrawSphericalTriangle(x[0], x[1], x[2], x[3], x[4], x[5])
```

3.2 HyperbolicTriangles.py

```
import matplotlib.pyplot as plt
import numpy
import random


def CrossProd (x1, y1, x2, y2):
    return x1*y2 - x2*y1


def ComputeCircle (x1, y1, x2, y2):

    if (x1 == None or x2 == None or y1 == None or y2 == None): return None, None, None

    b1 = x1*x1 + y1*y1 + 1
    b2 = x2*x2 + y2*y2 + 1

    det = 4*CrossProd(x1, y1, x2, y2)

    if (det == 0): return None, None, None

    a1 = (2*y2*b1 - 2*y1*b2)/det
    a2 = (-2*x2*b1 + 2*x1*b2)/det

    r = numpy.sqrt(a1*a1 + a2*a2 - 1)

    return a1, a2, r


def ComputeArgFromCentre (x, y, b, c):
    X = x - b
    Y = y - c

    if (X == 0):
        if (Y == 0): return 0
        if (Y > 0): return numpy.pi/2
        if (Y < 0): return numpy.pi/2

    t = numpy.arctan(Y/X)

    if (X < 0):
        if (Y >= 0): t += numpy.pi
        else: t -= numpy.pi

    return t
```

```
def GenerateLinePoints (x1, y1, x2, y2):
    n = 100

    x = [x1 + (x2 - x1)*i/n for i in range(n+1)]
    y = [y1 + (y2 - y1)*i/n for i in range(n+1)]

    return x, y


def GenerateCirclePoints (b, c, r, x1, y1, x2, y2):

    # If Line then do line computations
    if (b == None): return GenerateLinePoints(x1, y1, x2, y2)

    # Generate points on the circle from z1 to z2
    a1 = ComputeArgFromCentre(x1, y1, b, c)
    a2 = ComputeArgFromCentre(x2, y2, b, c)

    # Use the cross product to determine which way to go, clock or anticlock.

    d = CrossProd(x1, y1, x2, y2)
    flag = numpy.sign(d)
    if (a2 < a1):
        a2 += 2*numpy.pi
    if (flag == 1):
        a2 -= 2*numpy.pi

    n = 100

    t = [a1 + (a2-a1)*i/n for i in range(n+1)]
    x = [b + r*numpy.cos(t[i]) for i in range(n+1)]
    y = [c + r*numpy.sin(t[i]) for i in range(n+1)]

    return x, y


def DrawHyperbolicTriangle (x1, y1, x2, y2, x3, y3):
    # If both are null we assume that means infinity

    # Generate Appropriate data of circles
    a1, a2, r1 = ComputeCircle(x1, y1, x2, y2)
    b1, b2, r2 = ComputeCircle(x2, y2, x3, y3)
    c1, c2, r3 = ComputeCircle(x3, y3, x1, y1)
```

```
    fig = plt.figure()
    ax = fig.add_subplot(aspect = 'equal')


    circle1 = plt.Circle((a1, a2), r1, edgecolor = 'black', facecolor = "none")
    circle2 = plt.Circle((b1, b2), r2, edgecolor = 'black', facecolor = "none")
    circle3 = plt.Circle((c1, c2), r3, edgecolor = 'black', facecolor = "none")

    ax.add_patch(circle1)
    ax.add_patch(circle2)
    ax.add_patch(circle3)


    # Generate Points
    p1, q1 = GenerateCirclePoints(a1, a2, r1, x1, y1, x2, y2)
    p2, q2 = GenerateCirclePoints(b1, b2, r2, x2, y2, x3, y3)
    p3, q3 = GenerateCirclePoints(c1, c2, r3, x3, y3, x1, y1)

    x = p1 + p2 + p3
    y = q1 + q2 + q3

    plt.plot(x, y, color = "black")
    plt.fill(x, y, color = "pink")

    # One error about filling inside not outside when shape includes infinity
    # This is only thing that really needs to be finished up.

    circle = plt.Circle((0, 0), 1, linewidth = 1.5, edgecolor = 'red', facecolor = "none")
    ax.add_patch(circle)

    plt.plot(x1, y1, marker="o", markersize = 5, markerfacecolor = "red", markeredgecolor =
    plt.plot(x2, y2, marker="o", markersize = 5, markerfacecolor = "green", markeredgecolor
    plt.plot(x3, y3, marker="o", markersize = 5, markerfacecolor = "blue", markeredgecolor =


    ax.set_xlim((-1.5, 1.5))
    ax.set_ylim((-1.5, 1.5))

    plt.show()


def GenerateRandomTriangle():
    x = [0 for i in range(6)]
    for i in range(3):
        while True:
            a = random.uniform(-1,1)
```

```
        b = random.uniform(-1,1)
        if (a*a + b*b <= 1):
            x[2*i] = a
            x[2*i + 1] = b
            break

DrawHyperbolicTriangle(x[0], x[1], x[2], x[3], x[4], x[5])
```

3.3 SphericalRoots.py

```
import matplotlib.pyplot as plt
import numpy
import SphericalTriangles as st

def DrawRootsPolygon(r, n):

    # Maths

    a = [0 for i in range(n)]
    b = [0 for i in range(n)]
    rad = [0 for i in range(n)]

    fig = plt.figure()
    ax = fig.add_subplot(aspect = 'equal')

    x = [r*numpy.cos(2*i*numpy.pi/n) for i in range(n)]
    y = [r*numpy.sin(2*i*numpy.pi/n) for i in range(n)]

    p = []
    q = []

    for i in range(n):
        a[i], b[i], rad[i] = st.ComputeCircle(x[i], y[i], x[(i+1) % n], y[(i+1) % n])
        p1, q1 = st.GenerateCirclePoints(a[i], b[i], rad[i], x[i], y[i], x[(i+1) % n], y[(i+
        p = p + p1
        q = q + q1

    mag2 = a[0]*a[0] + b[0]*b[0]
    l = 1- (numpy.sqrt(mag2 + 1))/numpy.sqrt(mag2)


    #Graphics

    circle = plt.Circle((0, 0), 1, edgecolor = 'red', facecolor = "none")
    ax.add_patch(circle)

    plt.plot(p, q, color = "black")
    ax.fill(p, q, color = "pink")

    for i in range(n):
        plt.plot([0, a[i]*l], [0, b[i]*l], color = "black")
        plt.plot([0, x[i]], [0, y[i]], color = "black")
        plt.plot(x[i], y[i], marker="o", markersize = 5, markerfacecolor = "red", markeredge
        plt.plot(a[i]*l, b[i]*l, marker="o", markersize = 5, markerfacecolor = "blue", marke
```

```
    ax.set_xlim((-1.2, 1.2))
    ax.set_ylim((-1.2, 1.2))

    plt.show()

def DrawAngleNgon(n):
    if (n not in [3, 4, 5]): return
    r = numpy.tan(1/2 * numpy.arccos(1/(numpy.sqrt(3)*numpy.tan(numpy.pi/n))))
    print(r)
    DrawRootsPolygon(r, n)

DrawAngleNgon(3)
```

3.4 HyperbolicRoots.py

```
 import matplotlib.pyplot as plt
import numpy
import HyperbolicTriangles as ht

def DrawRootsPolygon(r, n):

    # Maths

    a = [0 for i in range(n)]
    b = [0 for i in range(n)]
    rad = [0 for i in range(n)]

    fig = plt.figure()
    ax = fig.add_subplot(aspect = 'equal')

    x = [r*numpy.cos(2*i*numpy.pi/n) for i in range(n)]
    y = [r*numpy.sin(2*i*numpy.pi/n) for i in range(n)]

    p = []
    q = []

    for i in range(n):
        a[i], b[i], rad[i] = ht.ComputeCircle(x[i], y[i], x[(i+1) % n], y[(i+1) % n])
        p1, q1 = ht.GenerateCirclePoints(a[i], b[i], rad[i], x[i], y[i], x[(i+1) % n], y[(i+
        p = p + p1
        q = q + q1

    mag2 = a[0]*a[0] + b[0]*b[0]
    l = 1- (numpy.sqrt(mag2 - 1))/numpy.sqrt(mag2)


    #Graphics

    circle = plt.Circle((0, 0), 1, edgecolor = 'red', facecolor = "none")
    ax.add_patch(circle)

    plt.plot(p, q, color = "black")
    ax.fill(p, q, color = "pink")

    for i in range(n):
        plt.plot([0, a[i]*l], [0, b[i]*l], color = "black")
        plt.plot([0, x[i]], [0, y[i]], color = "black")
        plt.plot(x[i], y[i], marker="o", markersize = 5, markerfacecolor = "red", markeredge
        plt.plot(a[i]*l, b[i]*l, marker="o", markersize = 5, markerfacecolor = "blue", marke
```

```
    ax.set_xlim((-1.2, 1.2))
    ax.set_ylim((-1.2, 1.2))

    plt.show()

def DrawAngleNgon(n):
    if (n <= 6): return
    r = numpy.tanh(1/2 * numpy.arccosh(1/(numpy.sqrt(3)*numpy.tan(numpy.pi/n))))
    print(r)
    DrawRootsPolygon(r, n)

DrawAngleNgon(17)
```

3.5 TriangleInversion.py

```
import matplotlib.pyplot as plt
import numpy
import HyperbolicTriangles as ht
import SphericalTriangles as st

def R1 (x, y):
    return x, -y

def R2 (x, y, p):
    X = x*numpy.cos(2*numpy.pi/p) + y*numpy.sin(2*numpy.pi/p)
    Y =x*numpy.sin(2*numpy.pi/p) - y*numpy.cos(2*numpy.pi/p)
    return X, Y

def R3 (a, b, r, x, y):
    x1, y1 = x - a, y - b
    norm = x1*x1 + y1*y1
    x2 = r*r*x1/norm
    y2 = r*r*y1/norm
    return x2 + a, y2 + b

# Mobius map version of R3
def R3Other(a, b, r, x, y):
    if (x == None): return a, b
    if (x == a and y == b): return None, None

    c1 = r*r - a*a - b*b
    z2 = x*x + y*y
    c2 = a*a - b*b
    c3 = 2*a*b
    sq = (x-a)*(x-a) + (y-b)*(y-b)

    re = (c1 * (x-a) + a*z2 - (c2*x +c3*y) )/sq
    im = (c1 * (y-b) + b*z2 - (c3*x - c2*y) )/sq
    return re, im

def S1(a, b, r, p, x, y):
    u, v = R3(a, b, r, x, y)
    w, z = R2(u, v, p)
    return w, z

def S2(a, b, r, p, x, y):
    u, v = R1(x, y)
    w, z = R3(a, b, r, u, v)
    return w, z
```

```
def S3(a, b, r, p, x, y):
    u, v = R2(x, y, p)
    w, z = R1(u, v)
    return w, z

def GenerateStartingTriangleData(p):
    if (p > 6): # Hyperbolic
        r = numpy.tanh(1/2 * numpy.arccosh(1/(numpy.sqrt(3)*numpy.tan(numpy.pi/p))))
        x, y = r*numpy.cos(2*numpy.pi/p), r*numpy.sin(2*numpy.pi/p)

        a, b, rad = ht.ComputeCircle(r, 0, x, y)

        mag2 = a*a + b*b
        l = 1 - (numpy.sqrt(mag2 - 1))/numpy.sqrt(mag2)
        x3, y3 = l*a, l*b
    else: # Spherical
        r = numpy.tan(1/2 * numpy.arccos(1/(numpy.sqrt(3)*numpy.tan(numpy.pi/p))))
        x, y = r*numpy.cos(2*numpy.pi/p), r*numpy.sin(2*numpy.pi/p)

        a, b, rad = st.ComputeCircle(r, 0, x, y)

        mag2 = a*a + b*b
        l = 1 - (numpy.sqrt(mag2 + 1))/numpy.sqrt(mag2)
        x3, y3 = l*a, l*b

    return 0, 0, r, 0, x3, y3, a, b, rad

def ComputeInitialTriangle(p):
    x1, y1, x2, y2, x3, y3, a, b, rad = GenerateStartingTriangleData(p)

    if (p > 6): # Hyp
        p1, q1 = ht.GenerateCirclePoints(None, None, None, x1, y1, x2, y2)
        p2, q2 = ht.GenerateCirclePoints(a, b, rad, x2, y2, x3, y3)
        p3, q3 = ht.GenerateCirclePoints(None, None, None, x3, y3, x1, y1)
    else: # Spherical
        p1, q1 = st.GenerateCirclePoints(None, None, None, x1, y1, x2, y2)
        p2, q2 = st.GenerateCirclePoints(a, b, rad, x2, y2, x3, y3)
        p3, q3 = st.GenerateCirclePoints(None, None, None, x3, y3, x1, y1)

    x = p1 + p2 + p3
    y = q1 + q2 + q3

    return x, y, a, b, rad

def DrawS1(x, y, p, a, b, rad):
```

```
    l = len(x)
    pX2 = [0 for i in range(l)]
    pY2 = [0 for i in range(l)]

    for i in range(l):
        pX2[i], pY2[i] = S1(a, b, rad, p, x[i], y[i])

    fig = plt.figure()
    ax = fig.add_subplot(aspect = 'equal')

    # Guidelines
    plt.plot([-100, 100], [0, 0], color = "green")
    plt.plot([-a*100, a*100], [-b*100, b*100], color = "orange")
    circle = plt.Circle((a, b), rad, edgecolor = 'red', facecolor = "none")
    ax.add_patch(circle)

    plt.plot(x, y, color = "black")
    ax.fill(x, y, color = "red")

    plt.plot(pX2, pY2, color = "black")
    ax.fill(pX2, pY2, color = "green")

    ax.set_xlim((-0.2, 1))
    ax.set_ylim((-0.2, 1))

    if (p == 3):
        ax.set_xlim((-0.5, 1))
        ax.set_ylim((-0.2, 1.5))

    plt.show()

def DrawS2(x, y, p, a, b, rad):
    l = len(x)
    pX2 = [0 for i in range(l)]
    pX3 = [0 for i in range(l)]
    pY2 = [0 for i in range(l)]
    pY3 = [0 for i in range(l)]

    for i in range(l):
        pX2[i], pY2[i] = S2(a, b, rad, p, x[i], y[i])
        pX3[i], pY3[i] = S2(a, b, rad, p, pX2[i], pY2[i])

    fig = plt.figure()
    ax = fig.add_subplot(aspect = 'equal')
```

```
    # Guidelines
    plt.plot([-100, 100], [0, 0], color = "green")
    plt.plot([-a*100, a*100], [-b*100, b*100], color = "orange")
    circle = plt.Circle((a, b), rad, edgecolor = 'red', facecolor = "none")
    ax.add_patch(circle)

    plt.plot(x, y, color = "black")
    ax.fill(x, y, color = "red")

    plt.plot(pX2, pY2, color = "black")
    ax.fill(pX2, pY2, color = "green")

    plt.plot(pX3, pY3, color = "black")
    ax.fill(pX3, pY3, color = "blue")

    ax.set_xlim((-0.2, 1))
    ax.set_ylim((-0.6, 0.6))

    if (p <= 4):
        ax.set_xlim((-0.2, 2.4))
        ax.set_ylim((-1.3, 1.3))

    plt.show()

def DrawS3(x, y, p, a, b, rad):

    l = len(x)
    pX = x
    pY = y

    fig = plt.figure()
    ax = fig.add_subplot(aspect = 'equal')

    # Guidelines
    plt.plot([-100, 100], [0, 0], color = "green")
    plt.plot([-a*100, a*100], [-b*100, b*100], color = "orange")
    circle = plt.Circle((a, b), rad, edgecolor = 'red', facecolor = "none")
    ax.add_patch(circle)

    plt.plot(x, y, color = "black")
    ax.fill(x, y)

    for j in range(p-1):
        for i in range(l):
            pX[i], pY[i] = S3(a, b, rad, p, pX[i], pY[i])
```

```
            plt.plot(pX, pY, color = "black")
            ax.fill(pX, pY)

        ax.set_xlim((-1, 1))
        ax.set_ylim((-1, 1))

        plt.show()

    def DrawElement(x, y, p, a, b, rad, path, drawPath):

        l = len(x)
        pX = x
        pY = y

        fig = plt.figure()
        ax = fig.add_subplot(aspect = 'equal')

        # Guidelines
        plt.plot([-100, 100], [0, 0], color = "green")
        plt.plot([-a*100, a*100], [-b*100, b*100], color = "orange")
        circle = plt.Circle((a, b), rad, edgecolor = 'red', facecolor = "none")
        ax.add_patch(circle)

        m = len(path)
        if (m == 0):
            plt.plot(x, y, color = "black")
            ax.fill(x, y, color = "red")

        for j in range(m):
            move = path[j]
            i = 0
            while i < l:
                match move:
                    case 1:
                        pX[i], pY[i] = S1(a, b, rad, p, pX[i], pY[i])
                    case 2:
                        pX[i], pY[i] = S2(a, b, rad, p, pX[i], pY[i])
                    case 3:
                        pX[i], pY[i] = S3(a, b, rad, p, pX[i], pY[i])

                i += 1

            if (drawPath or j == m-1):
                plt.plot(pX, pY, color = "black")
                ax.fill(pX, pY)
```

```
    ax.set_xlim((-4, 4))
    ax.set_ylim((-4, 4))

    if (p > 6):
        ax.set_xlim((-1, 1))
        ax.set_ylim((-1, 1))

    plt.show()

def Main(p):
    x, y, a, b, r = ComputeInitialTriangle(p)
    DrawS3(x, y, p, a, b, r)
    #DrawElement(x, y, p, a, b, r, [1, 2, 1, 2, 1])

Main(5)
```

3.6 SymmetryGroups.py

```
import matplotlib.pyplot as plt
import numpy
import TriangleInversion as ti

# Principle behind this program is to, given p.
# Generate PSL(2,p) and find all the elements
# Generate one composition of generators for each element.
# Store Matrix elements as 4-tuples

def MultiplyMatrices(x, y, p):
    z0 = (x[0]*y[0] + x[1]*y[2]) % p
    z1 = (x[0]*y[1] + x[1]*y[3]) % p
    z2 = (x[2]*y[0] + x[3]*y[2]) % p
    z3 = (x[2]*y[1] + x[3]*y[3]) % p

    z = (z0, z1, z2, z3)
    return z

def IsSame(x, y, p):
    flag1 = True
    for i in range(4):
        if (x[i] != y[i]):
            flag1 = False
            break

    if (flag1): return True

    flag2 = True
    for i in range(4):
        if (x[i] != (-y[i] % p) ):
            flag2 = False
            break

    if (flag2): return True

    return False

def IsElementInArray(el, arr, listSize, p):
    for i in range(listSize):
        if (IsSame(el, arr[i], p)):
            return True
    return False

def CalculateGroup(p):
```

```
    g1 = (0, 1, p-1, 0)
    g2 = (0, p-1, 1, p-1)
    g3 = (1, 1, 0, 1)

    # Start from I then iterate through each element
    # by left multiplying by generators
    elements = [(1, 0, 0, 1)]
    paths = [[]]

    numElts = 1
    i = 0
    while (i < numElts):
        el = elements[i]

        new1 = MultiplyMatrices(g1, el, p)
        flag1 = IsElementInArray(new1, elements, numElts, p)
        if (not flag1):
            elements.append(new1)
            numElts += 1
            arr = [paths[i][j] for j in range(len(paths[i]))]
            arr.append(1)
            paths.append(arr)

        new2 = MultiplyMatrices(g2, el, p)
        flag2 = IsElementInArray(new2, elements, numElts, p)
        if (not flag2):
            elements.append(new2)
            numElts += 1
            arr = [paths[i][j] for j in range(len(paths[i]))]
            arr.append(2)
            paths.append(arr)

        new3 = MultiplyMatrices(g3, el, p)
        flag3 = IsElementInArray(new3, elements, numElts, p)
        if (not flag3):
            elements.append(new3)
            numElts += 1
            arr = [paths[i][j] for j in range(len(paths[i]))]
            arr.append(3)
            paths.append(arr)

        i += 1

    return  numElts, elements, paths

def FindElementPath(el, order, group, paths, p):
```

```
    for i in range(order):
        flag = IsSame(el, group[i], p)
        if (flag): return paths[i]

    return []

def FindPathElement(path, p):
    g1 = (0, 1, p-1, 0)
    g2 = (0, p-1, 1, p-1)
    g3 = (1, 1, 0, 1)

    el = (1, 0, 0, 1)
    for i in range(len(path)):
        match path[i]:
            case 1: el = MultiplyMatrices(g1, el, p)
            case 2: el = MultiplyMatrices(g2, el, p)
            case 3: el = MultiplyMatrices(g3, el, p)

    return el

def DrawElementTriangle(el, order, group, paths, p):
    path = FindElementPath(el, order, group, paths, p)
    x, y, a, b, r = ti.ComputeInitialTriangle(p)
    ti.DrawElement(x, y, p, a, b, r, path, False)

def DrawWholeGroup(order, group, paths, p, drawOriginVertex):
    x, y, a, b, r = ti.ComputeInitialTriangle(p)
    l = len(x)

    fig = plt.figure()
    ax = fig.add_subplot(aspect = 'equal')

    # Guidelines
    plt.plot([-100, 100], [0, 0], color = "green")
    plt.plot([-a*100, a*100], [-b*100, b*100], color = "orange")
    circle = plt.Circle((a, b), r, edgecolor = 'red', facecolor = "none")
    ax.add_patch(circle)

    tD = 0

    for i in range(order):
        m = len(paths[i])
        if (m == 0):
            plt.plot(x, y, color = "black")
            ax.fill(x, y)
            if (drawOriginVertex):
```

```
                    plt.plot(0, 0, marker="o", markersize = 5, markerfacecolor = "white", ma
                tD += 1


            pX = [x[i] for i in range(l)]
            pY = [y[i] for i in range(l)]

            for j in range(m):
                move = paths[i][j]
                k = 0
                while k < l:
                    match move:
                        case 1: pX[k], pY[k] = ti.S1(a, b, r, p, pX[k], pY[k])
                        case 2: pX[k], pY[k] = ti.S2(a, b, r, p, pX[k], pY[k])
                        case 3: pX[k], pY[k] = ti.S3(a, b, r, p, pX[k], pY[k])

                    k += 1

                if (j == m-1):
                    plt.plot(pX, pY, color = "black")
                    ax.fill(pX, pY)
                    tD += 1

                    if (drawOriginVertex):
                        plt.plot(pX[0], pY[0], marker="o", markersize = 5, markerfacecolor = "wh

        ax.set_xlim((-4, 4))
        ax.set_ylim((-4, 4))

        if (p == 7):
            ax.set_xlim((-1, 1))
            ax.set_ylim((-1, 1))

        plt.show()

    def CrudeKernel(p):
        order, group, paths = GetGroupData357(p)

        arr = [3, 1, 2, 3, 3, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2]
        el = FindPathElement(arr, p)
        otherPath = FindElementPath(el, order, group, paths, p)
        rev = [2, 2, 1, 3, 3, 3, 3, 3, 3, 2, 2, 1, 3, 3, 3, 3, 3, 2, 2, 1]
        revEl = FindPathElement(rev, p)

        identityPath = arr + rev
        print(identityPath)
```

```
    el2 = FindPathElement(identityPath, p)
    print(el2)
    x, y, a, b, r = ti.ComputeInitialTriangle(p)
    ti.DrawElement(x, y, p, a, b, r, identityPath, False)

def Main(p):
    order, group, paths = GetGroupData357(p)
    #x, y, a, b, r = ti.ComputeInitialTriangle(p)
    #ti.DrawElement(x, y, p, a, b, r, paths[18], False)
    DrawWholeGroup(order, group, paths, p, False)

Main(7)
```