Project 2.3: Non-Euclidean Geometry

April 30, 2021

1 Question 1

If a spherical line passes through z_1 and z_2, then we have

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

Substituting for $\overline{a}$ we get

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

Which is a formula for a in terms of z_1 and z_2. I then wrote a program to fill in the spherical triangle of vertices z_1, z_2 and z_3. The program finds the edge between each pair of vertices then joins the edges together. The main challenge is ensuring, given a spherical line between two points, which arc of the great circle to take. We know that on the sphere spherical triangles have side lengths less than π, so we are looking for the arc that is shorter on the sphere.

However, stereographic projection is not isometric so there is no guarantee that the shorter arc on the sphere corresponds to the shorter arc on the extended complex plane. To get around this, we can use the antipodal map, since the shorter arc will be mapped by the antipodal map to a subset of the longer arc, and so comparing these lengths will tell us which arc is shorter on the sphere.

To find the antipodal map on the complex plane we can conjugate the map on the sphere with stereographic projection:

$$\phi : \mathbb{C}\cup\infty \longrightarrow S^2 \qquad \sigma : S^2 \longrightarrow S^2 \qquad \phi^{-1} : S^2 \longrightarrow \mathbb{C}\cup\infty$$

$$\phi(x,y) = \left(\frac{2x}{1+x^2+y^2}, \frac{2y}{1+x^2+y^2}, \frac{-1+x^2+y^2}{1+x^2+y^2}\right) \quad \sigma(x,y,z) = (-x,-y,-z) \quad \phi^{-1}(x,y,z) = \left(\frac{x}{1-z}, \frac{y}{1-z}\right)$$

Then $\phi\circ\sigma\circ\phi^{-1} : (x,y) \longmapsto \left(\frac{-2x(1+x^2+y^2)}{(1+x^2+y^2)(2x^2+2y^2)}, \frac{-2y(1+x^2+y^2)}{(1+x^2+y^2)(2x^2+2y^2)}\right) = -\frac{z}{|z|^2} = -\frac{1}{\overline{z}}$, so using this we can find the lengths of each arc under the antipodal map, and thus be sure that we have found the shortest arc on the sphere. Figures 1 through 4 show some examples of the results of the program with different vertices. One can note that figure 2 shows the program choosing what looks like a longer arc between 1-2i and i because the arc is shorter on the Riemann sphere. The only issue is that when the spherical triangle contains ∞ by the nature of the fill command the wrong part of the triangle is filled, however the boundary of the triangle is still correct.

For the last part we note that great circles intersect each other at antipodal points, and that the 'equatorial' great circle on the sphere, the intersection of the sphere with the xy-plane, maps identically to itself under stereographic projection. Therefore spherical lines will intersect the unit circle at two points, z_0 and $-z_0$.

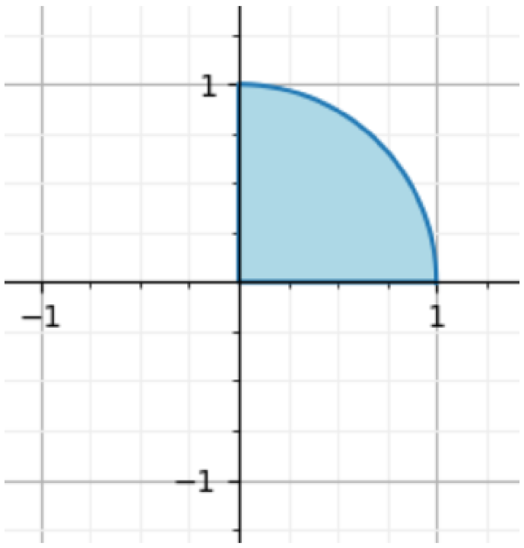

Figure 1: vertices 1, i, 0

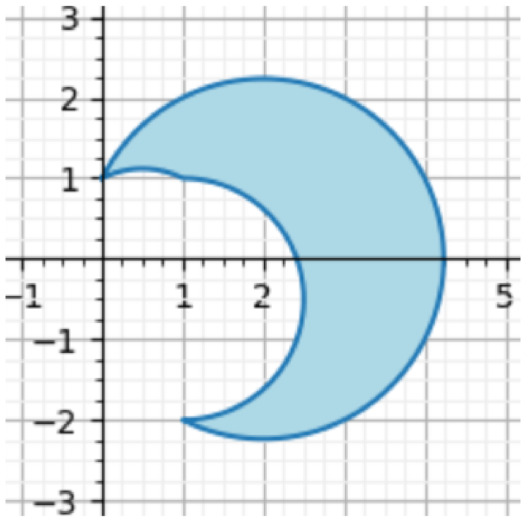

Figure 2: vertices 1, 1+i, 1-2i

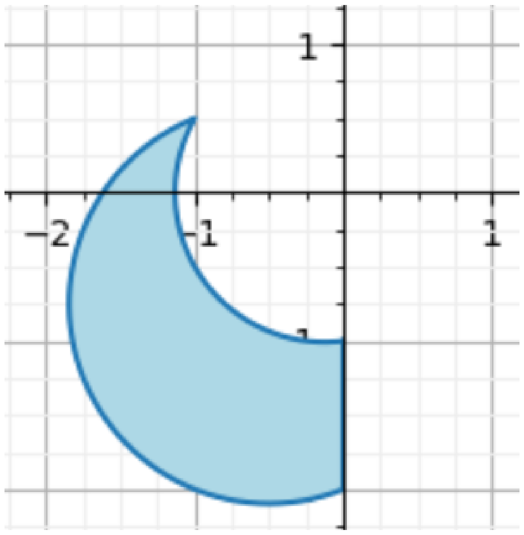

Figure 3: vertices -i, -2i, -1+0.5i

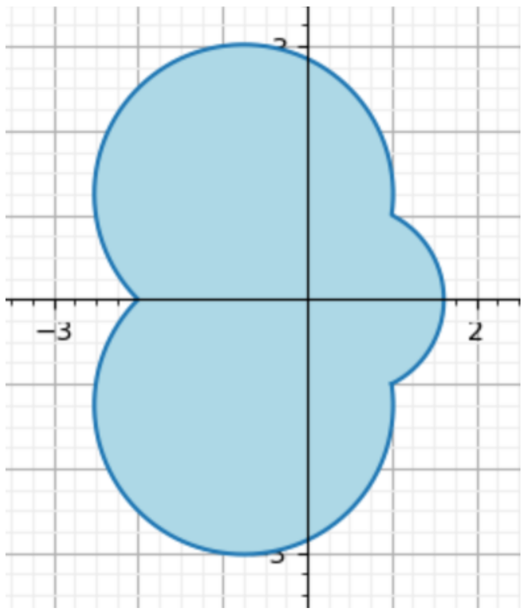

Figure 4: vertices 1+i, 1-i, -2

2 Question 2

We can mostly use the same methods in the hyperbolic plane as we did in the case of spherical lines, although there are some slight differences. One is that we get a different equation for the centre of circles, following the same method as before but changing +1 to -1 we get:

$$a = \frac{(|z_1|^2+1)z_2 - (|z_2|^2+1)z_1}{z_2\overline{z_1} - z_1\overline{z_2}}$$

We can also note that since $|a| > 1$ the shorter arc of the circle will always be the one contained in D, so we don't have to worry about antipodal maps as we did before. Then some results from the altered program are given in figures 5 and 6.

To find behaviour at the boundary we can view complex numbers as vectors in $\mathbb{R}^2$, and we see that if $z.z = 1$ then $(z-a).(z-a) = a.a - 1 \implies 1 - 2z.a + 1 = 0 \implies z.a = 1$, so then have that $z.(z-a) = 0$, which implies the angle between the radius of the disc and the radius of the circle through z is a right angle, which implies that hyperbolic lines intersect the boundary of D at right angles.

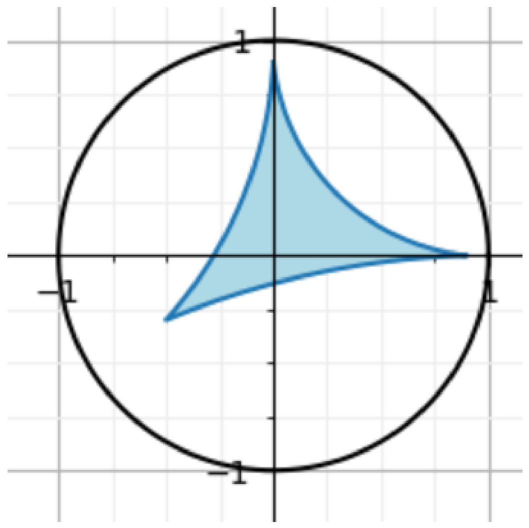

Figure 5: vertices 0.9i, 0.9, -0.5-0.3i

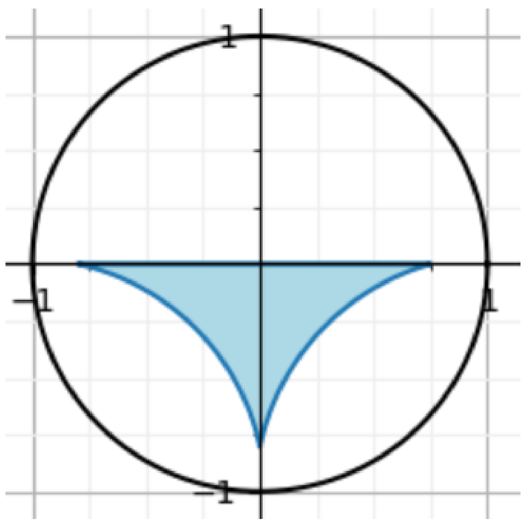

Figure 6: vertices -0.8, 0.75, -0.8j

3 Question 3

Firstly we note that given a regular n-gon, we can divide it into (n-2) triangles by connecting every vertex to a single one. Then let α_i, β_i and γ_i be the internal angles of the ith triangle. Then from Geometry we have that the area of the ith triangle is $\alpha_i + \beta_i + \gamma_i - \pi$. Also the sum of the internal angles of these triangles is exactly the sum of the internal angles of the n-gon, which is $\frac{2n\pi}{3}$. So we get that the area of the n-gon is

$$\sum_{i=1}^{n-2} \alpha_i + \beta_i + \gamma_i - \pi = \frac{2n\pi}{3} - (n-2)\pi = 2\pi - \frac{n\pi}{3}$$

Equally, if a regular n-gon with internal angle α and area $2\pi - \frac{n\pi}{3}$ exists, then the sum of internal angles is $n\alpha$ and

$$\begin{aligned} 2\pi - \frac{n\pi}{3} = \sum_{i=1}^{n-2} \alpha_i + \beta_i + \gamma_i - \pi &= n\alpha - (n-2)\pi \\ \implies \frac{n\pi}{3} &= n\pi - n\alpha \\ \implies \alpha &= \frac{2\pi}{3} \end{aligned}$$

So we get that a regular n-gon has internal angle $\frac{2\pi}{3}$ if and only if it has area $2\pi - \frac{n\pi}{3}$. So certainly there cannot exist a regular polygon with ≥ 6 vertices, as it would have area ≤ 0.

For $3 \leq n \leq 5$ we wish to show there exists r s.t. the n-gon with vertices at the roots of $z^n = r^n$ has area $2\pi - \frac{n\pi}{3}$. When r is 0 the area is 0, and when r=1 we are describing the unit circle, which corresponds to a hemisphere on the Riemann sphere and so has area 2π. We also have that area varies continuously with r, so by the intermediate value theorem there must exist r¿0 such that the corresponding n-gon has area $0 < 2\pi - \frac{n\pi}{3} < 2\pi$.

If we instead work in the hyperbolic plane, again can divide up our n-gon into (n-2) hyperbolic triangles with internal angles α_i, β_i and γ_i. From Geometry we have that the area of a hyperbolic triangle is $pi - \alpha_i - \beta_i - \gamma_i$. So the area of the n-gon is

$$\sum_{i=1}^{n-2} pi - \alpha_i - \beta_i - \gamma_i = (n-2)\pi - \frac{2n\pi}{3} = \frac{n\pi}{3} - 2\pi$$

And by a similar argument as before a regular hyperbolic n-gon has internal angle $\frac{2\pi}{3}$ if and only if it has area $\frac{n\pi}{3} - 2\pi$. Therefore to get a positive area we must have $n > 6$.So now need to show for all $n > 6$ such an n-gon exists.

Certainly if r=0 the area is 0. If r = 1 the internal angles must all be zero, since we showed in Q2 that hyperbolic lines all meet the unit disc at right angles. So the total internal angle of such an n-gon is 0. Then

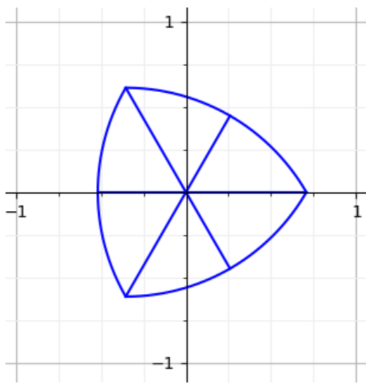

Figure 7: n=3

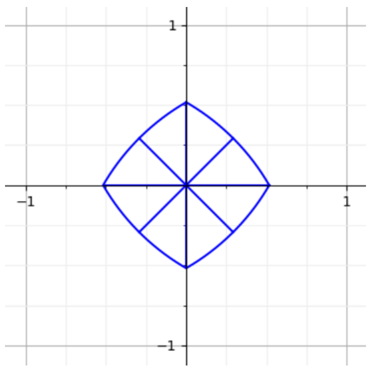

Figure 8: n=4

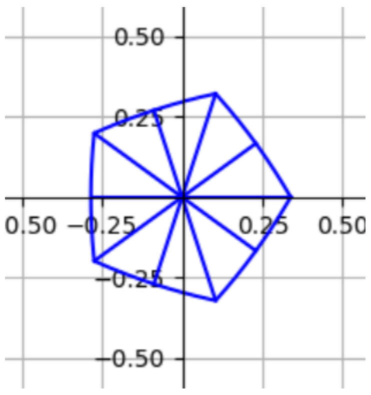

Figure 9: n=5

the area is

$$\sum_{i=1}^{n-2} pi - \alpha_i - \beta_i - \gamma_i = (n-2)\pi$$

Then we have $\forall n > 6, 0 < \frac{n\pi}{3} - 2\pi < (n-2)\pi$. Then since area depends continuously on r, by the intermediate value theorem, there exists an $r > 0$ such that the n-gon with vertices satisfying $z^n = r^n$ has area $\frac{n\pi}{3} - 2\pi$ and therefore has internal angle $\frac{2\pi}{3}$.

Given an n-gon with internal angle $\frac{2\pi}{3}$ drawing lines from the origin to each vertex subdivides it into n triangles with internal angles $\frac{2\pi}{n}$, $\frac{pi}{3}$ and $\frac{pi}{3}$. If we divide these triangles further in half with a line from the origin to halfway along the opposite side we then get 2n triangles with internal angles $\frac{\pi}{n}$, $\frac{\pi}{3}$ and $\frac{\pi}{2}$. Finding and displaying this subdivision once the correct value of r has been found. To do this we can use the fact that formula (2) states that

$$\cos(\frac{\pi}{2}) + \cos(\frac{\pi}{3})\cos(\frac{\pi}{n}) = \sin(\frac{\pi}{3})\sin(\frac{\pi}{n})\cos(a)$$

and that a is defined to be $2\tan^{-1}(r)$, then we get a value for r:

$$r = \tan(\frac{1}{2}\cos^{-1}(\cot(\frac{\pi}{3})\cot(\frac{\pi}{n})))$$

The program was then able to produce figures with this radius and connect the vertices and midpoints to the centre with lines. The results for n=3,4 and 5 are displayed in figures 7, 8 and 9.

For the hyperbolic case we can use the same process, but we must adjust the formula to find r to use equation (4). So instead have

$$\cos(\frac{\pi}{2}) + \cos(\frac{\pi}{3})\cos(\frac{\pi}{n}) = \sin(\frac{\pi}{3})\sin(\frac{\pi}{n})\cosh(a)$$

Which combined with $a = 2\tanh^{-}1(r)$ yields

$$r = \tanh(\frac{1}{2}\cosh^{-1}(\cot(\frac{\pi}{3})\cot(\frac{\pi}{n})))$$

Then the results of the program for some selected values of n>6 are given in figures 10,11 and 12.

4 Question 4

Firstly we note that S3 is the composition of reflections in two lines through the origin that are an angle of $\frac{\pi}{p}$ apart. We recognise this as a rotation about the origin of angle $\frac{2\pi}{p}$. Thinking now in terms of the sphere this implies that the composition of two spherical reflections in great circles is a rotation about the intersection of those circles. Therefore by symmetry of the sphere we can see that S1 and S2 are spherical rotations about one of the vertices with angle π and $\frac{2\pi}{3}$ respectively.

This then tells us that the order of S1 is p, the order of S2 is 3, and the order of S3 is 2. We'd now like to find the orders of the generators of PSL(2,5) given in (5).

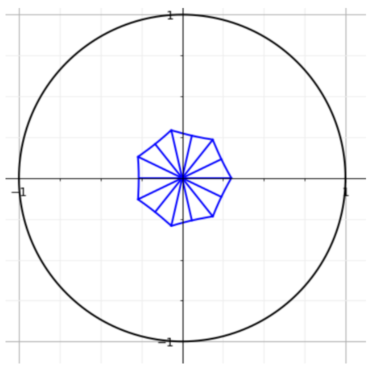

Figure 10: n=7

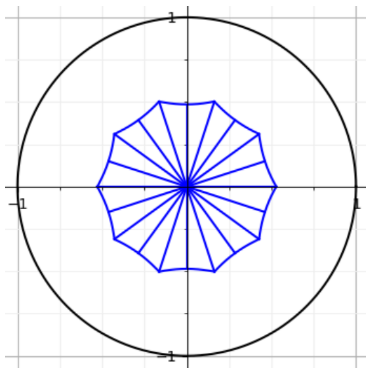

Figure 11: n=10

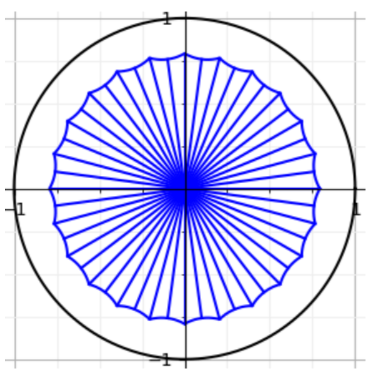

Figure 12: n=24

$$\sigma_1^2 = \pm \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}^2 = \pm \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\sigma_2^3 = \pm \begin{pmatrix} 0 & -1 \\ 1 & -1 \end{pmatrix}^3 = \pm \begin{pmatrix} 0 & -1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} -1 & 1 \\ -1 & 0 \end{pmatrix} = \pm \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

So σ_1 has order 2 and σ_2 has order 3, which are both the same as S1 and S2. We will show σ_3 has order p by showing $\sigma_3^n = \pm \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix}$, which would imply that σ_3^p is the identity since we are working mod p. Indeed assume $\sigma_3^{n-1} = \pm \begin{pmatrix} 1 & n-1 \\ 0 & 1 \end{pmatrix}$ then:

$$\sigma_3^n = \sigma_3 \sigma_3^{n-1} = \pm \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & n-1 \\ 0 & 1 \end{pmatrix} = \pm \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix}$$

So σ_3 has order p, and we are done.

5 Question 5

In order to find these admissible pairs we can first list the elements of PSL(2,3) as products of the generators σ_1, σ_2 and σ_3. Then, for each element in PSL(2,3) we can apply the sequence of transformations S_i with the same index as that element to the initial triangle.

I wrote a program to do this by repeatedly applying the generators to all the matrices known to be in PSL(2,p), then stopping when no more can be found and recording the sequence of generators that created that matrix. The results for p = 3, 5 and 7 are shown in figures 13, 14 and 15.

In the case of p=3 we can see there are 12 triangles which never overlap. It is also clear from the nature of S_i as in some sense rotations about points that it is not possible to generate any new triangles by applying more transformations.

In the case of p=5 we can count 60 triangles which also never overlap. Once again we can also see that no more triangles could be generated by applying more transformations.

In the case of p=7 it is possible (although it takes some time!) to count 168 triangles. I have also added numbering of each triangle that demonstrates this. In this case however it is clear that it would be possible to generate additional triangles through additional transformations, since there are points that have 'missing' triangles which a rotation would fill.

6 Question 6

In some sense we can view figure 13 as the stereographic projection of a sphere subdivided as a tetrahedron, likewise figure 14 can be viewed as the projection of the Riemann sphere subdivided as a dodecohedron.

Figures 16 and 17 demonstrate a way of viewing this decomposition.

We also know that PSL(2,3) is isomorphic to the orientation preserving rotations of a tetrahedron and PSL(2,7) is isomorphic to the orientation preserving rotations of a dodecahedron. This relation is suggested by

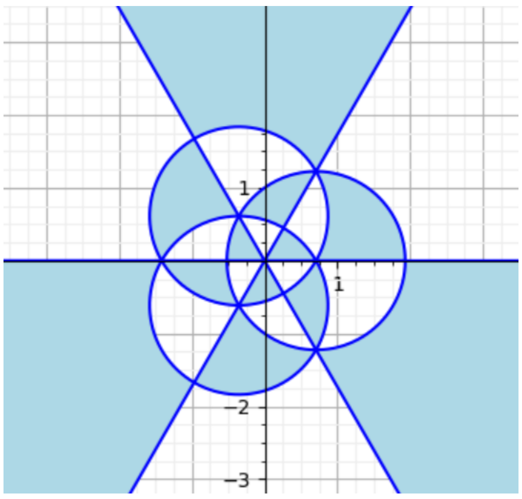
Figure 13: p=3

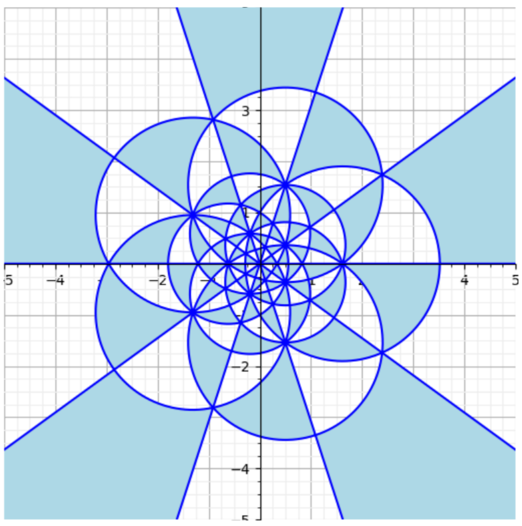
Figure 14: p=5

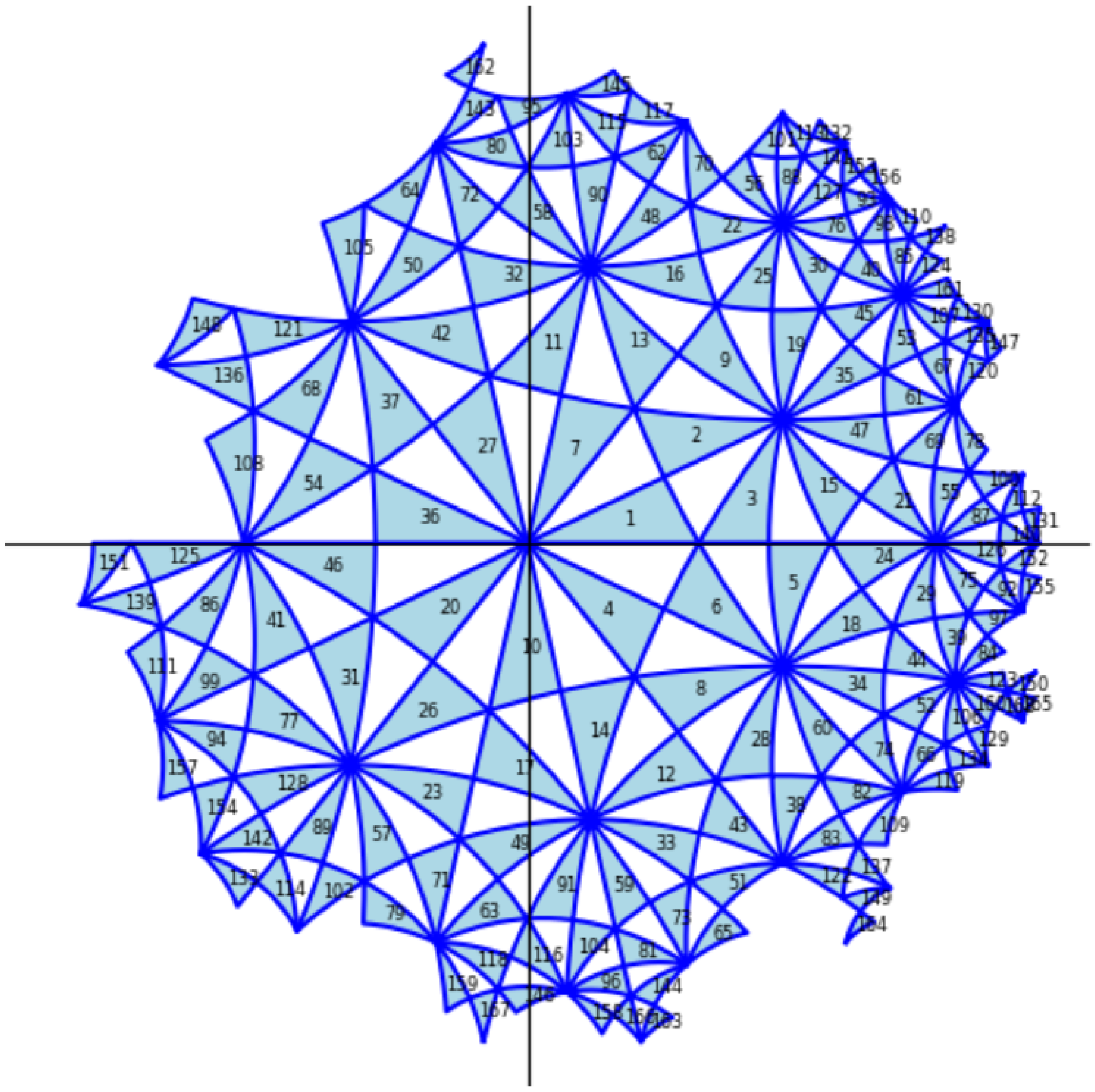
Figure 15: p=7

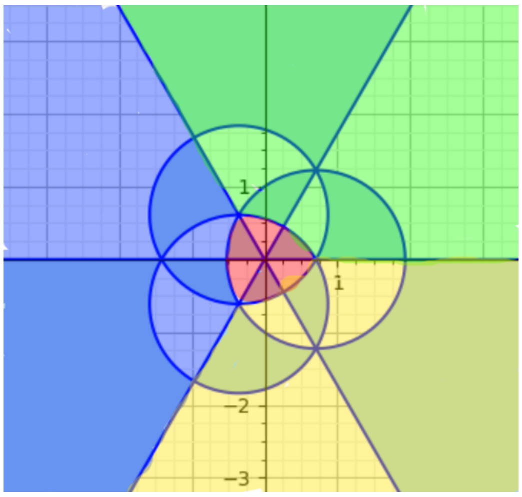

Figure 16: sides of a tetrahedron

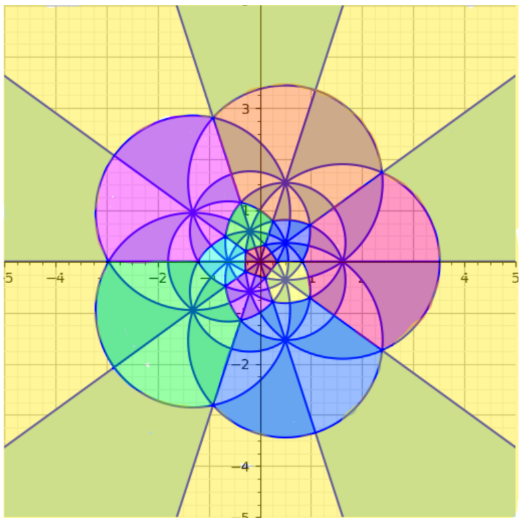

Figure 17: sides of a deodecahedron

the fact that (taking for example the case of a tetrahedron) starting with one orientation of a triangle PSL(2,3) exactly generated every other triangle in every orientation, which is exactly what you would expect from the group of rotations. The fact that they are orientation preserving is shown by the negative space of the graph, which would be filled by any reflections.

7 Code listing

7.1 Overview

I did this project using Python 3.8.6 64 bit, and the package numpy version 1.19.3, which adds many MATLAB functions and structures to Python. The first part of my code imported a variety of python packages that I would use in the code.

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

7.2 Processes

7.2.1 Spherical centre finding algorithm

This was my program to find the centre of a spherical circle given two points.

```
def findCentre(z1,z2):
    #This function assumes checks have been made to ensure points are not
    #colinear with zero
    numer = (abs(z1)**2-1)*z2-(abs(z2)**2-1)*z1

    denom = np.conj(z1)*z2-np.conj(z2)*z1

    return numer/denom
```

7.2.2 Spherically connect two points

This function takes as input two points on the complex plane and outputs a complex vector of points connecting the two along the shortest (in the sense of spherical distance) path between them.

```
def connect(z1,z2,points):
    #produces a vector of complex numbers along the great circle
    #starts from z1 and finishes at z2
    connection = np.zeros((points+1),dtype = complex)
    if z1==0 or abs(np.imag(z2/z1))<10**(-5):
        for i in range(points+1):
            connection[i]=z1+(z2-z1)*i/points
        return connection
    else:
        a = findCentre(z1,z2)
        arg = np.angle((z2-a)/(z1-a))
        anti1 = -1/np.conj(z1)
        anti2 = -1/np.conj(z2)
        antia= findCentre(anti1,anti2)
        antiarg = np.angle((anti2-antia)/(anti1-antia))
        if abs(arg)<=abs(antiarg):
            for i in range(points+1):
                connection[i]=a+(z1-a)*np.exp(1j*i*arg/points)
            return connection
        else:

            for i in range(points+1):
                connection[i]=antia+(anti1-antia)*np.exp(1j*i*(antiarg)/points)
            connection = np.conj(connection)
            connection = -np.reciprocal(connection)
            return connection
```

7.2.3 Create a spherical polygon

This program takes a set of points and outputs a vector of spherical lines between them in order.

```
def polygon(vertices,points):
    number = len(vertices)
    output = np.zeros((points*number),dtype = 'complex')
    for i in range(number):
        output[i*1000:(i+1)*1000]=connect(vertices[i],vertices[(i+1)%numb
        er],points)[:points]
    return output
```

7.2.4 Find centre of hyperbolic circle

This function takes two complex numbers as input and outputs the centre of the hyperbolic circle that passes through those two points.

```
def findHypCentre(z1,z2):
    #This function assumes checks have been made to ensure points are not
    #colinear with zero
    numer = (abs(z1)**2+1)*z2-(abs(z2)**2+1)*z1

    denom = np.conj(z1)*z2-np.conj(z2)*z1
```

7.2.5 connect two points with a hyperbolic line

This program takes two points as input and outputs a vector connecting them with a hyperbolic line.

```
def hypConnect(z1,z2,points):
    #produces a vector of complex numbers along the great circle
    #starts from z1 and finishes at z2
    connection = np.zeros((points+1),dtype = complex)
    if z1==0 or abs(np.imag(z2/z1))<10**(-5):
        for i in range(points+1):
            connection[i]=z1+(z2-z1)*i/points
        return connection
    else:
        a = findHypCentre(z1,z2)
        arg = np.angle((z2-a)/(z1-a))
        for i in range(points+1):
            connection[i]=a+(z1-a)*np.exp(1j*i*arg/points)
        return connection
```

7.2.6 Create a hyperbolic polygon

This function takes a list of complex points and connects them with hyperbolic lines, outputting a vector of points.

```
def hypPolygon(vertices ,points):
    number = len(vertices)
    output = np.zeros((points*number),dtype = 'complex')
    for i in range(number):
        output[i*1000:(i+1)*1000]=hypConnect(vertices[i],vertices[(i+1)
        %number],points)[:points]
    return output
```

7.2.7 Create spherical polygon divided into triangles

This is the first program for question 3, it takes as input either 3,4 or 5 and then draws both a regular n-gon and lines dividing into triangles with internal angle $\frac{\pi}{2}$,$\frac{\pi}{3}$ and $\frac{\pi}{n}$.

```
def divider(n,points):
    cosa= np.cos(np.pi/3)*np.cos(np.pi/n)/(np.sin(np.pi/3)*np.sin(np.pi/n))
    r = np.tan((1/2)*np.arccos(cosa))
    vertices = np.zeros((n),dtype = 'complex')
    todraw = []
    for i in range(n):
        vertices[i]=r*np.exp(2j*np.pi*i/n)
    midvertices = np.zeros((n),dtype='complex')
    for i in range(n):
        nexti = (i+1)%n
        midvertices[i]=connect(vertices[i],vertices[nexti],2*points)[points]
    todraw.append(polygon(vertices ,points))
    for i in range(n):
        todraw.append(connect(0,vertices[i],points))
        todraw.append(connect(0,midvertices[i],points))
    return todraw
```

7.2.8 Create hyperbolic polygon divided into triangles

This is exactly equivalent to the previous program except it takes $n \geq 7$, and so draws hyperbolic lines rather then spherical ones.

```
def hypDivider(n,points):
    cosha= np.cos(np.pi/3)*np.cos(np.pi/n)/(np.sin(np.pi/3)*np.sin(np.pi/n))
    r = np.tanh((1/2)*np.arccosh(cosha))
    vertices = np.zeros((n),dtype = 'complex')
    todraw = []
    for i in range(n):
        vertices[i]=r*np.exp(2j*np.pi*i/n)
    midvertices = np.zeros((n),dtype='complex')
    for i in range(n):
        nexti = (i+1)%n
        midvertices[i]=hypConnect(vertices[i],vertices[nexti],2*points)[points]
    todraw.append(hypPolygon(vertices ,points))
    for i in range(n):
        todraw.append(hypConnect(0,vertices[i],points))
        todraw.append(hypConnect(0,midvertices[i],points))
    return todraw
```

7.2.9 Multiply two matrices mod p

This short functions multiplies two input matrices mod p.

```
def pmultiply(M1,M2,p):
    M = np.matmul(M1,M2)
    Mprime = np.mod(M,p)
    return Mprime
```

7.2.10 Equivalence in PSL(2,p)

This function takes two matrices and checks whether they are equivalent in PSL(2,p). It is used in later functions.

```
def PSLequal(M1,M2,p):
    M1= np.mod(M1,p)
    M2 = np.mod(M2,p)
    if np.all(M1==M2):
        return True
    elif np.all(M1==np.mod(-M2,p)):
        return True
    else:
        return False
```

7.2.11 Invert points in a line

This function takes a set of points and an angle θ, and then inverts the points in the line with argumment θ.

```
ef lineInvert(z,theta):
    return(np.exp(2*1j*theta)*np.conj(z))
```

7.2.12 Invert in circle

This program takes a set of points, a centre and a radius and then inverts the points in the circle with that centre and radius.

```
def circleInvert(z,a,r):
    return(a+r**2*np.reciprocal(np.conj(z)-np.conj(a)))
```

7.2.13 Apply transformations S1, S2 and S3

Given information about the shape of the Δ_0 triangle, and the index of the transformation, apply the appropriate transformation to the inputted set of points.

```
def rotator(z,z1,a,index,p):
    if index == 0:
        return lineInvert(circleInvert(z,a,abs(z1-a)),np.pi/p)
    elif index == 1:
        return circleInvert(lineInvert(z,0),a,abs(z1-a))
    elif index == 2:
        return lineInvert(lineInvert(z,np.pi/p),0)
    else:
        print('index not in range')
        return False
```

7.2.14 Find admissible triangles

This program first exhaustively finds every element in PSL(2,p) and a sequence of generators that generate it by repeatedly applying generators to the elements we know are in PSL(2,p) until it can't add more.

Then, for each element in PSL(2,p) it applies the equivalent reflection to the starting triangle Δ_0. It then returns the resulting triangles in a list.

```
def triangulator(p,points):
    PSL = [[np.array([[1,0],[0,1]]),[]]]
    operations = [np.array([[0,1],[-1,0]]),np.array([[0,-1],[1,-1]])
    ,np.array([[1,1],[0,1]])]
    for i in range(1000):
        print(len(PSL))
        newPSL=list(PSL)
        finished = True
        for M in PSL:
            for R in range(3):
                inPSL=False
                for K in PSL:
                    if PSLequal(pmultiply(operations[R],M[0],p),K[0],p):
                        inPSL=True
                    else:
                        pass
                if not inPSL and finished:
                    new = [pmultiply(operations[R],M[0],p),list(M[1])]
                    new[1].append(R)
                    newPSL.append(new)
                    finished = False
                    break
            if not finished:
                break

        PSL=list(newPSL)
        if finished:
            break
    triangles = []
    if p<6:
        cosa1= np.cos(np.pi/3)*np.cos(np.pi/p)/(np.sin(np.pi/3)*np.sin(np.pi/p))
        r1 = np.tan((1/2)*np.arccos(cosa1))
        cosa2 = (np.cos(np.pi/3)+np.cos(np.pi/2)*np.cos(np.pi/p))/
        (np.sin(np.pi/2)*np.sin(np.pi/p))
        r2 = np.tan((1/2)*np.arccos(cosa2))
        z1=r1
        z2=r2*np.exp(1j*np.pi/p)
        triangle0=polygon([0,z1,z2],points)
        a = findCentre(z1,z2)
    else:
        cosha1= np.cos(np.pi/3)*np.cos(np.pi/p)/(np.sin(np.pi/3)
        *np.sin(np.pi/p))
        r1 = np.tanh((1/2)*np.arccosh(cosha1))
        cosha2 = (np.cos(np.pi/3)+np.cos(np.pi/2)*
        np.cos(np.pi/p))/(np.sin(np.pi/2)*np.sin(np.pi/p))
        r2 = np.tanh((1/2)*np.arccosh(cosha2))
        z1=r1
        z2=r2*np.exp(1j*np.pi/p)
        triangle0=hypPolygon([0,z1,z2],points)
        a = findHypCentre(z1,z2)
    for i in PSL:
        newtriangle = np.array(triangle0)

        for k in i[1]:

            newtriangle = rotator(newtriangle,z1,a,k,p)
        triangles.append(newtriangle)

    return [triangles,PSL]
```

7.3 Plotting

The previous programs all returned vectors of points, its then necessary to plot the result on an appropriate graph. I used Matplotlib to do this.

Here is the code, although it was altered slightly depending on each situation.

```
fig = plt.figure(figsize = (10,10))
axis = fig.add_subplot(1,1,1)
axis.spines['left'].set_position('zero')
axis.spines['bottom'].set_position('zero')
axis.spines['top'].set_color('none')
axis.spines['right'].set_color('none')
#plt.plot(np.real(vector),np.imag(vector))
triang = triangulator(5,1000)
print(triang[1])
triangles=triang[0]
for i in range(len(triangles)):
    centrex=np.mean(np.real(triangles[i]))
    centrey=np.mean(np.imag(triangles[i]))
    plt.text(centrex-0.01,centrey-0.01,str(i+1),size = 'xx-small')

    if i<167:
        plt.plot(np.real(triangles[i]),np.imag(triangles[i]),color='b')
        plt.fill(np.real(triangles[i]),np.imag(triangles[i]),color='#add8e6')
    else:
        plt.plot(np.real(triangles[i]),np.imag(triangles[i]),color='r')
        plt.fill(np.real(triangles[i]),np.imag(triangles[i]),color='r')
#plt.plot(np.real(newvec),np.imag(newvec))
#for i in todraw:
    #plt.plot(np.real(i),np.imag(i),color = 'b')
autox = plt.xlim()
autoy = plt.ylim()


most = min(max(abs(autox[0]),abs(autox[1]),abs(autoy[0]),abs(autoy[1])),5)

plt.xlim(-most,most)
plt.ylim(-most,most)
majorticks = np.arange(-np.floor(most),np.floor(most)+1,1)
majorticks=majorticks[majorticks!=0]
minorticks = np.arange(-np.floor(most),np.floor(most)+0.25,0.25)
minorticks=minorticks[minorticks!=0]
plt.gca().set_aspect('equal', adjustable='box')

axis.set_yticks(minorticks,minor=True)
axis.set_xticks(minorticks,minor=True)
axis.set_yticks(majorticks)
axis.set_axisbelow(True)
axis.set_xticks(majorticks)
axis.grid(which='minor',color = '#EEEEEE')
axis.grid()

#adds circle for hyperbolic disc
ticks = np.linspace(0.01,2*np.pi,1000)
#circle = np.exp(1j*ticks)
#plt.plot(np.real(circle),np.imag(circle),color='black')
#plt.fill(np.real(vector),np.imag(vector),color='#add8e6')
plt.show()
```