UNIVERSITÉ DU LUXEMBOURG

Bachelor en Mathématiques

Mathématiques expérimentales

Semestre d'hiver 2021

Les fractions continues et les surfaces triangulaires de Hecke

Auteurs :
Béatrice Bach
Léa Micard
Lara Suys

Superviseurs :
Prof. Dr. Gabor Wiese
Prof. Dr. Lassina Dambele

Table des matières

Introduction

Notre objectif pour ce projet est de montrer que tout point dans le demi-plan de Poincaré, qui est le sous-ensemble des nombres complexes dont la partie imaginaire est strictement positive, peut être ramené dans le domaine fondamental par une certaine combinaison. Nous voulons ensuite trouver l'algorithme qui nous donne cette combinaison.

Dans une première section, nous allons définir les actions sur le demi-plan de Poincaré et nous allons introduire les matrices S et T.
Dans la deuxième section, nous parlerons brièvement des fractions continues.
Ensuite dans la troisième section, nous discuterons des surfaces triangulaires de Hecke.
Enfin, nous terminerons ce projet par la quatrième section où tous nos programmes seront présentés et expliqués. Nous animerons les programmes à l'aide de `Pygame` et `Matplotlib`.

Dans la section annexe, tous les programmes peuvent être retrouvés sous leurs formes intégrales.

Toutes les images de notre projet ont été réalisées par nous-même. Nous avons utilisé soit PSTricks, soit nos programmes.

1 Action sur le demi-plan de Poincaré

Définition 1. Le *demi-plan de Poincaré*, noté $\mathfrak{H}$, est l'ensemble des nombres complexes dont la partie imaginaire est strictement positive.

$$\begin{aligned}\mathfrak{H} &:= \{\tau \in \mathbb{C} : Im(z) > 0\} \subset \mathbb{C} \\ &:= \{\tau = x + iy \in \mathbb{C} : y > 0\}\end{aligned}$$

Définition 2. Le groupe $GL_n(K)$ est le groupe des matrices carrées inversibles de taille n à coefficients dans K où K est un corps commutatif. Il est appelé *groupe général linéaire* de degré n d'un corps commutatif K.

$$GL_n(K) = \{\gamma \in \mathcal{M}_n(K) : det(\gamma) \neq 0\} \quad \text{avec } K \text{ un corps commutatif.}$$

Définition 3. Le groupe $SL_n(K)$ est le groupe des matrices carrées inversibles de taille n et de déterminant égal à 1 à coefficients dans K où K est un corps commutatif. Il est appelé *groupe spécial linéaire* de degré n d'un corps commutatif K.

$$SL_n(K) = \{\gamma \in \mathcal{M}_n(K) : det(\gamma) = 1\} \quad \text{avec } K \text{ un corps commutatif.}$$

Remarque. Nous allons prioritairement utiliser les ensembles suivants :

$$GL_2(\mathbb{R}) := \{\gamma \in \mathcal{M}_2(\mathbb{R}) : det(\gamma) \neq 0\}$$

$$GL_2^+(\mathbb{R}) := \{\gamma \in \mathcal{M}_2(\mathbb{R}) : det(\gamma) > 0\}$$

$$SL_2(\mathbb{R}) := \{\gamma \in \mathcal{M}_2(\mathbb{R}) : det(\gamma) = 1\}$$

$$SL_2(\mathbb{Z}) := \{\gamma \in \mathcal{M}_2(\mathbb{Z}) : det(\gamma) = 1\}$$

Nous faisons maintenant agir le groupe $SL_2(\mathbb{Z})$ sur le demi-plan de Poincaré $\mathfrak{H}$.

$$gz = \frac{az+b}{cz+d} \quad \forall g = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in SL_2(\mathbb{Z}) \text{ et } \forall z = x + iy \in \mathfrak{H}.$$

Comme z appartient à $\mathfrak{H}$, nous savons que $Im(z) > 0$.
De plus $g \in SL_2(\mathbb{Z})$, d'où nous savons que :

- Si $c = 0$, alors $d \neq 0$ et donc $cz + d \neq 0$.
- Si $c \neq 0$, alors $Im(cz + d) = cIm(z) \neq 0$ et donc $cz + d \neq 0$.

Nous pouvons en conclure que $cz + d \neq 0 \ \ \forall g = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in SL_2(\mathbb{Z})$ et $\forall z \in \mathfrak{H}$, ce qui signifie que cette quantité est bien définie.

Définition 4. Soit E un ensemble et $(G, *, e)$ un groupe de loi $*$ et d'élément neutre e. Une *action de G sur E* est une application

$$\begin{array}{ccc} G \times E & \to & E \\ (g, x) & \mapsto & g \cdot x \end{array}$$

satisfaisant les propriétés suivantes :

1. $\forall\, x \in E \quad e \cdot x = x$
2. $\forall (g, g') \in G^2 \quad \forall\, x \in E \quad g' \cdot (g \cdot x) = (g'\, g) \cdot x$, avec $g \cdot x \in E$ et $g'\, g \in G$.

Proposition 5. Soit $g = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in SL_2(\mathbb{Z})$. Alors l'application

$$\begin{array}{ccc} SL_2(\mathbb{Z}) \times \mathfrak{H} & \to & \mathfrak{H} \\ gz & \mapsto & \frac{az+b}{cz+d} \end{array}$$

est une action de $SL_2(\mathbb{Z})$ sur le demi-plan de Poincaré $\mathfrak{H}$.

Remarque. Les transformations de la forme

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \tau := \frac{a \times \tau + b}{c \times \tau + d} \quad \text{avec } \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in SL_2(\mathbb{R}) \text{ et } \tau \in \mathfrak{H}$$

sont appelées *transformations de Möbius.*

Démonstration de la Proposition 5.

Soient $g = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in SL_2(\mathbb{Z})$ et $z \in \mathfrak{H}$.

$$\begin{aligned} Im(gz) &= Im\left(\frac{az+b}{cz+d}\right) \\ &= Im\left(\frac{(az+b)\overline{(cz+d)}}{(cz+d)\overline{(cz+d)}}\right) \\ &= Im\left(\frac{(az+b)(c\overline{z}+d)}{|cz+d|^2}\right) \\ &= Im\left(\frac{acz\overline{z}+adz+bc\overline{z}+bd}{|cz+d|^2}\right) \\ &= Im\left(\frac{adz+bc\overline{z}}{|cz+d|^2}\right) \\ &= \frac{ad\, Im(z) + bc\, Im(\overline{z})}{|cz+d|^2} \\ &= \frac{ad\, Im(z) - bc\, Im(z)}{|cz+d|^2} \end{aligned}$$

$$= \frac{(ad-bc)Im(z)}{|cz+d|^2}$$

$$= \frac{\det(z)\, Im(z)}{|cz+d|^2} \quad \text{avec } \det(z) = ad-bc = 1$$

$$= \frac{Im(z)}{|cz+d|^2} > 0$$

$Im(gz) > 0$, donc $g\,z \in \mathfrak{H}$.

$I_2 z = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} z = \frac{1 \times z + 0}{0 \times z + 1} = z$, donc $\forall z \in \mathfrak{H},\ I_2 z = z$.

Soient $g = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, g' = \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} \in SL_2(\mathbb{Z})$ et $z \in \mathfrak{H}$.

$$(gg')z = \left(\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} \right) z$$

$$= \begin{pmatrix} aa'+bc' & ab'+bd' \\ ca'+dc' & cb'+dd' \end{pmatrix} z$$

$$= \frac{(aa'+bc')z + ab' + bd'}{(ca'+dc')z + cb' + dd'}$$

$$= \frac{aa'z + ab' + bc'z + bd'}{ca'z + cb' + dc'z + dd'}$$

$$= \frac{\frac{aa'z+ab'+bc'z+bd'}{c'z+d'}}{\frac{ca'z+cb'+dc'z+dd'}{c'z+d'}}$$

$$= \frac{\frac{aa'z+ab'}{c'z+d'} + \frac{bc'z+bd'}{c'z+d'}}{\frac{ca'z+cb'}{c'z+d'} + \frac{dc'z+dd'}{c'z+d'}}$$

$$= \frac{\frac{aa'z+ab'}{c'z+d'} + b}{\frac{ca'z+cb'}{c'z+d'} + d}$$

$$= \frac{a\frac{a'z+b'}{c'z+d'} + b}{c\frac{a'z+b'}{c'z+d'} + d}$$

$$= \begin{pmatrix} a & b \\ c & d \end{pmatrix} \left(\frac{a'z + b'}{c'z + d'} \right)$$

$$= g \left(\frac{a'z + b'}{c'z + d'} \right)$$

$$= g(g'z)$$

D'où $(gg')z = g(g'z)\ \forall z \in \mathfrak{H}, \forall g, g' \in SL_2(\mathbb{Z})$.

Nous pouvons en conclure qu'il s'agit bien d'une action. □

Deux éléments importants du groupe $SL_2(\mathbb{Z})$ sont $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ et $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$.
Faisons désormais agir S et T sur $z \in \mathfrak{H}$:

$$Sz = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} z = \frac{0 \times z - 1}{1 \times z + 0} = -\frac{1}{z}$$

$$Tz = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} z = \frac{1 \times z + 1}{0 \times z + 1} = z + 1$$

Nous pouvons observer que S provoque une inversion et que T provoque une translation d'une unité vers la droite.
Nous avons de plus que

$$S^2 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = -I_2$$

$$ST = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix}$$

$$(ST)^3 = \begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix}^3 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I_2$$

$$T^{-1} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$$

Définition 6. Soit G un groupe et soit E un ensemble sur lequel G agit. L'image d'un point $x \in E$ par l'action de l'élément $g \in G$ est notée $g(x)$. Un sous-ensemble $F \subset E$ est appelé domaine fondamental pour l'action du groupe G sur l'ensemble E si :

1. $\bigcup_{g \in G} g(F) = E$ et
2. $\forall g \neq g' \in G,\ g(F) \cap g'(F) = \emptyset$

Proposition 7. Le domaine fondamental pour l'action du groupe modulaire sur le demi-plan de Poincaré est défini par

$$D = \left\{ z \in \mathfrak{H} : \left(-\frac{1}{2} < Re(z) < 0 \text{ et } |z| > 1 \right) \text{ ou } \left(0 \leq Re(z) \leq \frac{1}{2} \text{ et } |z| \geq 1 \right) \right\}.$$

Démonstration.

Notons $D = \{z \in \mathfrak{H},\ |Re(z)| \leq \frac{1}{2} \text{ et } |z| > 1\}$.
Nous allons montrer que D est un domaine fondamental de l'action du groupe modulaire sur H, c'est-à-dire que :

1. Toute orbite rencontre D en un ou deux points,
2. Si deux points de D sont dans une même orbite, alors ils sont sur la frontière ∂D de D.

Soient $z, z' \in \mathfrak{H}$ tels que $Im(z') \geq Im(z)$ et $z' = gz$ où $g = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in SL_2(\mathbb{Z})$.
Nous savons que la forme quadratique, $\mathbb{R}^2 \to \mathbb{R}, (c,d) \mapsto |cz+d|^2$ admet un minimum étant donné qu'elle est définie positive sur $\mathbb{Z}^2 \backslash \{(0,0)\}$.
De plus, nous savons par la démonstration de la proposition 5 que $Im(gz) = \dfrac{Im(z)}{|cz+d|^2}$.
Donc, nous pouvons considérer l'ensemble $E \subset SL_2(\mathbb{Z})z$ des éléments z' tels que $Im(z')$ est maximal.
De plus, E est invariant par l'application $z' \mapsto z' + 1$.
Nous en déduisons qu'il existe $z' \in E$ tel que $|Re(z')| \leq \dfrac{1}{2}$.
Ensuite, nous observons que $-\dfrac{1}{z'} \in SL_2(\mathbb{Z})z$ mais nous savons que $Im\left(-\dfrac{1}{z'}\right) = \dfrac{Im(z')}{|z'|^2}$.
Nous concluons que $|z'| \geq 1$.
En résumé, nous obtenons que $z' \in SL_2(\mathbb{Z})z \cap D$, donc $G'z \cap D \neq \emptyset$, avec G' le sous-groupe de $SL_2(\mathbb{Z})$ engendré par S et T.
Ce qui établit le point 1.

Nous déduisons par le supposition faites au début de la démonstration que $|cz+d| \leq 1$ et en particulier $|c\, Im(z)| \leq 1$ donc $|c| \leq 1$.
Regardons deux cas différents :

1. Si $c = 0$ alors $d = a = \pm 1$ et donc $g = \pm \begin{pmatrix} 1 & b \\ 0 & 1 \end{pmatrix}$.

   Il existe donc $n \in \mathbb{Z}$ tel que $g = \pm T^n$.

2. Si $c = 1$ et si nous prenons l'opposé de g alors $|z+d| \leq 1$.
   Nous en déduisons que $z \in C$, avec C le cercle centré en $d \in \mathbb{Z} \subset \mathbb{R}$ de rayon au plus 1.

   La figure suivant la démonstration nous montre que $d = -1, 0, 1$ et que $|z+d| = |z| = 1$.

   Donc, soient C_d le cercle centré en d tel que $|z+d| = 1$ et C_0 le cercle centré en 0 tel que $|z| = 1$, on a : $z \in C_d \cup C_0$.

Il y a donc deux nouveaux cas :

i. Si $d = 0$, les cercles C_d et C_0 sont concentriques. Alors, $b = -1$ et $g = \begin{pmatrix} a & -1 \\ 1 & 0 \end{pmatrix}$.

Nous avons $z' = gz = a - \dfrac{1}{z}$ et $Re(z') - a = Re(-\frac{1}{z})$.

Comme $|Re(z')|$ et $|Re(-\dfrac{1}{z}) \leq \dfrac{1}{2}$, nous avons : $|a| \leq 1$ donc $a \in \{-1, 0, 1\}$.

a : Si $a = 0$ alors $z' = -\dfrac{1}{z}, g = S$ et $z = i$.

b : Si $a = -1$ alors $g = \begin{pmatrix} -1 & -1 \\ 1 & 0 \end{pmatrix} = (ST)^2$ et $z' = -1 - \dfrac{1}{z}, \quad z = \rho$.

c : Si $a = 1$ alors $g = \begin{pmatrix} 1 & -1 \\ 1 & 0 \end{pmatrix} = -(TS)^{-2}$ et $z' = 1 - \dfrac{1}{z}, \quad z = -\overline{\rho}$.

ii. Si $d = \pm 1$, alors $b = -1$ et $a = 0$. Donc $g = \begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix} = ST$ et $z' = -\dfrac{1}{z+1}$, $z = -1 - \dfrac{1}{z'}$.

Et par symétrie nous retrouvons les cas précédents.

Ce qui démontre le théorème. □

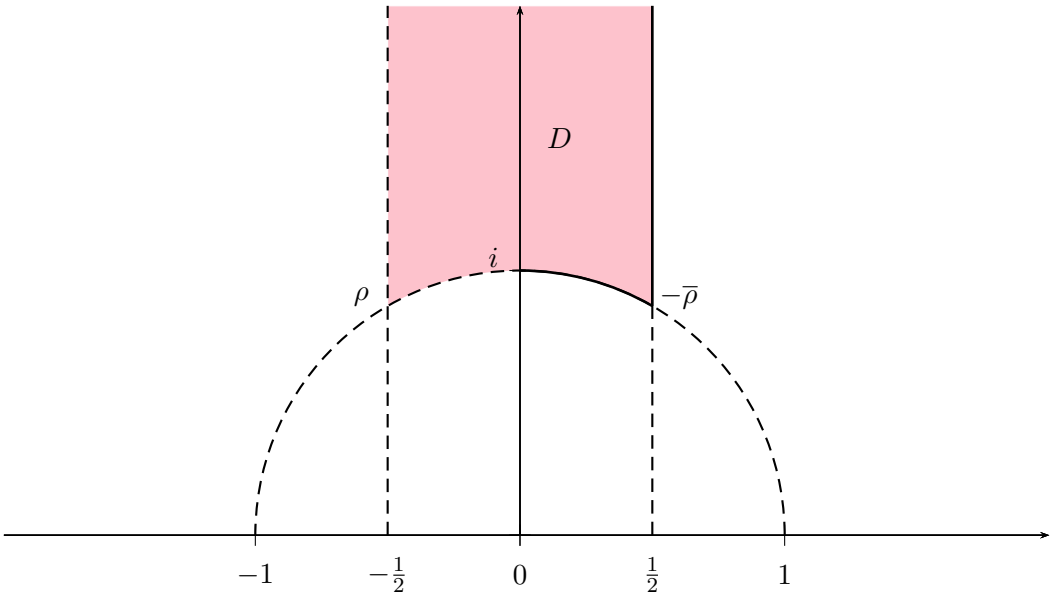


FIGURE 1 – Domaine Fondamental D

La figure suivante représente les transformations du domaine fondamental D par les éléments $\{I_2, T, TS, ST^{-1}S, S, ST, STS, T^{-1}S, T^{-1}\}$ du groupe $SL_2(\mathbb{Z})$.

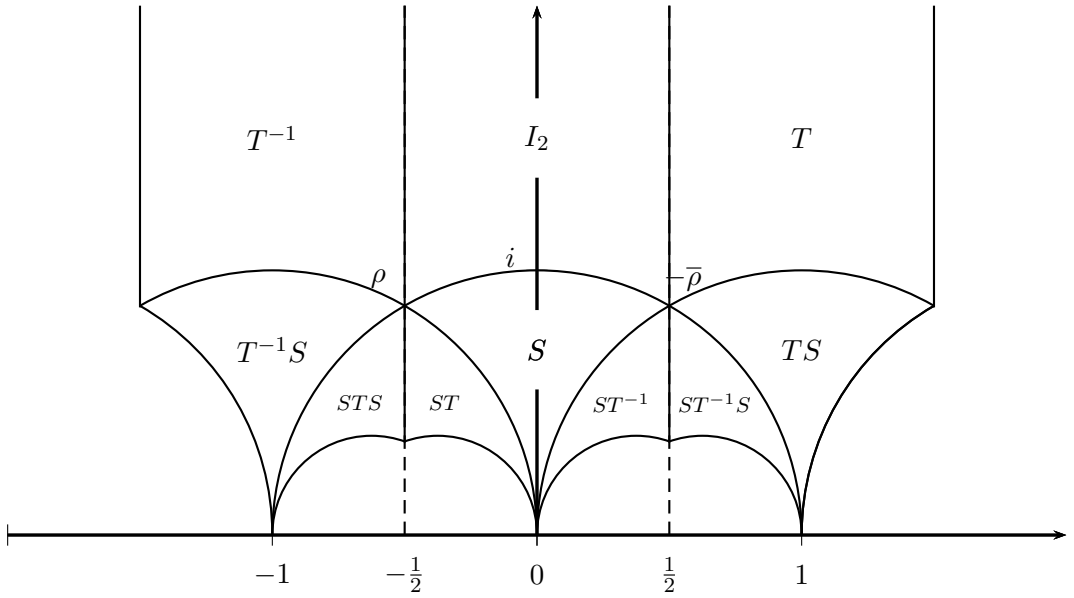


FIGURE 2 – Transformations du domaine fondamental D par les différentes actions du groupe $SL_2(\mathbb{Z})$

Théorème 8. Soit G' le sous-groupe de $SL_2(\mathbb{Z})$ engendré par S et T.
Pour tout $z \in \mathfrak{H}$ il existe $g \in G'$ tel que $gz \in D$.

Démonstration.

Soit $z \in \mathfrak{H}$.

Soit $g = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in G' = \langle S, T \rangle$, où G' est le sous-groupe de $SL_2(\mathbb{Z})$ engendré par les matrices S et T.

Alors $Im(gz) = \dfrac{Im(z)}{|cz+d|^2}$.

La forme quadratique

$$\begin{aligned} \mathbb{R}^2 &\to \mathbb{R} \\ (c,d) &\mapsto |cz+d|^2 \end{aligned}$$

est définie positive. Donc, elle admet un minimum sur $\mathbb{Z}^2 \backslash \{(0,0)\}$.

$\Rightarrow$ Il existe donc un élément $g \in G'$ tel que $Im(gz)$ est maximal.

$\Rightarrow$ Il existe $n \in \mathbb{Z}$ tel que $-\frac{1}{2} < Re(T^n gz) = Re(gz) + n \leq \frac{1}{2}$.

Notons que $Im(T^n gz) = Im(gz + n) = Im(gz)$ et que $T^n g \in G'$.

Supposons que $|T^n gz| < 1$.

Alors $|ST^n gz| = \left|\frac{-1}{T^n gz}\right| = \frac{1}{|T^n gz|} > 1$.
$\Rightarrow \frac{1}{|T^n gz|^2} > 1$

$\Rightarrow Im(ST^n gz) = Im\left(\frac{-1}{T^n gz}\right) = \frac{Im(T^n gz)}{|T^n gz|^2} > Im(T^n gz)$ ce qui contredit la maximalité de $Im(gz)$.

Nous devons donc avoir $|T^n gz| \geq 1$.
Ce qui nous donne $g' = T^n g \in G'$ avec la propriété désiré. □

Corollaire 9. $SL_2(\mathbb{Z})$ est engendré par S et T.

Démonstration.
On remarque que $S^2 = -I_2$ et que $T^n = \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix}$ pour tous $n \in \mathbb{Z}$.
De plus, si $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in SL_2(\mathbb{Z})$, alors

$$SM = \begin{pmatrix} -c & -d \\ a & b \end{pmatrix} \text{ et } T^n M = \begin{pmatrix} a + nc & b + nd \\ c & d \end{pmatrix}$$

Prenons alors $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in SL_2(\mathbb{Z})$.

Si $c = 0$ on a $ad = 1$, ce qui nous laisse deux possibilités :

$$M = \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} = T^m, \text{ d'où } T^{-m} M = I_2$$

ou

$$M = \begin{pmatrix} -1 & m \\ 0 & -1 \end{pmatrix} = -T^{-m}, \text{ d'où } S^2 T^m M = I_2$$

Supposons maintenant que $c \neq 0$. Si $|a| \geq |c|$, on écrit $a = cq + r$ la division euclidienne de a par c, avec $0 \leq r \leq |c|$. Nous pouvons en déduire que la matrice $M' = T^{-q}M$ a un coefficient supérieur gauche $a - qc$ strictement inférieur en valeur absolue à son coefficient inférieur gauche c.
Multiplions à gauche la matrice M' par S afin d'échanger les deux coefficients de tel sorte que :

$$SM' = \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} \text{ vérifie } : |c'| < |c| \text{ et } a' \neq 0.$$

On obtient alors une matrice $M^* = \begin{pmatrix} a^* & b^* \\ 0 & d^* \end{pmatrix}$ dont le coefficient inférieur gauche est nul.

De plus, on sait écrire cette matrice comme le produit de S et T.

□

2 Fractions continues

Définition 10. Une *fraction continue* est une expression de la forme :

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \dots}}}$$

Nous rappelons que $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ et $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ et que pour tout $k \in \mathbb{N}$:

$$Sz = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} z = \frac{0 \times z - 1}{1 \times z + 0} = -\frac{1}{z}$$

$$Tz = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} z = \frac{1 \times z + 1}{0 \times z + 1} = z + 1$$

$$T^k z = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^k z = \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix} z = \frac{1 \times z + k}{0 \times z + 1} = z + k$$

$$T^{-1} z = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^{-1} z = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} z = \frac{1 \times z - 1}{0 \times z + 1} = z - 1$$

$$T^{-k} z = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^{-k} z = \begin{pmatrix} 1 & -k \\ 0 & 1 \end{pmatrix} z = \frac{1 \times z - k}{0 \times z + 1} = z - k$$

Nous pouvons observer que des fractions continues apparaissent dans l'action de $SL_2(\mathbb{Z})$ sur le demi-plan de Poincaré $\mathfrak{H}$.

En effet, si nous voulons ramener un point quelconque $z \in \mathfrak{H}$ dans le domaine fondamental D, nous savons qu'il existe $n_1, n_2, ..., n_r \in \mathbb{Z}$ tel que $T^{n_r} S T^{n_{r-1}} S ... S T^{n_2} S T^{n_1} z \in D$.

En calculant les coordonnées de ce nouveau point, nous rencontrons les fractions continues :

$$\begin{aligned} T^{n_r} S T^{n_{r-1}} S ... S T^{n_2} S T^{n_1} z &= T^{n_r} S T^{n_{r-1}} S ... S T^{n_2} S \left(z + n_1 \right) \\ &= T^{n_r} S T^{n_{r-1}} S ... S T^{n_2} \left(\frac{-1}{z + n_1} \right) \\ &= T^{n_r} S T^{n_{r-1}} S ... S \left(n_2 + \frac{-1}{z + n_1} \right) \\ &= T^{n_r} S T^{n_{r-1}} S ... \left(\frac{-1}{n_2 + \frac{-1}{z + n_1}} \right) \end{aligned}$$

$$
\begin{aligned}
&= T^{n_r} S T^{n_{r-1}} S ... \left(\frac{-1}{n_3 + \frac{-1}{n_2 + \frac{-1}{z+n_1}}} \right) \\
&= T^{n_r} S T^{n_{r-1}} \left(\frac{-1}{n_{r-2} + ... \frac{-1}{n_3 + \frac{-1}{n_2 + \frac{-1}{z+n_1}}}} \right) \\
&= T^{n_r} S \left(n_{r-1} + \frac{-1}{n_{r-2} + ... \frac{-1}{n_3 + \frac{-1}{n_2 + \frac{-1}{z+n_1}}}} \right) \\
&= T^{n_r} \left(\frac{-1}{n_{r-1} + \frac{-1}{n_{r-2} + ... \frac{-1}{n_3 + \frac{-1}{n_2 + \frac{-1}{z+n_1}}}}} \right) \\
&= n_r + \frac{-1}{n_{r-1} + \frac{-1}{n_{r-2} + ... \frac{-1}{n_3 + \frac{-1}{n_2 + \frac{-1}{z+n_1}}}}}
\end{aligned}
$$

Nous pouvons observer l'apparition d'une fraction continue qui gagne de plus en plus en profondeur. D'ailleurs nous savons que $r \in \mathbb{N}$, donc cette fraction continue est finie.

Voici un exemple plus précis d'une fraction continue résultante de l'action de $SL_2(\mathbb{Z})$ sur $\mathfrak{H}$.

$$
\begin{aligned}
ST^4ST^2ST^3z &= T^4ST^2S(T^3z) \\
&= ST^4ST^2S(z+3) \\
&= ST^4ST^2 \frac{-1}{z+3} \\
&= ST^4S \left(2 + \frac{-1}{z+3} \right) \\
&= ST^4 \left(\frac{-1}{2 + \frac{-1}{z+3}} \right) \\
&= S \left(4 + \frac{-1}{2 + \frac{-1}{z+3}} \right) \\
&= \frac{-1}{4 + \frac{-1}{2 + \frac{-1}{z+3}}}
\end{aligned}
$$

3 Surfaces triangulaires de Hecke

Soient $q \geq 3$ un entier et $\lambda_q := 2\cos\left(\frac{\pi}{q}\right)$.

Soient $S := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ et $T_q := \begin{pmatrix} 1 & \lambda_q \\ 0 & 1 \end{pmatrix}$.

Si S et T_q satisfont la relation $(ST_q)^q = -I_2$, alors G_q est le $q^{\text{ième}}$ groupe de triangles de Hecke engendré par S et T_q.

$$Sz = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} z = \frac{0 \times z - 1}{1 \times z + 0} = -\frac{1}{z}$$

$$T_q z = \begin{pmatrix} 1 & \lambda_q \\ 0 & 1 \end{pmatrix} z = \frac{1 \times z + \lambda_q}{0 \times z + 1} = z + \lambda_q$$

Nous avons de plus que

$$T_q^{-1} = \begin{pmatrix} 1 & -\lambda_q \\ 0 & 1 \end{pmatrix}$$

$$ST_q = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & \lambda_q \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & \lambda_q \end{pmatrix}$$

Nous allons a présent voir quelques exemples.

Pour $q = 3$, nous rencontrons le cas spécial où $T_3 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = T$ avec $\lambda_3 = 2\cos\left(\frac{\pi}{3}\right) = 1$.

Nous pouvons vérifier que la relation $(ST_3)^3 = \begin{pmatrix} -\lambda_3 & 1-\lambda_3^2 \\ -1+\lambda_3^2 & \lambda_3^3 - 2\lambda_3 \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = I_2$ est satisfaite.

Pour $q = 4$, nous avons $T_4 = \begin{pmatrix} 1 & \lambda_4 \\ 0 & 1 \end{pmatrix}$ avec $\lambda_4 = 2\cos\left(\frac{\pi}{4}\right) = \sqrt{2}$.

$$(ST_4)^4 = \begin{pmatrix} 1-\lambda_4^2 & -\lambda_4^3 + 2\lambda_4 \\ \lambda_4^3 - 2\lambda_4 & \lambda_4^4 - 3\lambda_4^2 + 1 \end{pmatrix} = \begin{pmatrix} 1-\sqrt{2}^2 & -\sqrt{2}^3 + 2\sqrt{2} \\ \sqrt{2}^3 - 2\sqrt{2} & \sqrt{2}^4 - 3\sqrt{2}^2 + 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = I_2$$

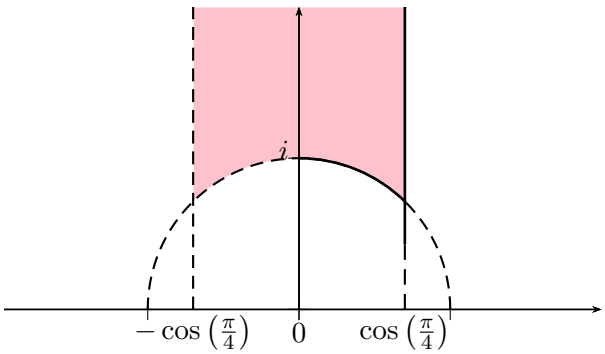


FIGURE 3 – Cas où $q = 4$ donc $\lambda_4 = 2\cos\left(\frac{\pi}{4}\right)$

Pour $q = 6$, nous avons $T_6 = \begin{pmatrix} 1 & \lambda_6 \\ 0 & 1 \end{pmatrix}$ avec $\lambda_6 = 2\cos\left(\frac{\pi}{6}\right) = \sqrt{3}$.

$(ST_6)^6 = \begin{pmatrix} -\lambda_6^4 + 3\lambda_6^2 - 1 & -\lambda_6^5 + 4\lambda_6^3 - 3\lambda_6 \\ \lambda_6^5 - 4\lambda_6^3 + 3\lambda_6 & \lambda_6^6 - 5\lambda_6^4 + 6\lambda_6^2 - 1 \end{pmatrix} = \begin{pmatrix} -\sqrt{3}^4 + 3\sqrt{3}^2 - 1 & -\sqrt{3}^5 + 4\sqrt{3}^3 - 3\sqrt{3} \\ \sqrt{3}^5 - 4\sqrt{3}^3 + 3\sqrt{3} & \sqrt{3}^6 - 5\sqrt{3}^4 + 6\sqrt{3}^2 - 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = I_2$

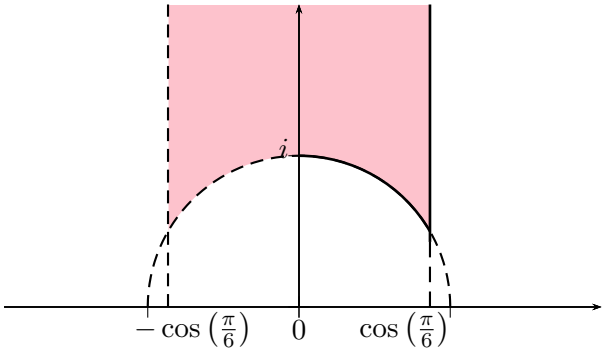


FIGURE 4 – Cas où $q = 6$ donc $\lambda_6 = 2\cos\left(\frac{\pi}{6}\right)$

Pour $q = 7$, nous avons $T_7 = \begin{pmatrix} 1 & \lambda_7 \\ 0 & 1 \end{pmatrix}$ avec $\lambda_7 = 2\cos\left(\frac{\pi}{7}\right)$.

$(ST_7)^7 = \begin{pmatrix} -\lambda_7^5 + 4\lambda_7^3 - 3\lambda_7 & -\lambda_7^6 + 5\lambda_7^4 - 6\lambda_7^2 + 1 \\ \lambda_7^6 - 5\lambda_7^4 + 6\lambda_7^2 - 1 & \lambda_7^7 - 6\lambda_7^5 + 10\lambda_7^2 - 4\lambda_7 \end{pmatrix} = I_2$

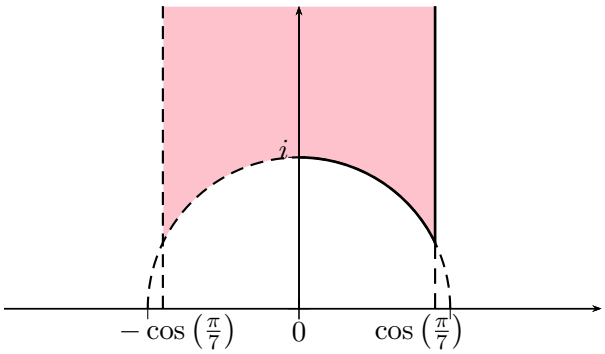


FIGURE 5 – Cas où $q = 7$ donc $\lambda_7 = 2\cos\left(\frac{\pi}{7}\right)$

4 Programmes

Dans nos programmes, nous allors prioritairement travailler avec la librairie `matplotlib` et `pygame`.

`Matplotlib` est une librairie qui permet de tracer des graphes. Elle est basé sur la notion d'`axe` qui n'est rien d'autre qu'un graphe individuel appartenant à une figure. Les `axis` sont les axes de coordonnées appartenant à un objet axe.

`Pygame` est un module qui offre des outils permettant de créer des jeux. Pour les animations, nous devons alors travailler avec des images qui sont enfilées les unes derrières les autres ce qui donne finalement lieu à une animation fluide. `Pygame` travaille avec des pixels, ce qui rend notre tâche plus difficile, car contrairement à la libraire `matplotlib`, il n'y a pas de notion d'axe.

C'est pour cela que nous avons écrit deux programmes d'introduction avec `matplotlib` pour nous familiariser avec les algorithmes et les chemins de réflexion. Ensuite, nous avons créée les animations à l'aide de `pygame`.

4.1 Programme d'introduction avec Matplotlib

L'objectif de ce programme d'introduction est de pouvoir visualiser l'action du groupe sur des points arbitraires dans le demi-plan de Poincaré $\mathfrak{H}$. C'est le premier programme que nous avons établi et il nous a servi comme base pour les programmes à venir. Il permet à l'utilisateur de jouer avec les transformations par les matrices S, T et T^{-1}, le but étant de trouver le bon chemin, c'est-à-dire la bonne combinaison des matrices S, T et T^{-1}, pour arriver dans le domaine fondamental.

Tout d'abord nous commençons par importer toutes les librairies nécessaires pour notre programme. Plus précisément, nous importons la librairie `numpy` à cause de la fonction `arrange()` qui retourne une liste de valeurs régulièrement espacées dans un intervalle prédéfini. Nous importons la librairie `math` pour pouvoir utiliser la fonction `sqrt()` qui retourne la racine carrée. Nous importons les librairies `pylab` et `pyplot` pour la représentation visuelle et nous utilisons la librairie `random` pour générer des valeurs aléatoires.

```
1 import numpy as np
2 from math import sqrt
3 import matplotlib.pyplot as plt
4 from pylab import *
5 from random import randint,random,uniform
```

Une fois les librairies nécessaires importées, nous pouvons commencer avec le coeur du programme. Nous voulons visualiser le domaine fondamental D pour l'action du groupe modulaire

sur le demi-plan de Poincaré (voir définition-proposition 7) où D est défini comme suit :

$$D = \{z \in \mathfrak{H} : (-\frac{1}{2} < Re(z) < 0 \text{ et } |z| > 1) \text{ ou } (0 \leq Re(z) \leq \frac{1}{2} \text{ et } |z| \geq 1)\}.$$

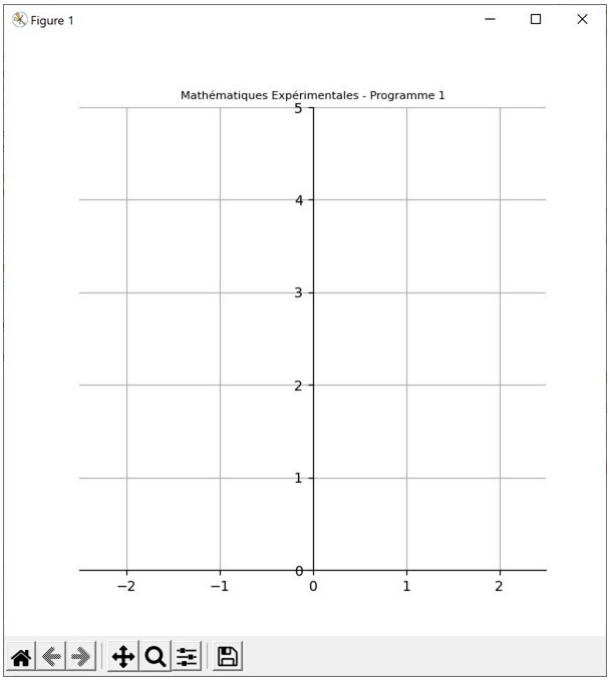
Nous devons donc d'abord créer une fenêtre et des axes. Une fenêtre est initialement une fenêtre vide, c'est-à-dire une fenêtre sans axes. En appelant la fonction `figure()`, nous créons une telle fenêtre vide.

Ensuite nous appelons la fonction `add_subplot()` afin de créer des axes. Les trois arguments de cette fonction désignent le nombre de rangées dans lesquelles la fenêtre est partagée, le nombre de colonnes dans lesquelles la fenêtre est partagée et la position du graphe dans ce nouveau quadrillage. Comme nous ne voulons qu'un seul graphe, les trois arguments sont attribués à des 1.

Une fois les axes créés, nous embellissons encore l'aspect visuel de la fenêtre et du repère. Nous ordonnons par exemple que l'origine $(0, 0)$ du repère soit bien centrée dans notre fenêtre. Nous définissons aussi la champ qui est initialement visible en fonction de l'axe des abscisses et de l'axe des ordonnées. Noter que dans l'environnement `matplotlib`, l'utilisateur a la possibilité d'agrandir et de faire bouger les axes et il peut ainsi changer la vue du repère. Nous définissons donc uniquement la vue de départ pour un utilisateur qui peut alors "jouer" librement avec le repère.

```
# Créer une fenêtre
fig = plt.figure(figsize=(6, 6))

# Créer des axes
ax = fig.add_subplot(1,1,1)

ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
plt.grid()
plt.ylim([0,5])
plt.xlim([-2.5,2.5])

# Créer et afficher un titre pour le graphique
plt.title('Mathématiques Expérimentales - Programme 1', fontsize=8)
```

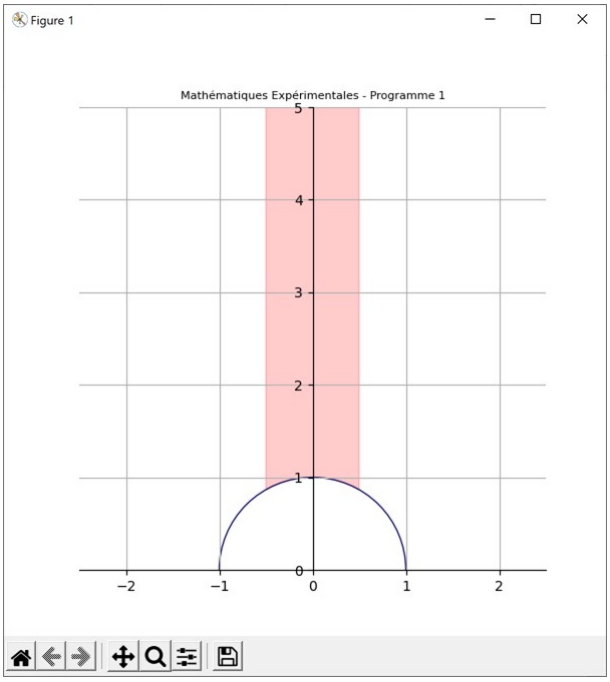
Si nous compilons ceci en ajoutant la commande `plt.show()` qui nous permet de visualiser la fenêtre et les axes créés, nous obtenons la fenêtre suivante :



Désormais nous avons le fondement qui nous permet de construire notre domaine fondamental D. Pour cela, nous créons un cercle de centre 0 et de rayon 1 et nous l'ajoutons à notre graphique. Ensuite nous colorions le domaine fondamental, ce qui est assez délicat, comme il n'y a pas de fonction prédéfinie qui nous permet de colorier l'intérieur d'un cercle. Ainsi, nous créons d'abord une liste de réels régulièrement espacés dans l'intervalle qui va de -0.5 à 0.5 avec un incrément de 0.0001. Nous parcourons maintenant cette liste et pour chaque réel x dans cette liste, nous dessinons la droite verticale qui va de $\sqrt{1-x^2}$ à l'infini. En effet, $(x, \sqrt{1-x^2})$ dénote le point d'abscisse x qui se trouve sur le cercle unitaire. De cette manière nous colorions tous les points au-dessus du cercle unitaire dont l'abscisse se trouve entre -0.5 et 0.5.

```
# Créer un cercle et l'ajouter au graphique
circle = plt.Circle((0,0),1,fill=False, color="midnightblue")
ax.add_artist(circle)

# Colorier le domaine fondamental
x = np.arange(-0.5,0.5,0.0001)
domaine = ax.fill_between(x,sqrt(1-x**2),10**10 , alpha=0.2, color="red")
```

En compilant notre code après y avoir ajouté la fonction `plt.show()`, nous obtenons la fenêtre suivante :



Jusqu'à présent, nous nous sommes juste occupés de la représentation de la fenêtre et de son graphique. Nous passons maintenant à la partie plus intéressante. Nous voulons désormais générer un point aléatoire sur le graphique. Pour cela, nous définissons la partie réelle de notre point en appelant la fonction `uniform()`. `uniform(-2,2)` retourne une valeur réelle aléatoire entre -2 et 2. Nous définissons la partie imaginaire de notre nombre complexe en appelant la fonction `random()`, qui retourne une valeur réelle aléatoire entre 0 et 1.

De plus, nous initialisons deux listes `xpoint_list` et `ypoint_list` qui contiennent respectivement la partie réelle et la partie imaginaire de notre point. Ces listes vont nous permettre de stocker toutes les informations sur les points qui vont être générés en appliquant les matrices S, T et T^{-1} à notre point initial. Nous allons alors simplement ajouter les coordonnées de ces nouveaux points aux listes afin de ne pas perdre d'informations.

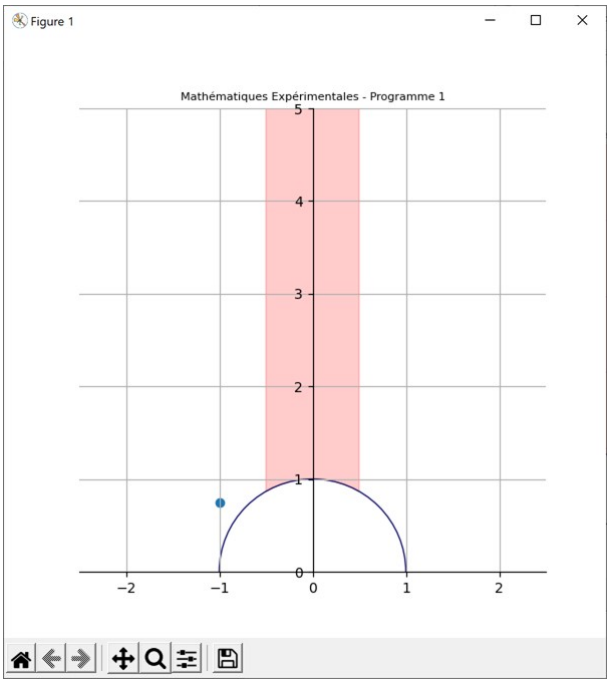
Le point initial a été crée et nous devons maintenant le rendre visible sur le graphique. La fonction `ax.scatter()` prend deux arguments qui doivent être sous forme de liste d'une même longueur. Soit l la longueur des listes entrées en tant qu'arguments. Alors pour tout i dans $\{1, 2, ..., l\}$, la fonction dessine le point dont l'abscisse est le $i^{\text{ème}}$ élément de la première liste et dont l'ordonnée est le $i^{\text{ème}}$ élément de la deuxième liste. `ax.scatter()` dessine donc en tout l points. Nous utilisons désormais cette fonction pour dessiner notre point initial et pour le rendre visible à l'utilisateur sur notre graphique.

Nous voulons aussi que notre point s'affiche sous forme de nombre complexe dans la console, c'est pourquoi nous utilisons la fonction `print()`. Nous arrondissons les parties réelles et imaginaires au centième près avec la fonction `round()`, comme il se peut que le programme ait généré des

nombres décimaux avec beaucoup de places après la virgule.

```
# Générer un point aléatoire de départ
xpoint = uniform(-2,2)
ypoint = random()

# Créer deux listes de stockage
xpoint_list = [xpoint]
ypoint_list = [ypoint]

# Dessiner les points sur le graphique
ax.scatter(xpoint_list,ypoint_list)

# Afficher le point généré
print(str(round(xpoint,2)) + " + " + str(round(ypoint,2)) + "i")
```

En ajoutant `plt.show()`, nous obtenons la représentation suivante :



Dans la console, nous voyons le texte suivant :

```
-0.99 + 0.74i
```

Cela signifie que pour cette compilation, le point aléatoirement choisi par le programme est égal à $-0.99 + 0.74i$.

Construisons maintenant la fonction `S_Spiegelung()`.

Rappelons-nous que la matrice S est définie par

$$S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

Soit $z := x + iy \in \mathfrak{H}$. Nous avons alors que

$$\begin{aligned} Sz &= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} z \\ &= -\frac{1}{z} \\ &= -\frac{1}{x+iy} \\ &= -\frac{x-iy}{x^2+y^2} \\ &= \frac{-x}{x^2+y^2} + \frac{y}{x^2+y^2} i \end{aligned}$$

Nous savons alors qu'en appliquant la matrice S à un point $x + iy$ dans $\mathfrak{H}$, nous allons obtenir le point de coordonnées $\left(\frac{-x}{x^2+y^2}, \frac{y}{x^2+y^2}\right)$.

La fonction `S_Spiegelung()` que nous allons maintenant implémenter utilise cette relation. En effet, la fonction `S_Spiegelung()` applique la matrice S au dernier point dans la liste et ajoute ses coordonnées aux listes respectives par la fonction `append()`. Puis la fonction va colorier tous les anciens points en gris et elle va colorier le nouveau point dans une couleur aléatoire.

Ensuite la fonction contrôle si le nouveau point se situe dans le domaine fondamental. Pour un point $x + iy$ dans $\mathfrak{H}$, être dans le domaine fondamental signifie que soit $-0.5 < x < 0$ et $y^2 > \sqrt{1-x^2}$ ou soit $0 \leq x \leq 0.5$ et $y^2 \geq \sqrt{1-x^2}$. Si le nouveau point satisfait à ces conditions, le domaine fondamental va devenir vert et sinon il va rester rouge.

La fonction `S_Spiegelung()` marque aussi dans la console qu'elle a été utilisée en imprimant "S". Elle y affiche ensuite le nouveau point obtenu.

```
1 def S_Spiegelung(val):
2     newx = - xpoint_list[-1]/(xpoint_list[-1]**2 + ypoint_list[-1]**2)
3     newy = ypoint_list[-1]/(xpoint_list[-1]**2 + ypoint_list[-1]**2)
4     xpoint_list.append(newx)
5     ypoint_list.append(newy)
6     ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="gray")
7     ax.scatter(xpoint_list[-1],ypoint_list[-1])
```

```
    if (-0.5 < xpoint_list[-1] < 0 and ypoint_list[-1]> sqrt(1-xpoint_list
    [-1]**2)) or (0 <= xpoint_list[-1] <= 0.5 and ypoint_list[-1]>= sqrt(1-
    xpoint_list[-1]**2)):
        domaine.set_color("green")
    else:
        domaine.set_color("red")
    print("S")
    print(str(round(xpoint_list[-1],2)) + " + " + str(round(ypoint_list[-1],2))
    + "i")
```

Il est important d'ajouter que si nous compilons notre programme avec cette nouvelle fonction, rien ne change par rapport à la version précédente. En effet, cette fonction `S_Spiegelung()` n'est appliquée que si elle est appelée à un endroit précis dans le code. Pour le moment, nous l'avons simplement définie.

Par le même principe, nous construisons maintenant la fonction `T_Translation()`.
Rappelons-nous que la matrice T est définie par

$$T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Soit $z := x + iy \in \mathfrak{H}$. Nous avons alors que

$$\begin{aligned} Tz &= \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} z \\ &= z + 1 \\ &= (x+1) + iy \end{aligned}$$

Nous savons alors qu'en appliquant la matrice T à un point $x + iy$ dans $\mathfrak{H}$, nous allons obtenir un nouveau point de coordonnées $(x + 1, y)$.

La fonction `T_Translation()` que nous allons implémenter utilise cette relation. Cette fonction est très similaire à la fonction `S_Spiegelung()`. Elle aussi calcule les coordonnées du nouveau point en appliquant la matrice T au dernier point dans la liste et les ajoutes ensuite aux listes respectives. Ensuite elle dessine les points sur le graphique et vérifie si le nouveau point se trouve dans le domaine fondamental. Elle change la couleur du domaine si cela est le cas. La fonction marque ensuite dans la console qu'elle a été utilisée en imprimant "T". Elle y affiche aussi le nouveau point obtenu.

```
def T_Translation(val):
    newx = xpoint_list[-1] + 1
```

```
    newy = ypoint_list[-1]
    xpoint_list.append(newx)
    ypoint_list.append(newy)
    ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="gray")
    ax.scatter(xpoint_list[-1],ypoint_list[-1])
    if (-0.5 < xpoint_list[-1] < 0 and ypoint_list[-1]> sqrt(1-xpoint_list
    [-1]**2)) or (0 <= xpoint_list[-1] <= 0.5 and ypoint_list[-1]>= sqrt(1-
    xpoint_list[-1]**2)):
        domaine.set_color("green")
    else:
        domaine.set_color("red")
    print("T")
    print(str(round(xpoint_list[-1],2)) + " + " + str(round(ypoint_list[-1],2))
    + "i")
```

Par le même principe, nous construisons désormais la fonction `T_TranslationInverse()`. Rappelons-nous que la matrice T^{-1} est définie par

$$T^{-1} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}.$$

Soit $z := x + iy \in \mathfrak{H}$. Nous avons alors que

$$\begin{aligned} T^{-1}z &= \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} z \\ &= z - 1 \\ &= (x-1) + iy \end{aligned}$$

Nous savons alors qu'en appliquant la matrice T^{-1} à un point $x+iy$ dans $\mathfrak{H}$, nous allons obtenir un nouveau point de coordonnées $(x-1, y)$.

La fonction `T_TranslationInverse()` que nous allons implémenter utilise cette relation. La fonction ressemble aux deux fonctions précédentes. Elle calcule les coordonnées du nouveau point et les ajoutes aux listes. Ensuite elle dessine le nouveau point sur le graphique et vérifie s'il se trouve dans le domaine fondamental, qui lui va changer de couleur si c'est le cas. La fonction marque ensuite dans la console qu'elle a été utilisée en imprimant "T$\hat{}$(-1)". Elle y affiche aussi le nouveau point obtenu.

```
def T_TranslationInverse(val):
    newx = xpoint_list[-1] - 1
    newy = ypoint_list[-1]
    xpoint_list.append(newx)
```

```
    ypoint_list.append(newy)
    ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="gray")
    ax.scatter(xpoint_list[-1],ypoint_list[-1])
    if (-0.5 < xpoint_list[-1] < 0 and ypoint_list[-1]> sqrt(1-xpoint_list
    [-1]**2)) or (0 <= xpoint_list[-1] <= 0.5 and ypoint_list[-1]>= sqrt(1-
    xpoint_list[-1]**2)):
        domaine.set_color("green")
    else:
        domaine.set_color("red")
    print("T^(-1)")
    print(str(round(xpoint_list[-1],2)) + " + " + str(round(ypoint_list[-1],2))
    + "i")
```

Maintenant nous voulons créer des boutons qui exercent une certaine commande après avoir été cliqué. En effet, nous voulons créer 3 boutons différents, un premier qui appelle la fonction `S_Spiegelung()` lorsqu'on le clique, un deuxième qui appelle la fonction `T_Translation()` lorsqu'on le clique et un troisième qui appelle la fonction `T_TranslationInverse()` lorsqu'on le clique.

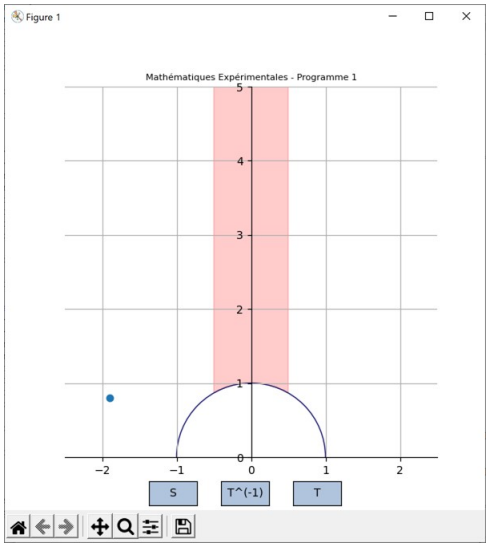
Par la fonction `axes()`, nous déterminons l'endroit respectif des boutons. La classe `Button` nous crée un bouton, mais nous devons spécifier son endroit, son nom et sa couleur. Ensuite la fonction `on_clicked()` lie une certaine action à un bouton. Nous lions la fonction `S_Spiegelung()` au bouton nommé "S", la fonction `T_Translation()` au bouton nommé "T" et la fonction `T_TranslationInverse()` au bouton nommé "T$\hat{}$(-1)". Les fonctions sont appelées dès que l'utilisateur clique sur le bouton qui leur est associé.

```
# Générer le boutton S
s_axes = plt.axes([0.3, 0.01, 0.1, 0.05])
s_button = Button(s_axes, 'S',color="lightsteelblue")
s_button.on_clicked(S_Spiegelung)

# Générer le bouton T
t_axes = plt.axes([0.6, 0.01, 0.1, 0.05])
t_button = Button(t_axes, 'T',color="lightsteelblue")
t_button.on_clicked(T_Translation)

# Générer le bouton T^(-1)
tinv_axes = plt.axes([0.45, 0.01, 0.1, 0.05])
tinv_button = Button(tinv_axes, 'T^(-1)',color="lightsteelblue")
tinv_button.on_clicked(T_TranslationInverse)
```

Finalement, il nous reste qu'à ajouter la fonction `show()` à notre code pour rendre la fenêtre visible.

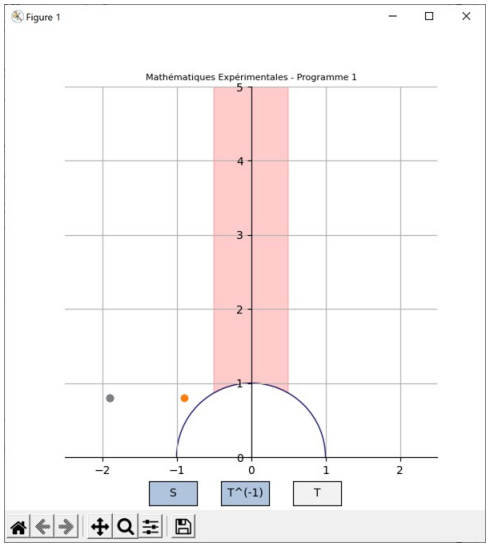
```
plt.show()
```

Voici un exemple de compilation.
Lorsque l'utilisateur compile le programme, un point aléatoire est crée. L'utilisateur voit un affichage similaire à celui-ci :



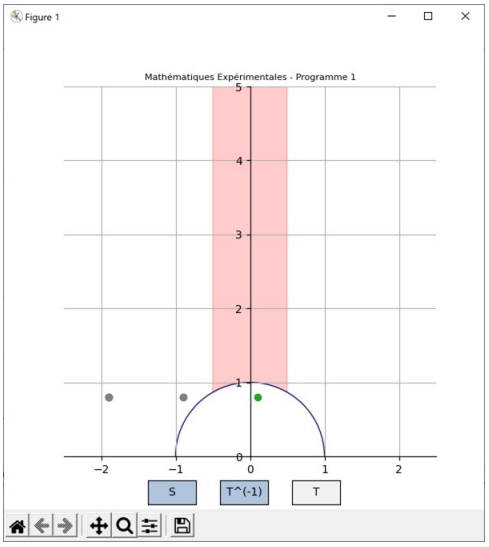

```
-1.89 + 0.8i
```

L'utilisateur est ensuite amené à chercher le bon "chemin" pour arriver dans le domaine fondamental. Bien sûr il peut aussi expérimenter et jouer avec les boutons, mais le but ultime est quand même d'arriver dans le domaine fondamental. Disons que l'utilisateur décide de cliquer sur le bouton "T". Alors nous avons l'affichage suivant :



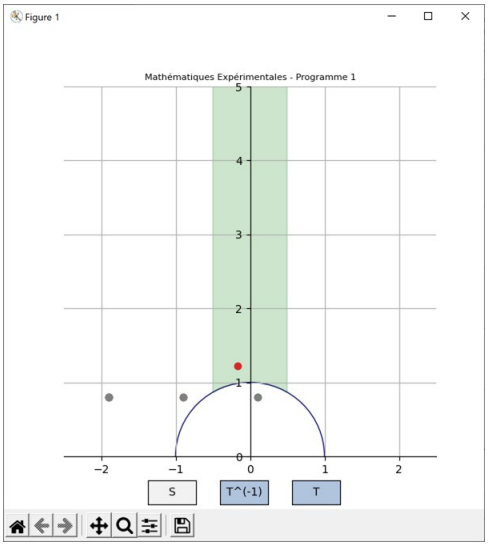
```
-1.89 + 0.8i
T
-0.89 + 0.8i
```

Disons que l'utilisateur décide alors de cliquer de nouveau sur le bouton "T". Alors nous avons l'affichage suivant :




```
-1.89 + 0.8i
T
-0.89 + 0.8i
T
0.11 + 0.8i
```

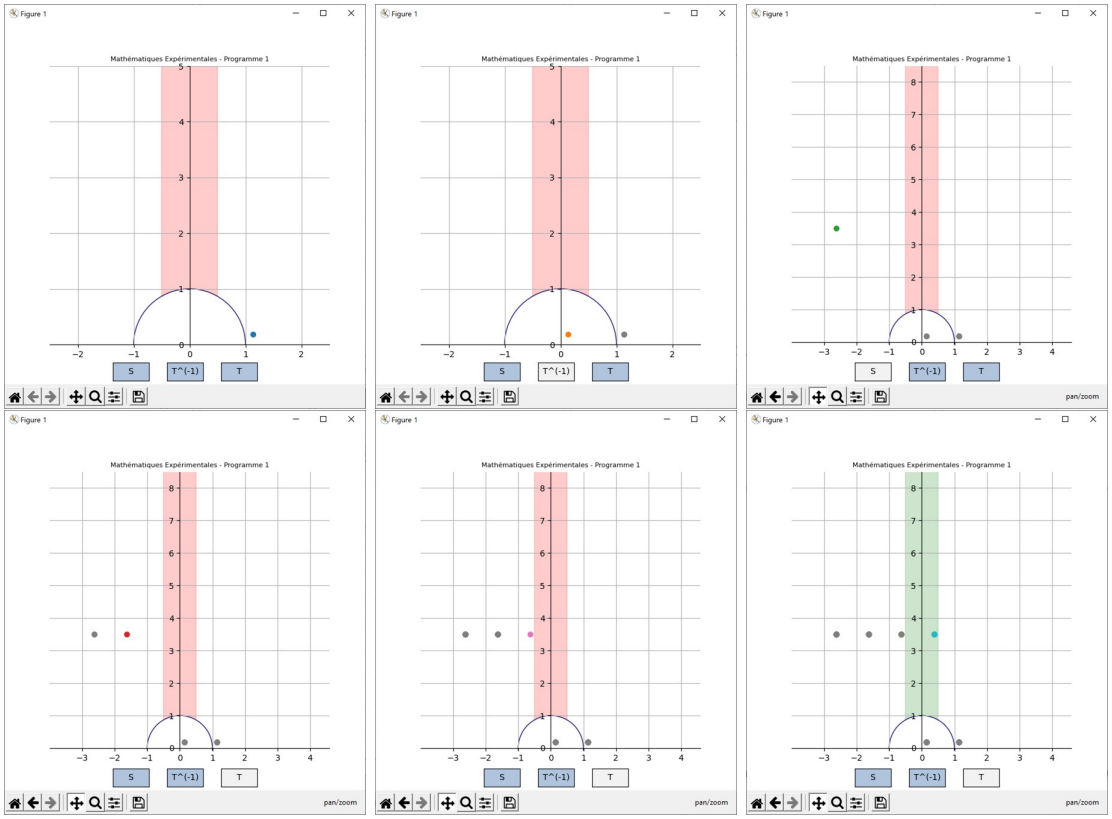
Disons que l'utilisateur décide ensuite de cliquer sur le bouton "S". Alors nous obtenons l'affichage suivant :



```
-1.89 + 0.8i
T
-0.89 + 0.8i
T
0.11 + 0.8i
S
-0.16 + 1.22i
```

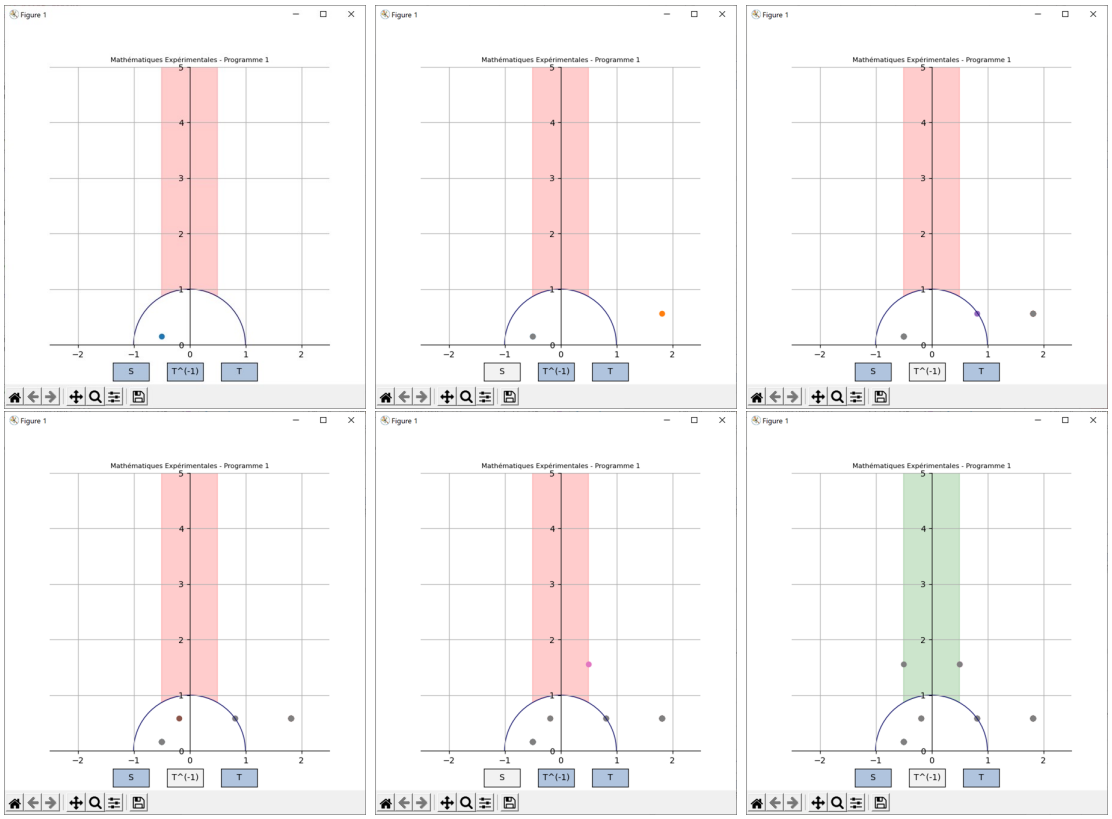
L'utilisateur est arrivé dans le domaine fondamental qui est devenu vert.
Nous remarquons que l'affichage dans la console grandit avec chaque clique sur un bouton. Aucune information n'est perdue et l'utilisateur peut facilement retracer son chemin.

Voici deux autres exemples de compilation et d'utilisation du programme. Veuillez noter que si un affichage contient un bouton gris, alors il s'agit du bouton poussé pour passer de la dernière photo à celle-la.




```
1.14 + 0.18i
T(-1)
0.14 + 0.18i
S
```

```
-2.63 + 3.49i
T
-1.63 + 3.49i
T
-0.63 + 3.49i
T
0.37 + 3.49i
```




```
-0.5 + 0.16i
S
1.81 + 0.58i
T(-1)
0.81 + 0.58i
T(-1)
-0.19 + 0.58i
S
0.51 + 1.56i
T(-1)
-0.49 + 1.56i
```

4.2 Programme pour la compréhension de l'algorithme avec Matplotlib

L'objectif de ce programme est de nous familiariser avec l'algorithme qui trouve lui-même la bonne combinaison des matrices S, T et T^{-1} permettant d'amener un point initial dans le domaine fondamental. Nous pouvons observer étape par étape si notre programme fait la bonne décision, c'est-à-dire s'il choisit la bonne matrice à appliquer. Le "chemin" du point est visualisé et l'utilisateur doit appuyer sur le bouton "Next" pour observer le parcours du point à son propre rythme. Ce programme reprend beaucoup de code du programme 4.1 et est une excellente base pour l'animation dans le programme qui va suivre celui-ci.

Au début de notre programme nous importons toutes les librairies nécessaires. Nous créons ensuite une fenêtre et y ajoutons des axes. Pour l'aspect visuel, nous définissons encore l'affichage initial que l'utilisateur va voir lors de la compilation du programme. Puis nous ajoutons le demi-cercle et le domaine fondamental au graphique. Cette première partie du code est identique au début du code du programme d'introduction 4.1.

```
import numpy as np
from math import sqrt
import matplotlib.pyplot as plt
from pylab import *
from random import randint,random,uniform

# Créer une fenêtre
fig = plt.figure(figsize=(6, 6))

# Créer des axes
ax = fig.add_subplot(1,1,1)

ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
plt.grid()
plt.ylim([0,5])
plt.xlim([-2.5,2.5])

# Créer un cercle et l'ajouter au graphique
circle = plt.Circle((0,0),1,fill=False, color="midnightblue")
ax.add_artist(circle)

# Colorier le domaine fondamental
x = np.arange(-0.5,0.5,0.0001)
domaine = ax.fill_between(x,sqrt(1-x**2),10**10 , alpha=0.2, color="green")
```

Nous définissons maintenant quelques variables qui vont apparaître tout au long du programme. Nous définissons d'un côté la liste `way_to_domain` dans laquelle seront ajoutées les différentes matrices nécessaires pour amener un certain point dans le domaine fondamental. D'un autre

côté, nous définissons la chaîne de caractère `way_detail` qui précise le chemin du point et que l'on va afficher dans la console après chaque action. Veuillez noter que par chemin d'un point $z \in \mathfrak{H}$ nous entendons la bonne combinaison des matrices S, T et T^{-1} qui, appliquée à z, retourne un point dans le domaine fondamental.

```
way_to_domain = ["z"]
way_detail = "z"
```

Nous définissons maintenant une nouvelle fonction `point_in_domain()` qui prend la partie réelle et la partie imaginaire en tant que paramètres. Cette fonction vérifie si un point donné se situe dans le domaine fondamental. Si cela est le cas, elle change la couleur du domaine fondamental. Cette fonction est très pratique, car elle peut facilement être appelée et évite ainsi des nombreuses lignes de structures conditionnelles `if` et `else`. Elle va être appelée à chaque fois qu'un nouveau point apparaît dans nos calculs.

```
def point_in_domain(x,y):
    if (-0.5 < x < 0 and y > sqrt(1-x**2)) or (0 <= x <= 0.5 and y >= sqrt(1-x
    **2)):
        domaine.set_color("green")
        return True
    else:
        domaine.set_color("red")
        return False
```

Nous définissons à présent un point initial arbitraire à l'aide des fonctions `random()` et `uniform()`. Nous créons deux listes différentes, dont une va contenir les parties réelles des points et l'autre la partie imaginaire. Par la fonction `ax.scatter()`, nous affichons ce point sur notre graphique. Simultanément, nous contrôlons à l'aide de la fonction antérieurement créée si ce point se trouve dans le domaine fondamental ou non. Puis ce point est affiché dans la console.

```
xpoint = uniform(-2,2)
ypoint = random()
xpoint_list = [xpoint]
ypoint_list = [ypoint]
ax.scatter(xpoint_list,ypoint_list)
point_in_domain(xpoint_list[-1],ypoint_list[-1])
print("z = " +str(round(xpoint,2)) + " + " + str(round(ypoint,2)) + " i ")
```

Ensuite, nous définissons une nouvelle fonction `new_try()` qui a comme but de définir un nouveau point de départ qui doit alors à son tour trouver son chemin dans le domaine fondamental. Pour cela, la fonction change la couleur de tous les anciens point pour les mettre dans l'arrière-plan. Ensuite, elle vide toutes nos listes et variables, pour pouvoir définir un tout nouveau point de départ. Elle crée ensuite ce nouveau point arbitraire qu'elle affiche sur le graphique et qu'elle

ajoute aux listes respectives. Elle appelle aussi la fonction `point_in_domain` et affiche le nouveau point de départ dans la console.

```
def new_try(val):
    ax.scatter(xpoint_list,ypoint_list,color="gainsboro")
    global way_detail
    xpoint_list.clear()
    ypoint_list.clear()
    xpoint_list.append(uniform(-2,2))
    ypoint_list.append(random())
    ax.scatter(xpoint_list,ypoint_list)
    way_to_domain.clear()
    way_to_domain.append("z")
    way_detail = "z"
    point_in_domain(xpoint_list[-1],ypoint_list[-1])
    print()
    print("z = " +str(round(xpoint,2)) + " + " + str(round(ypoint,2)) + " i ")
```

Similairement au programme d'introduction 4.1, nous créons les trois fonctions `S_Spiegelung()`, `T_Translation()` et `T_TranslationInverse()` qui appliquent les matrices respectives S, T et T^{-1} au dernier point dans la liste. À l'intérieur de ces fonctions, la fonction `point_in_domain()` est appelée pour contrôler si le nouveau point créé se trouve dans le domaine fondamental.

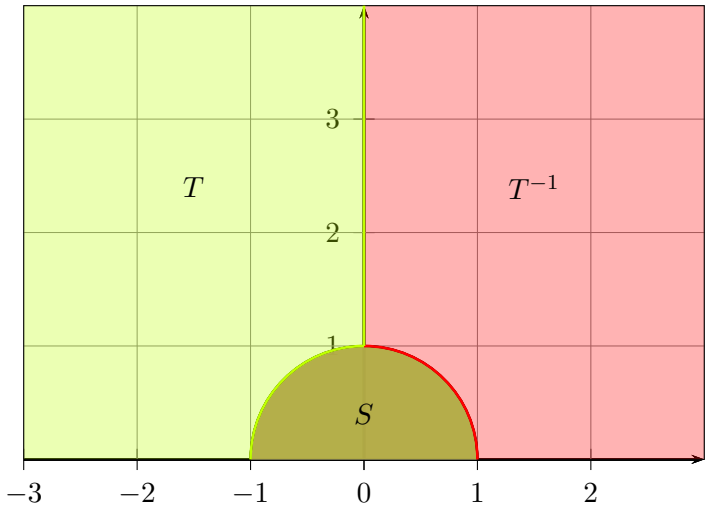
```
def S_Spiegelung(val):
    newx = - xpoint_list[-1]/(xpoint_list[-1]**2 + ypoint_list[-1]**2)
    newy = ypoint_list[-1]/(xpoint_list[-1]**2 + ypoint_list[-1]**2)
    xpoint_list.append(newx)
    ypoint_list.append(newy)
    ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="grey")
    ax.scatter(xpoint_list[-1],ypoint_list[-1])
    way_to_domain.append("S")
    point_in_domain(xpoint_list[-1],ypoint_list[-1])

def T_Translation(val):
    newx = xpoint_list[-1] + 1
    newy = ypoint_list[-1]
    xpoint_list.append(newx)
    ypoint_list.append(newy)
    ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="grey")
    ax.scatter(xpoint_list[-1],ypoint_list[-1])
    way_to_domain.append("T")
    point_in_domain(xpoint_list[-1],ypoint_list[-1])

def T_TranslationInverse(val):
    newx = xpoint_list[-1] - 1
    newy = ypoint_list[-1]
    xpoint_list.append(newx)
    ypoint_list.append(newy)
```

```
26      ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="gray")
27      ax.scatter(xpoint_list[-1],ypoint_list[-1])
28      way_to_domain.append("T^(-1)")
29      point_in_domain(xpoint_list[-1],ypoint_list[-1])
```

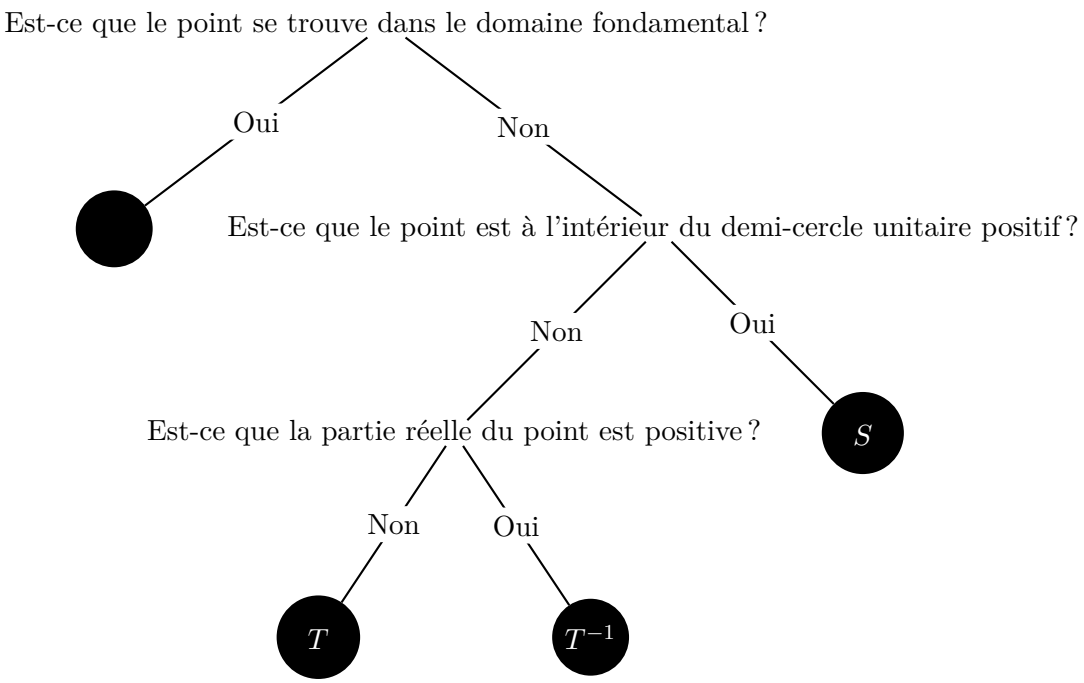
Maintenant nous programmons la partie et l'algorithme le plus complexe du programme entier. Nous définissons une fonction `way()` qui décide quelle est l'action suivante nécessaire pour arriver dans le domaine fondamental. Elle n'agit que si le dernier point dans notre liste n'est pas encore dans le domaine fondamental et s'il l'est déjà, elle ne fait rien. Nous exprimons ceci à l'aide d'une condition `if`. Nous différencions 3 situations :

- Notre dernier point se trouve à l'intérieur du demi-cercle unitaire positif. Nous appliquons alors la fonction `S_Spiegelung()`.
- Notre dernier point ne se trouve pas à l'intérieur du demi-cercle unitaire positif et sa partie réelle est positive. Nous appliquons alors la fonction `T_TranslationInverse()`.
- Notre dernier point ne se trouve pas à l'intérieur du demi-cercle unitaire positif et sa partie réelle est négative. Nous appliquons alors la fonction `T_Translation()`.

Dans la figure suivante, nous voyons quelle matrice la fonction `way()` va choisir en fonction de la partie du repère dans laquelle le point en question se trouve.



Les différentes conditions `if` peuvent être visualisés à l'aide du schéma suivant :



La fonction `way()` affiche aussi le chemin parcouru dans notre console.

```
def way(val):
    if not((-0.5 < xpoint_list[-1] < 0 and ypoint_list[-1]> sqrt(1-xpoint_list
    [-1]**2)) or (0 <= xpoint_list[-1] <= 0.5 and ypoint_list[-1]>= sqrt(1-
    xpoint_list[-1]**2))):
        if sqrt(xpoint_list[-1]**2 + ypoint_list[-1]**2) < 1:
            S_Spiegelung(1)
        else:
            if xpoint_list[-1] > 0:
                T_TranslationInverse(1)
            else:
                T_Translation(1)

        way_detail = ""
        for i in range(len(way_to_domain)):
            way_detail = way_to_domain[i] + way_detail

        print(way_detail + " = " + str(round(xpoint_list[-1],2)) + " + " + str(
    round(ypoint_list[-1],2)) + "i")
```

Nous créons à présent les deux boutons "Next" et "New Point". Le bouton "Next" fait appel à la fonction `way()` et affiche ainsi le prochain point pour arriver dans le domaine fondamental. Une fois arrivé dans le domaine fondamental, le bouton "Next" ne marche plus, comme la fonction `way()` a été implémentée de telle façon à ne plus rien faire une fois le domaine fondamental est atteint. L'utilisateur peut ensuite appuyer sur le bouton "New Point" et les anciens points vont

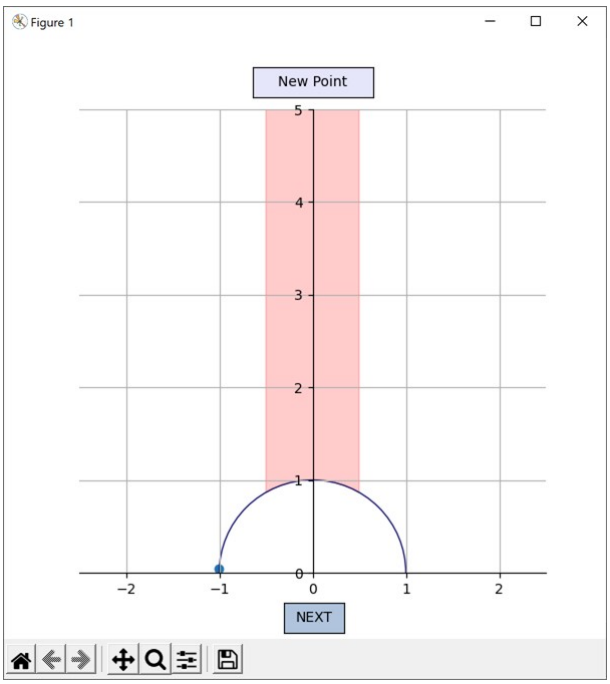
se mettre dans l'arrière-plan et un nouveau point de départ va apparaître. Il faut noter que la console affiche à chaque utilisation du bouton "Next" les matrices appliquées au point initial dans la console.

```
# Générer le bouton "Next Step"
next_axes = plt.axes([0.465, 0.01, 0.1, 0.05])
next_button = Button(next_axes, 'NEXT',color="lightsteelblue")
next_button.on_clicked(way)

# Générer le bouton "New Point"
renew_axes = plt.axes([0.413, 0.9, 0.2, 0.05])
renew_button = Button(renew_axes, 'New Point',color="lavender")
renew_button.on_clicked(new_try)
```

Finalement, nous devons encore ajouter la fonction `show()` pour rendre la fenêtre visible.

```
plt.show()
```

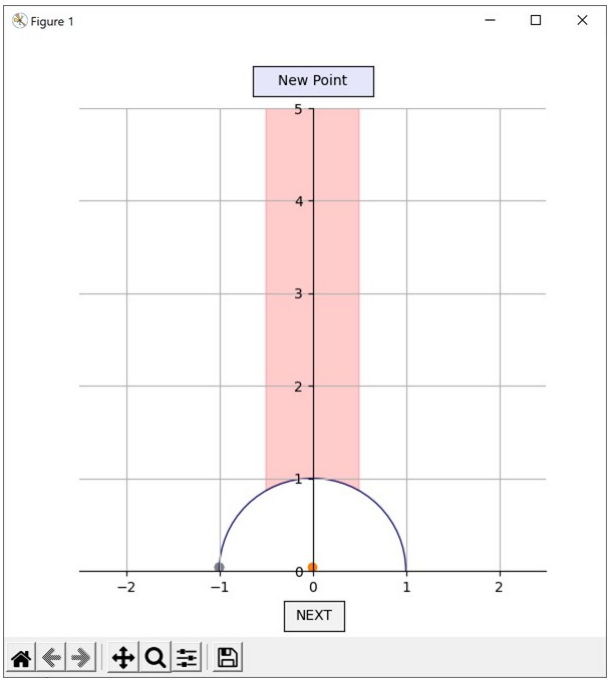
Lorsque l'utilisateur compile le programme, un point aléatoire est crée et affiché. L'utilisateur voit un affichage similaire à celui-ci :



`z = -1.0 + 0.05 i`

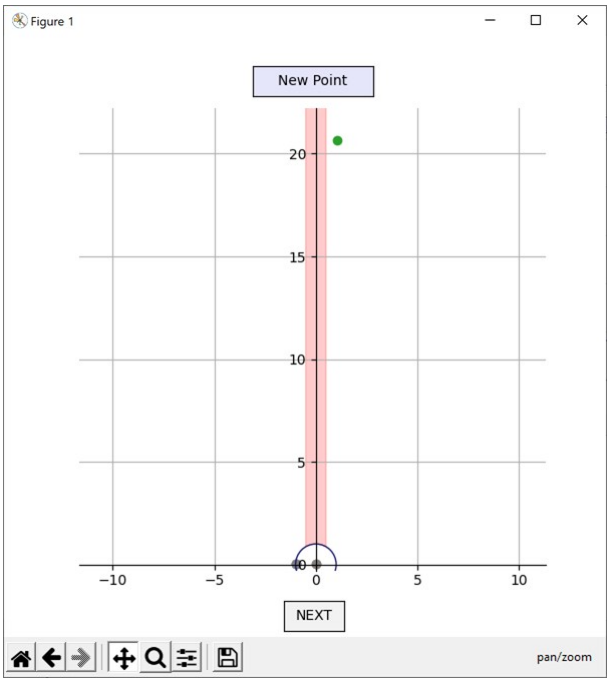
L'utilisateur a ensuite la possibilité de voir la prochaine point dans le but d'arriver dans le domaine fondamental en poussant sur le bouton "NEXT" ou il peut obtenir un nouveau point de départ en poussant sur le bouton "New Point".

Disons que l'utilisateur décide de cliquer sur le bouton "Next". Alors l'affichage suivant est obtenu :



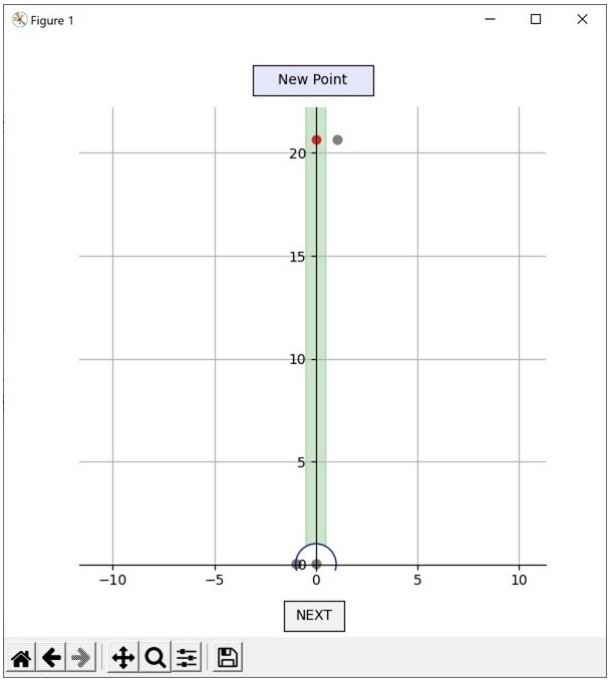

```
z = -1.0 + 0.05 i
Tz = 0.0 + 0.05i
```

La console indique qu'on a appliqué la matrice T au point de départ. Disons que l'utilisateur décide de cliquer de nouveau sur le bouton "Next". Alors l'affichage suivant est obtenu :



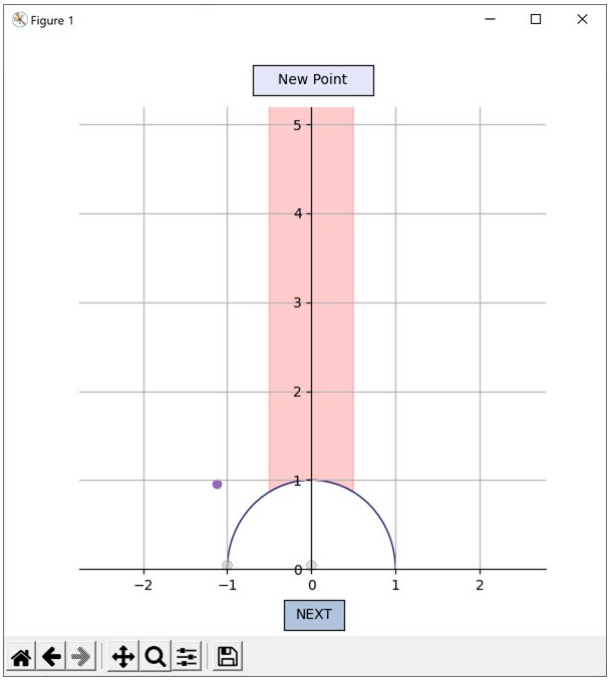
```
z = -1.0 + 0.05 i
Tz = 0.0 + 0.05i
STz = 1.03 + 20.65i
```

Le nouveau point est sorti du champ de vision initial, c'est pourquoi l'utilisateur a ici manipulé le graphique de telle façon à ce que le point est de nouveau visible.
La console indique qu'on a appliqué les matrices T et S au point de départ. La façon de l'écriture suggère aussi que la matrice T a d'abord été appliquée et ensuite la matrice S.
Comme le domaine fondamental n'a toujours pas été atteint, l'utilisateur pousse de nouveau le bouton "Next". L'affichage suivant est obtenu :




```
z = -1.0 + 0.05 i
Tz = 0.0 + 0.05i
STz = 1.03 + 20.65i
T^(-1)STz = 0.03 + 20.65i
```

Le domaine fondamental change de couleur et devient vert. La dernière ligne dans la console affiche le chemin parcouru et les matrices utilisées pour passer de z à un point dans le domaine fondamental. L'utilisateur peut alors cliquer sur le bouton "Next", mais il ne va rien se passer.
Si l'utilisateur décide à présent de cliquer sur le bouton "New Point", un nouveau point arbitraire va apparaître sur l'écran. L'utilisateur voit alors un affichage similaire au suivant :



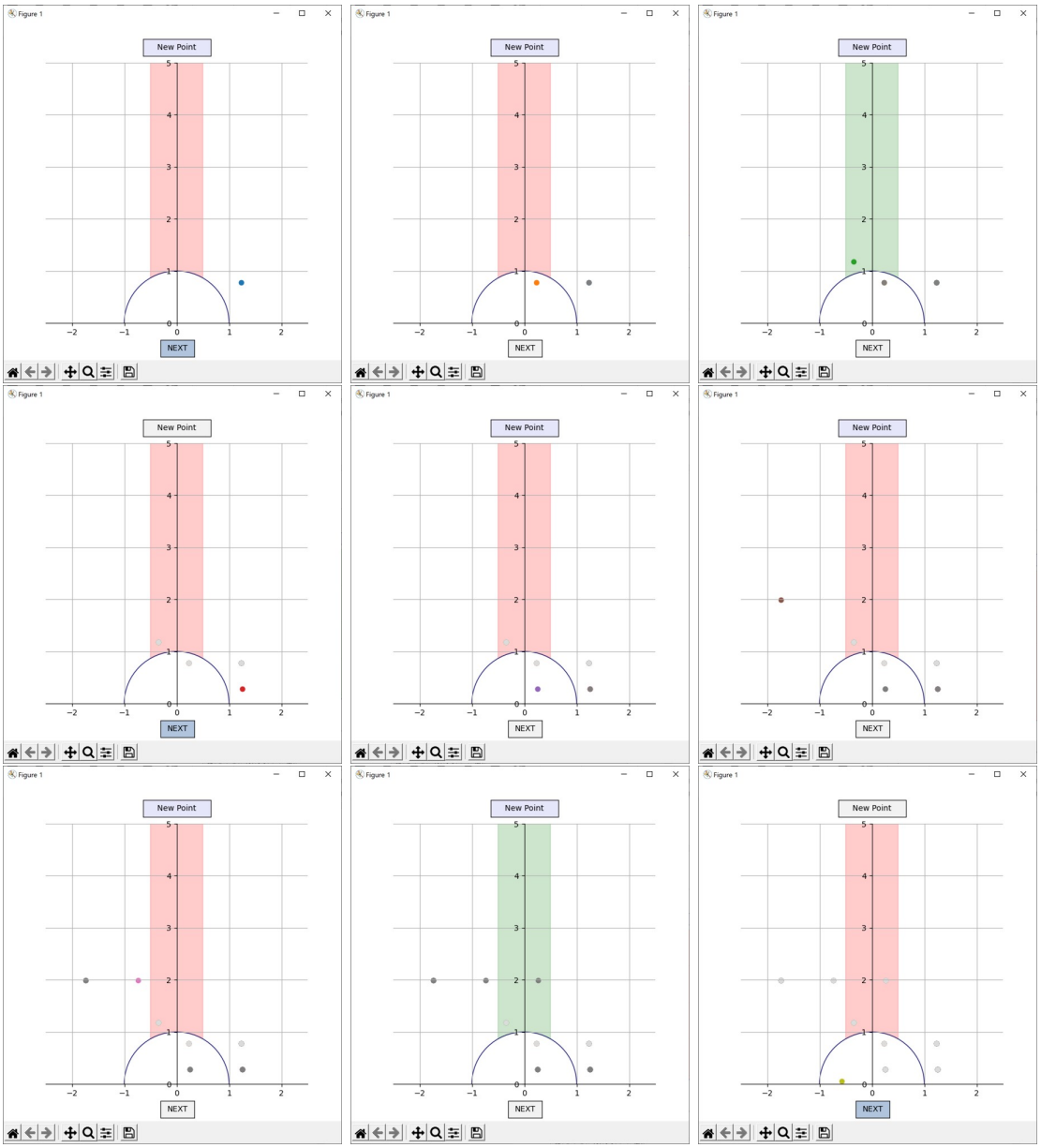

```
z = -1.0 + 0.05 i
Tz = 0.0 + 0.05i
STz = 1.03 + 20.65i
T^(-1)STz = 0.03 + 20.65i

z = -1.0 + 0.05 i
```

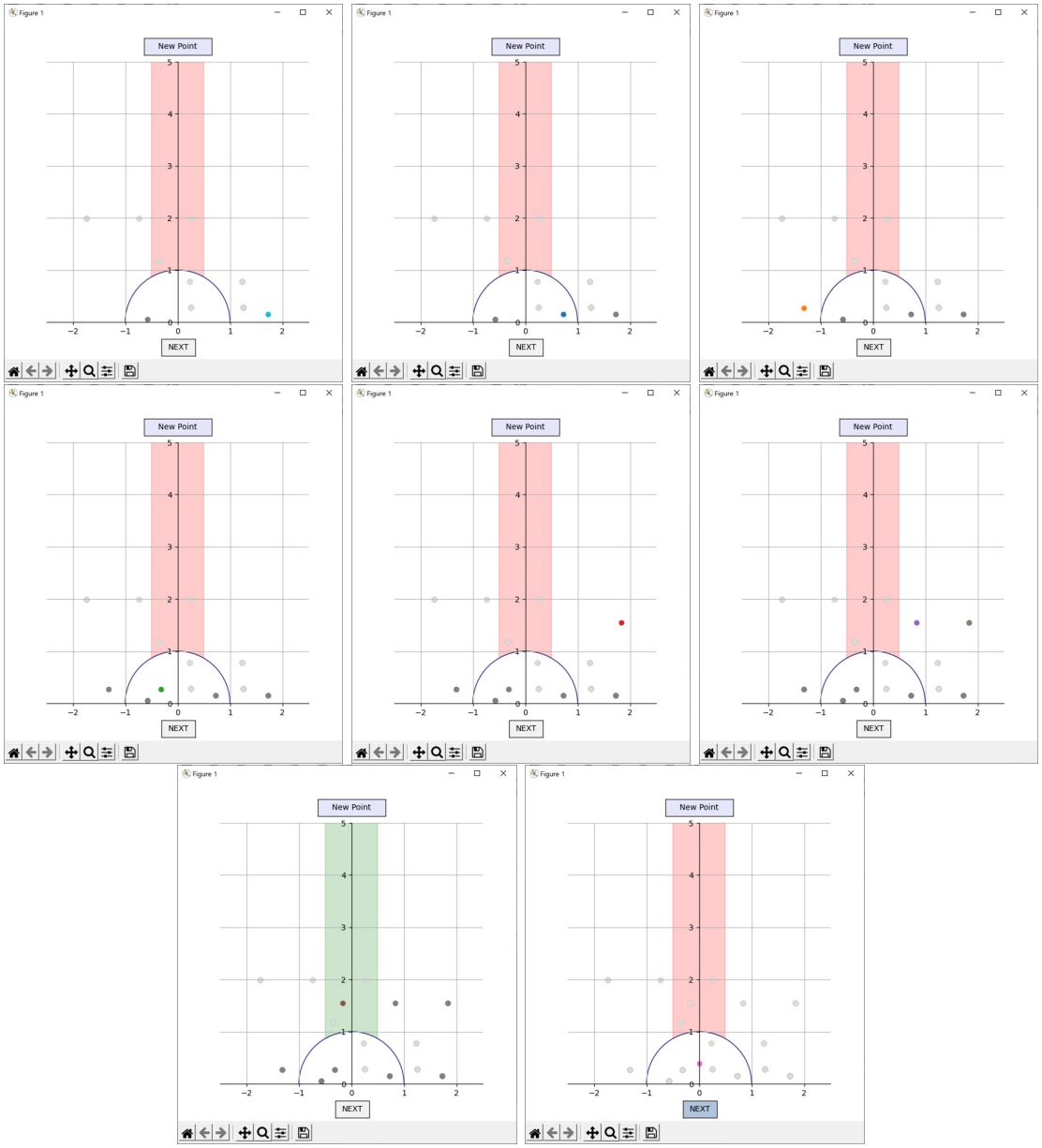
L'utilisateur peut alors de nouveau commencer le jeu et suivre le chemin du point ayant comme but d'atteindre le domaine fondamental.
Un poussant 2 fois sur "Next", les 2 affichages suivants apparaissent :

Le texte suivant a alors été ajouté à la console :

```
z = -1.0 + 0.05 i
Tz = -0.13 + 0.96i
STz = 0.13 + 1.03i
```

Voici un autre exemple de compilation du programme :






```
z = 1.23 + 0.78 i
T^(-1)z = 0.23 + 0.78i
ST^(-1)z = -0.35 + 1.17i

z = 1.23 + 0.78 i
T^(-1)z = 0.25 + 0.28i
ST^(-1)z = -1.74 + 1.99i
TST^(-1)z = -0.74 + 1.99i
TTST^(-1)z = 0.26 + 1.99i

z = 1.23 + 0.78 i
```

```
Sz = 1.73 + 0.15i
T^(-1)Sz = 0.73 + 0.15i
ST^(-1)Sz = -1.32 + 0.27i
TST^(-1)Sz = -0.32 + 0.27i
STST^(-1)Sz = 1.84 + 1.55i
T^(-1)STST^(-1)Sz = 0.84 + 1.55i
T^(-1)T^(-1)STST^(-1)Sz = -0.16 + 1.55i

z = 1.23 + 0.78 i
```

4.3 Programme d'animation avec Pygame

Ce programme est notre programme principal d'animation qui visualise d'une manière fluide le parcours d'un point qui veut atteindre le domaine fondamental. Le point peut être choisi par l'utilisateur par un simple clique sur le bouton gauche de la souris. Il va être développé dans la section 4.4 pour visualiser les fractions continues et il va être légèrement adapté dans la section 4.5 pour les surfaces triangulaires de Hecke. Avec plus de 200 lignes, il est beaucoup plus complexe que les programmes précédents, mais il reprend beaucoup d'idées que nous venons de rencontrer dans la section 4.1 et 4.2.

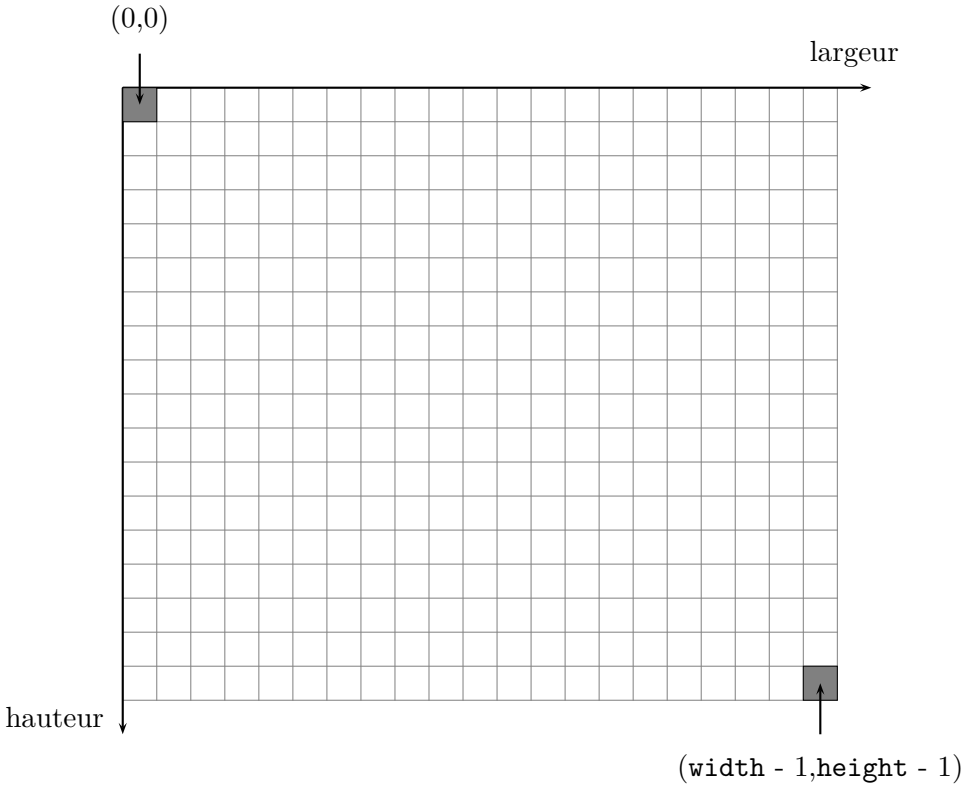
Nous devons d'abord importer toutes les librairies nécessaires. Nous importons la librairie entière de `pygame` et de `sys` qui nous permettent la visualisation et l'animation. Nous importons aussi toutes les fonctions locales de `pygame`. Nous importons quelques fonctions mathématiques de la librairie `math` et nous importons des fonctions qui nous retournent des nombres aléatoirs de la librairie `random`. La librairie `numpy` nous permet d'utiliser une fonction qui retourne une liste de valeurs régulièrement espacés dans un certain intervalle.

```
1 import pygame, sys
2 from pygame.locals import *
3 import numpy as np
4 from math import sqrt, asin, sin
5 from random import randint, random, uniform
```

Ensuite nous utilisons la commande `init()` de la librairie `pygame`, qui ouvre la surface de dessin. Il est important d'ajouter `quit()` à la fin de chaque compilation, pour que la surface puisse se fermer correctement.

Nous définissons la largeur `width` et la hauteur `height` de notre fenêtre. `width` vaut 801 et `height` vaut 700, ce qui signifie que notre fenêtre va avoir les dimensions 801x700 pixels. Par la commande `display.set_mode()`, nous créons ensuite une surface de dessin appelée `screen` de largeur `width` et de hauteur `height`.

Nous définissons également la variable `bottom`, qui définit la distance entre l'axe des abscisses et le fond de la fenêtre.
Pour visualiser la surface de dessin en terme de pixels, il faut savoir que le pixel $(0, 0)$ se trouve en haut à gauche. Nous avons `width` pixels de gauche à droite et `height` pixels du haut en bas. Le pixel tout en bas à gauche est alors (`width`-1,`height`-1).



Nous définissons aussi la variable `unity`, qui nous dit combien de pixels correspondent à une unité dans le repère que nous allons définir et dessiner. Nous avons décidé que 100 pixels correspondent à une unité.

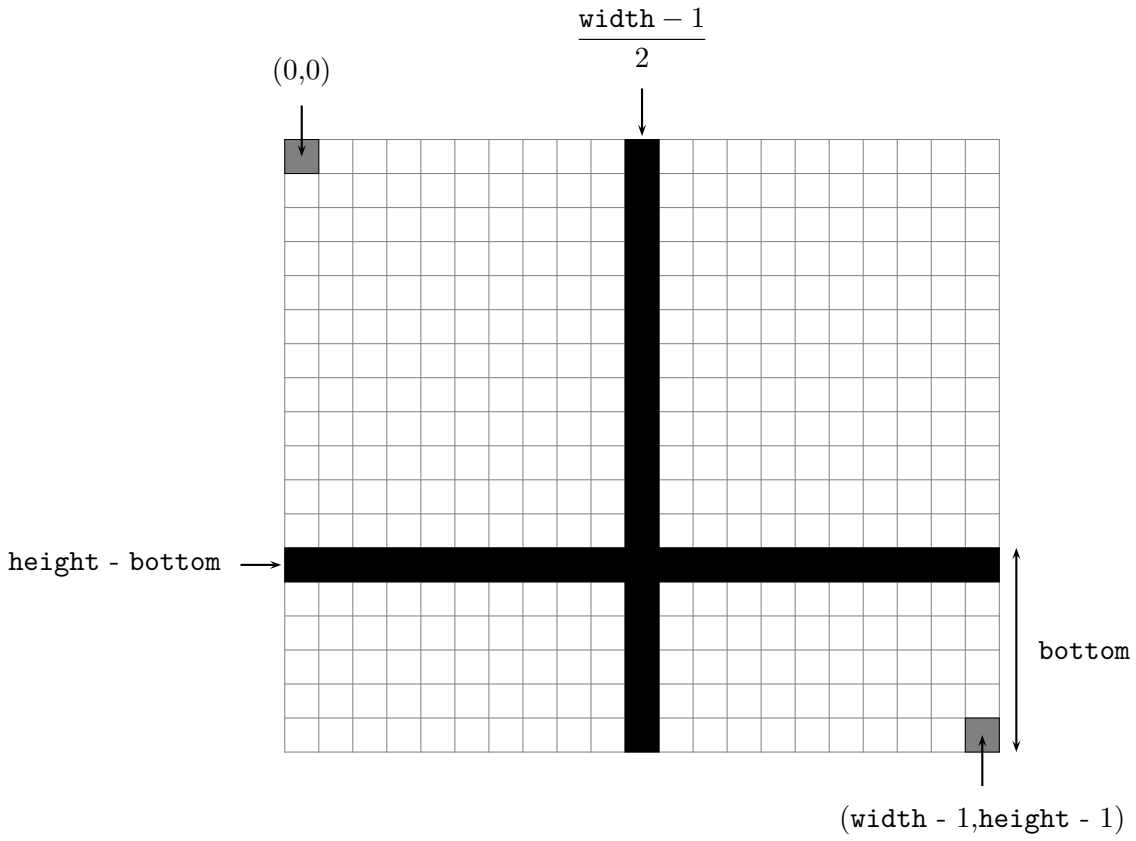
```
#Création de la surface de dessin
pygame.init()
width = 801
height = 700
bottom = 30
unity = 100
size = (width,height)
screen = pygame.display.set_mode(size)
```

Nous commençons maintenant la partie des fonctions. Les premières fonctions définies sont les actions par les matrices S, T et T^{-1}. Elles prennent comme arguments la partie réelle et la partie imaginaire du nombre complexe que l'on veut transformer par une des matrices. Elles retournent un tuple (x, y) qui représente la partie réelle et la partie imaginaire de l'application de la matrice respective au nombre complexe entré.

```
def S(x,y):
    newx = -x/(x**2 + y**2)
    newy = y/(x**2 + y**2)
    return (newx,newy)
```

```
5 def T(x,y):
6     return (x+1,y)
7
8 def Tinverse(x,y):
9     return (x-1,y)
```

Nous définissons maintenant des fonctions très importantes, qui facilitent l'écriture du programme. En effet, `pygame` travaille avec des pixels qui sont disposés d'une certaine manière. Il n'y a pas de fonction prédéfinie qui nous permet de transformer la surface en repère. C'est pourquoi nous devons le dessiner et le définir nous-mêmes. Nous voulons un dessin qui ressemble à celui-ci :



Le point $(0,0)$ correspond ainsi au tuple de pixels $\left(\frac{\texttt{width}-1}{2}, \texttt{height}-\texttt{bottom}\right)$.

Pour un point (x,y), nous savons alors que x correspond à $\frac{\texttt{width}-1}{2} + x \times \texttt{unity}$ en pixels et que y correspond à $\texttt{height} - \texttt{bottom} - y \times \texttt{unity}$ en pixels.
Similairement, nous trouvons alors que le couple de pixels (w,h) correspond au couple

$$\left(\frac{w - \frac{\texttt{width}-1}{2}}{\texttt{unity}}, -\frac{h - \texttt{height} + \texttt{bottom}}{\texttt{unity}}\right)$$

dans notre repère.

Pour nous faciliter ses calculs, nous définissons des fonctions qui nous transforment d'un côté les pixels en points dans notre repère et vice versa.

La fonction `widthpixel()` nous traduit l'abscisse d'un point dans notre repère en coordonnée de largeur en pixels.
La fonction `heightpixel()` nous traduit l'ordonnée d'un point dans notre repère en coordonnée d'hauteur en pixels.
La fonction `xaxis()` traduit la largeur d'un point en pixels en abscisse de ce point dans notre repère.
La fonction `yaxis()` traduit la hauteur d'un point en pixels en ordonnée de ce point dans notre repère.

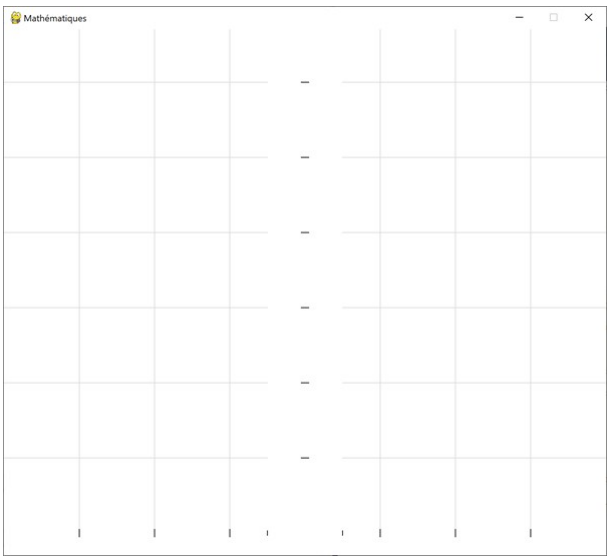
```
def widthpixel(x):
    return width//2 + x*unity

def heightpixel(y):
    return height - bottom - y*unity

def xaxis(x):
    return (x-width//2)/unity

def yaxis(y):
    return -(y-height+bottom)/unity
```

Nous définissons à présent une fonction `drawgrid()` qui prend comme arguments la largeur et la hauteur de la surface de dessin et qui dessine les lignes auxiliaires du graphique. Elle ajoute des droites horizontales et verticales pour chaque entier de notre repère.

```
def drawgrid(width,height):
    pygame.draw.line(screen,"black",(widthpixel(0.5),heightpixel(-0.03)),(widthpixel(0.5),heightpixel(0.03)))
    pygame.draw.line(screen,"black",(widthpixel(-0.5),heightpixel(-0.03)),(widthpixel(-0.5),heightpixel(0.03)))

    for m in range(1,(width//100)//2):
        pygame.draw.line(screen,"gainsboro",(widthpixel(m),height-bottom ),(widthpixel(m),0))
        pygame.draw.line(screen,"gainsboro",(widthpixel(-m),height-bottom ),(widthpixel(-m),0))
        pygame.draw.line(screen,"black",(widthpixel(m),heightpixel(-0.05)),(widthpixel(m),heightpixel(0.05)))
        pygame.draw.line(screen,"black",(widthpixel(-m),heightpixel(-0.05)),(widthpixel(-m),heightpixel(0.05)))

    for n in range(1,height//100):
        pygame.draw.line(screen,"gainsboro",(0,heightpixel(n)),(width//2 - unity*0.5,heightpixel(n)))
```

```
13          pygame.draw.line(screen,"gainsboro",(width//2 + unity*0.5,heightpixel(n)
    ),(width,heightpixel(n)))
14          pygame.draw.line(screen,"black",(widthpixel(-0.05),heightpixel(n)),(
    widthpixel(0.05),heightpixel(n)))
```

En appelant cette fonction incorporée dans un programme qui se termine, nous obtenons la représentation suivante :



Nous définissons à présent trois fonctions qui nous augmentent la lisibilité du code du coeur de cette animation. La première fonction `distance` nous retourne la distance entre deux points dont les coordonnées sont entrées en tant qu'attributs. La fonction `combinedlistprint()` prend deux listes comme arguments. Si les deux listes sont de même taille, pour tout i elle imprime le nombre complexe dont la partie réelle est le $i^{\text{ième}}$ élément de la première liste et dont la partie imaginaire est le $i^{\text{ième}}$ élément de la deuxième liste. La fonction `listtostring()` affiche le chemin total que le point a parcouru pour arriver d'un point initial à un point final. Les matrices utilisées doivent être entrées comme attributs sous forme d'une liste, ainsi que les coordonnées du point initial et du point final.

```
1  def distance(x1,y1,x2,y2):
2      return sqrt((x1-x2)**2 + (y1-y2)**2)
3
4  def combinedlistprint(list1,list2):
5      combinedlist = []
6      if len(list1) == len(list2):
7          for i in range(len(list1)):
8              print(str(round(list1[i],3))+" + " + str(round(list2[i],2)) + "i")
9
10 def listtosting(way_to_domain,oldx,oldy,newx,newy):
11     if oldx != newx or oldy != newy:
```

```
        way = "( "+str(round(oldx,3))+" + "+str(round(oldy,3))+"i )"
        for i in range(len(way_to_domain)):
            way = str(way_to_domain[i]) + way
        print(way + " = " + str(round(newx,3)) + " + " +str(round(newy,3))+ "i")
```

La fonction suivante va être utilisée à plusieurs reprises dans notre boucle principale. En effet, la fonction `indomain()` prend les coordonnées d'un point comme attributs et retourne le booléen `True` si le point se trouve dans le domaine fondamental et retourne le booléen `False` si cela n'est pas le cas.

```
def indomain(x,y):
    return ((-0.5 < x <0 and x**2 + y**2 > 1) or (0 <= x <= 0.5 and x**2 + y**2
    >= 1))
```

La fonction `updatepoints()` va être utilisée pour créer l'animation en soi. Elle va retourner les points qui se trouvent sur la droite entre un point de départ et un point d'arrivée. Pour cela, elle calcule d'abord la pente de cette droite et ensuite son ordonnée à l'origine. À l'aide de l'équation de la droite, les points entre le point de départ et le point d'arrivée peuvent facilement être trouvés et retrouvés.

```
def update_points(oldx,oldy,newx,newy,i):
    if oldx != newx :
        m = (oldy - newy)/(oldx - newx)
        b = newy - m*newx
        step = distance(oldx,oldy,newx,newy)/100
        if oldx < newx :
            movingx = oldx + step*i
            movingy = m*movingx + b
            return (widthpixel(movingx),heightpixel(movingy))
        else:
            movingx = oldx - step*i
            movingy = m*movingx + b
            return (widthpixel(movingx),heightpixel(movingy))
```

Nous devons ensuite ajuster encore certains détails de notre surface de dessin. Par la fonction `fill()` nous définissons la couleur de l'arrière-plan de la surface de dessin et par la fonction `display.set_caption()` nous donnons un titre à la surface. Nous définissons aussi une nouvelle variable `FPS`, qui dénote la fréquence d'image. Par la fonction `time.Clock()` nous initions une sorte d'horloge dans notre programme. En effet, avoir 60 en tant que fréquence d'image nous dit que pendant une seconde, notre boucle principale va être parcourue 60 fois.

Nous créons aussi deux listes vides `xlist` et `ylist` et nous définissons la valeur initiale de nos curseurs `i` et `j`.

```
#Titre de la fenêtre
pygame.display.set_caption("Mathématiques")

screen.fill("white")

#Fréquence d'image
FPS = 60
clock = pygame.time.Clock()

xlist = []
ylist = []

i = 0
j = 1
```

Nous pouvons désormais procéder à l'implémentation de la boucle principale de notre programme. Cette boucle va être parcourue pour chaque image tant que la variable `done` reste `False`. Si `done` devient `True`, alors `not done` devient `False` et la boucle ne va plus être parcourue. Nous définissons que `done` devient `True` si l'utilisateur clique sur le bouton rouge de fermeture de fenêtre.

De plus, si l'utilisateur enfonce la touche gauche de la souris et clique sur un point dans la surface de dessin, nous ajoutons les coordonnées en pixels de ce point sous forme de coordonnées d'un point dans notre repère dans nos listes. Nous colorons la surface de dessin encore une fois en blanc.

Nous définissons à présent l'algorithme principal du programme, qui est très similaire à l'algorithme utilisé dans le programme précédent. Des schémas pour la compréhension et la visualisation de l'algorithme peuvent être retrouvés à la page 34. En résumé, tant que notre dernier point dans la liste n'est pas dans le domaine fondamental, l'algorithme doit trouver une action pour amener le point dans ce domaine. Si le dernier point dans la liste se situe à l'intérieur du demi-cercle unitaire positif, la matrice S est appliquée. Si le dernier point dans la liste ne se trouve pas à l'intérieur du demi-cercle unitaire positif, la matrice T^{-1} est appliquée lorsque sa partie réelle est positive et la matrice T est appliquée lorsque sa partie réelle est négative. À chaque fois qu'une matrice est appliquée, elle est conservée dans la liste `way_to_domain`.

Ensuite nous affichons tous les points de notre chemin ainsi que le détail des matrices dans la console à l'aide des fonctions `combinedlistprint()` et `listtostring()`.

```
#Boucle principale
done = False
while not done:
    for event in pygame.event.get():
        if event.type == QUIT:
            done = True
```

```
7
8           elif event.type == MOUSEBUTTONDOWN:
9               if event.button == 1:
10                  x_mouse = pygame.mouse.get_pos()[0]
11                  y_mouse = pygame.mouse.get_pos()[1]
12
13                  if yaxis(y_mouse)>0:
14                      x = xaxis(x_mouse)
15                      y = yaxis(y_mouse)
16                      xlist.clear()
17                      ylist.clear()
18                      xlist.append(x)
19                      ylist.append(y)
20                      way_to_domain = []
21                      screen.fill("white")
22
23                      while not indomain(xlist[-1],ylist[-1]):
24                          currentx = xlist[-1]
25                          currenty = ylist[-1]
26
27                          if sqrt(currentx**2 + currenty**2) < 1:
28                              xlist.append(S(currentx,currenty)[0])
29                              ylist.append(S(currentx,currenty)[1])
30                              pygame.draw.circle(screen,"black",(widthpixel(xlist
    [-1]),heightpixel(ylist[-1])),5,0)
31                              way_to_domain.append("S")
32
33                          else:
34                              if currentx > 0:
35                                  xlist.append(Tinverse(currentx,currenty)[0])
36                                  ylist.append(Tinverse(currentx,currenty)[1])
37                                  pygame.draw.circle(screen,"black",(widthpixel(
    xlist[-1]),heightpixel(ylist[-1])),5,0)
38                                  way_to_domain.append("T^(-1)")
39                              else:
40                                  xlist.append(T(currentx,currenty)[0])
41                                  ylist.append(T(currentx,currenty)[1])
42                                  pygame.draw.circle(screen,"black",(widthpixel(
    xlist[-1]),heightpixel(ylist[-1])),5,0)
43                                  way_to_domain.append("T")
44
45                      combinedlistprint(xlist,ylist)
46                      listtosting(way_to_domain,xlist[0],ylist[0],xlist[-1],ylist
    [-1])
47                      print()
```

À présent nous avons la liste entière des complexes qui constituent le chemin vers le domaine fondamental. La prochaine partie de notre programme va s'occuper entièrement de l'animation.

En effet, à chaque fois que la boucle principale est parcourue, nous dessinons un nouveau point sur la surface. Comme cela se passe tellement vite, l'oeil de l'utilisateur ne va pas pouvoir discerner les images individuelles, mais il va voir une animation fluide. Nous dessinons non seulement les points consécutifs par la fonction `draw.circle()`, mais nous dessinons aussi les droites qui lient ces points par la fonction `draw.line()`.

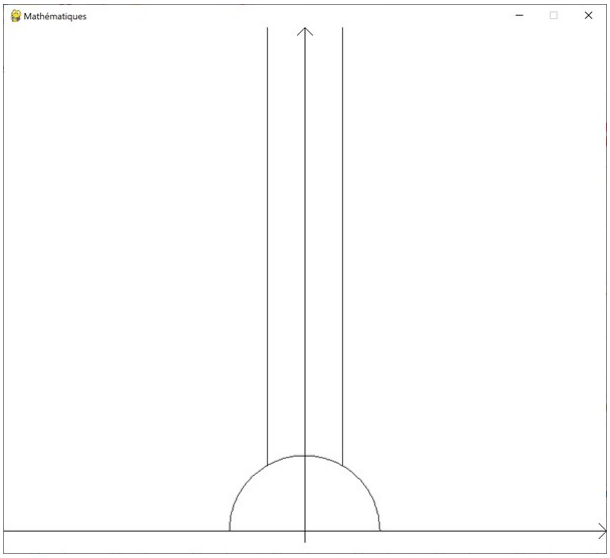
```
    if len(xlist) > 1 and j < len(xlist):
        oldx = xlist[j-1]
        oldy = ylist[j-1]
        newx = xlist[j]
        newy = ylist[j]

        xmove = update_points(oldx,oldy,newx,newy,i)[0]
        ymove = update_points(oldx,oldy,newx,newy,i)[1]

        if indomain(xaxis(xmove),yaxis(ymove)) and j == (len(xlist)-1):
            screen.fill("white")
            pygame.draw.rect(screen,"green",(width//2 - unity*0.5,0,unity,height
))
            pygame.draw.circle(screen,"white",(width//2,height-bottom),unity)
            drawgrid(width,height)

        else:
            screen.fill("white")
            pygame.draw.rect(screen,"red",(width//2 - unity*0.5,0,unity,height))
            pygame.draw.circle(screen,"white",(width//2,height-bottom),unity)
            drawgrid(width,height)

        if oldx < newx :
            if xmove <= width//2 + newx*unity:
                pygame.draw.circle(screen,"black",(xmove,ymove),5,0)
                pygame.draw.line(screen, "black", (widthpixel(oldx),heightpixel(
oldy)),(xmove,ymove))
                i = i + 1
            else:
                i = 0
                j = j+1

        else:
            if width//2 + newx*unity <= xmove :
                pygame.draw.circle(screen,"black",(xmove,ymove),5,0)
                pygame.draw.line(screen, "black", (widthpixel(oldx),heightpixel(
oldy)),(xmove,ymove))
                i = i + 1
            else:
                i = 0
                j = j+1

        for k in range(j):
```

```
            pygame.draw.circle(screen,"black",(widthpixel(xlist[k]),heightpixel(
    ylist[k])),5,0)

        if j > 1:
            for k in range(j-1):
                pygame.draw.line(screen, "black", (widthpixel(xlist[k]),
    heightpixel(ylist[k])),(widthpixel(xlist[k+1]),heightpixel(ylist[k+1])))

    elif len(xlist) == 1:
        screen.fill("white")
        drawgrid(width,height)
        pygame.draw.rect(screen,"green",(width//2 - unity*0.5,0,unity,height))
        pygame.draw.circle(screen,"white",(width//2,height-bottom),unity)
        pygame.draw.circle(screen,"black",(widthpixel(xlist[-1]),heightpixel(
    ylist[-1])),5,0)

    elif j >= len(xlist) :
        i = 0
        j = 1
        xlist.clear()
        ylist.clear()
```

Dans la boucle principale, nous dessinons aussi les axes principales ainsi que le demi-cercle unitaire positif. La fonction `display.update()` fait une mise à jour de tous les nouveaux éléments ajoutés au graphique et les rend visible. La fonction `tick()` insert des pauses pour respecter les fréquences voulues.

```
    #Dessiner le graphique
    pygame.draw.line(screen,"black",(width//2,0),(width//2,height-bottom+15))
    pygame.draw.line(screen,"black",(0,height-bottom),(width,height-bottom))
    pygame.draw.line(screen,"black",(width//2,0),(width//2+10,10))
    pygame.draw.line(screen,"black",(width//2,0),(width//2-10,10))
    pygame.draw.line(screen,"black",(width,height-bottom),(width-10,height-
    bottom+10))
    pygame.draw.line(screen,"black",(width,height-bottom),(width-10,height-
    bottom-10))
    pygame.draw.arc(screen,"black",(width//2-unity,height-bottom-unity,unity*2,
    unity*2),0,3.1416)
    pygame.draw.line(screen,"black",(width//2+unity*0.5,height-bottom-unity
    *0.866),(width//2+unity*0.5,0))
    pygame.draw.line(screen,"black",(width//2-unity*0.5,height-bottom-unity
    *0.866),(width//2-unity*0.5,0))

    pygame.display.update()
    clock.tick(FPS)
```

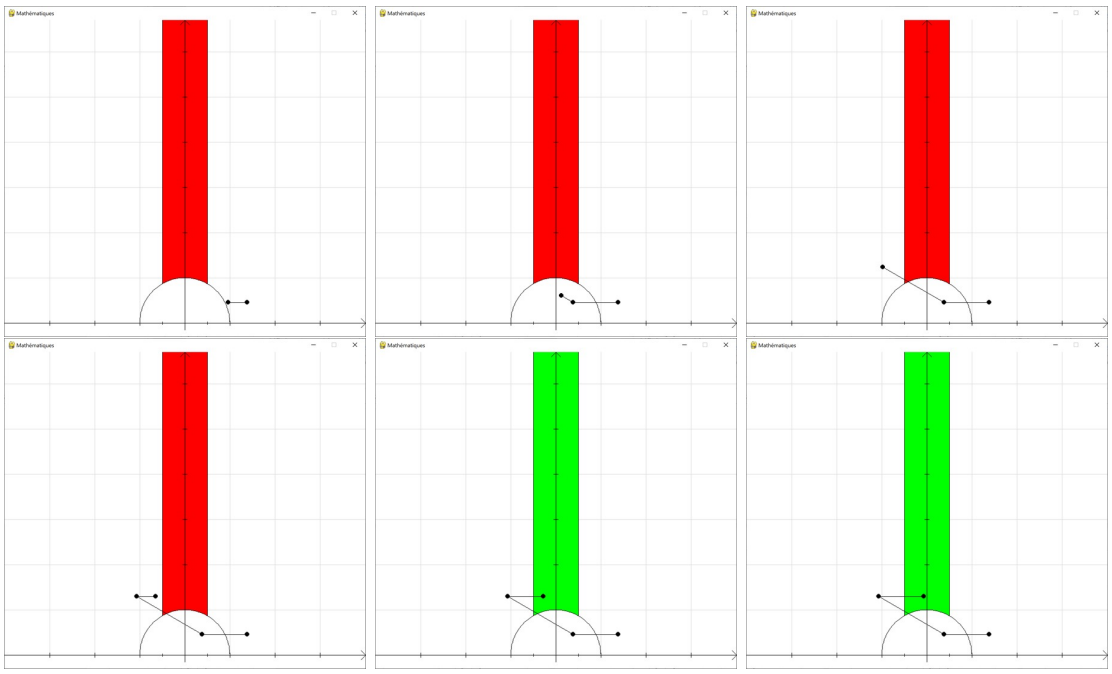
Cette dernière partie nous donne l'image suivante :



Pour que notre programme puisse bien se terminer, nous devons encore ajouter les fonction `quit()` et `exit()`.

```
#Fermer la fenêtre et quitter le programme
pygame.quit()
sys.exit()
```

Voici un exemple de ce programme. Bien sûr, il s'agit d'une animation fluide et ce ne sont donc que des photographies instantanées qui ne peuvent pas réflecter le charme entier du programme.



```
1.38 + 0.46i
0.38 + 0.46i
-1.067 + 1.29i
-0.067 + 1.29i
TST^(-1)( 1.38 + 0.46i ) = -0.067 + 1.292i
```

4.4 Fractions continues

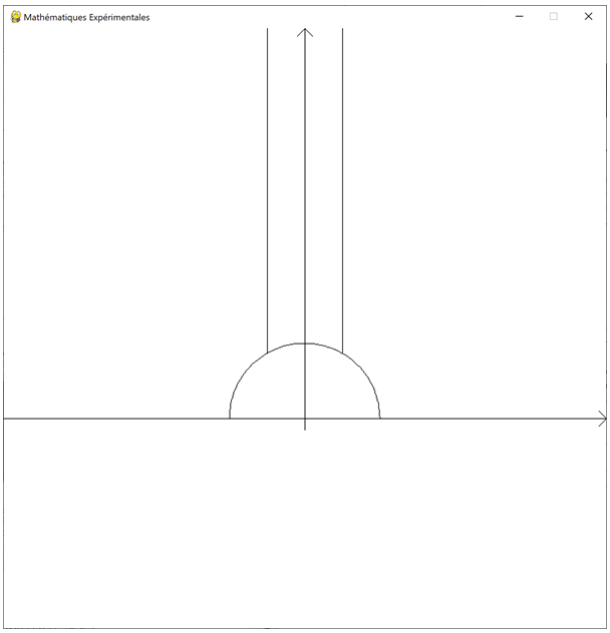
Ce programme reprend le programme de la section 4.3 et y ajoute l'affichage des fractions continues. Ces fractions continues résultent des calculs en appliquant les matrices S, T et T^{-1} à un nombre complexe. Visualiser ces fractions continues n'est pas une tâche facile. Avec plus de 500 lignes de code, il s'agit d'un programme lourd et complexe. L'utilisateur peut ainsi lier les fractions complexes avec le chemin que le point parcourt.

Nous n'ajoutons donc que quelques lignes de programme au programme d'animation de la section 4.3 et nous obtenons un programme similaire à ce dernier, mais qui est encore mieux.

D'abord, nous devons initialiser l'environnement qui nous permet d'afficher du texte dans notre fenêtre. Nous agrandissons aussi notre fenêtre, qui a désormais une largeur de 801 pixels et une hauteur de 800 pixels.

```
pygame.font.init()
width = 801
height = 550
Height = 800
size = (width,Height)
screen = pygame.display.set_mode(size)
myfont = pygame.font.SysFont('freesansbold.ttf', 30)
```

Notre fenêtre avec le graphique ressemble alors à ceci :



Nous définissons aussi une nouvelle fonction `prettylist()` qui retourne une version plus ordonnée de la liste qui contient les matrices du chemin. Au lieu de par exemple avoir 10 fois

consécutives la matrice T dans la liste, nous n'avons qu'un seul élément T^{10} dans notre nouvelle liste.

```
def prettylist(list1):
    list2 = []
    i = 0
    while i < len(list1) and len(list1) != 0 :
        a = 1
        j = i + 1

        while j < len(list1):

            if list1[i] == list1[j]:
                a = a + 1
                j = j + 1
            else:
                j = len(list1)

        if a > 1:
            if list1[i] == "T^(-1)":
                list2.append("T^(-"+str(a)+")")
            else:
                list2.append(list1[i]+"^"+str(a))
        else:
            list2.append(list1[i])

        i = i + a

    return list2
```

Dans la dernière rangée de la compilation ci dessous, nous pouvons observer l'aspect positif de cette nouvelle fonction.

```
0.48 + 0.02i
-2.08 + 0.09i
-1.08 + 0.09i
-0.08 + 0.09i
5.75 + 6.25i
4.75 + 6.25i
3.75 + 6.25i
2.75 + 6.25i
1.75 + 6.25i
0.75 + 6.25i
-0.25 + 6.25i
T^(-1) T^(-1) T^(-1) T^(-1) T^(-1) T^(-1) S T T S ( 0.48 + 0.02i ) = -0.25 + 6.25i
T^(-6) S T^2 S ( 0.48 + 0.02i ) = -0.25 + 6.25i
```

Nous définissons désormais la fonction `fraclist()` qui va retourner une liste contenant des tuples de la forme $(0, a)$ et $(1, b)$ où a désigne l'exposant de S et b l'exposant de T. Il s'agit d'une autre manière d'écrire notre liste qui contient les matrices S, T et T^{-1}.

```
def fraclist(list1):
    list2 = []
    i = 0
    while i < len(list1) and len(list1) != 0 :
        a = 1
        j = i + 1

        while j < len(list1):

            if list1[i] == list1[j]:
                a = a + 1
                j = j + 1
            else:
                j = len(list1)

        if list1[i] == "T^(-1)":
            #T^(-1) correspond à 1 avec un exposant négatif
            list2.append((1,-int(a)))
        elif list1[i] == "T":
            #T correspond à 1 avec un exposant positif
            list2.append((1,int(a)))
        elif list1[i] == "S":
            #S correspond à 0
            list2.append((0,int(a)))

        i = i + a

    return list2
```

La prochaine fonction nous permet de savoir le signe d'un nombre. Elle va être utilisée lors de la visualisation des fractions continues.

```
def signof(number):
    if number < 0:
        return " - "
    else:
        return ""
```

Les fonctions `frac1S()`,`frac2S()`, `frac3S()` et `frac4S()` visualisent les fractions continues de profondeur 1, 2, 3 et 4 respectivement. Nous savons que si S apparaît une fois dans le chemin, alors nous avons un unique trait de fraction dans notre résultat, si S apparaît deux fois dans le chemin, alors nous avons deux traits de fraction dans notre résultat et ainsi de suite. À l'aide de cette logique nous définissons les fonctions qui vont nous permettre de visualiser les fractions

continues résultantes du chemin dans notre surface de dessin.

```
def frac1S(list1,k,l):

    flist = fraclist(list1)

    if flist[0][0]==0:

        if len(flist) == 1:
            calculation1 = myfont.render("z", False, (0, 0, 0))
            screen.blit(calculation1,(width//2 + 90 + k, height + (Height - height)//4 + 20 + l - 20))

            minus1 = myfont.render(" - 1", False, (0, 0, 0))
            screen.blit(minus1,(width//2 + 70 + k, height + (Height - height)//4 - 15 + l - 20))

            pygame.draw.line(screen, "black", (width//2 + 35 + k, height + (Height - height)//4 + 12 + l- 20),(width//2 + 145 + k, height + (Height - height)//4 + 12 - 20 + l))

        else:
            calculation1 = myfont.render("z", False, (0, 0, 0))
            screen.blit(calculation1,(width//2 + 120 + k, height + (Height - height)//4 + 20 + l - 20))

            minus1 = myfont.render(" - 1", False, (0, 0, 0))
            screen.blit(minus1,(width//2 + 100 + k, height + (Height - height)//4 - 15 + l- 20))

            pygame.draw.line(screen, "black", (width//2 + 70 + k, height + (Height - height)//4 + 12 + l- 20),(width//2 + 180 + k, height + (Height - height)//4 + 12 + l- 20))

            if len(flist)>=2:

                if signof(flist[1][1]) == " - ":
                    calculation2 = myfont.render(signof(flist[1][1])+str(abs(flist[1][1]))+" + ", False, (0, 0, 0))
                    screen.blit(calculation2,(width//2 + 20 + k , height + (Height - height)//4 + l- 20))
                else:
                    calculation2 = myfont.render(signof(flist[1][1])+str(abs(flist[1][1]))+" + ", False, (0, 0, 0))
                    screen.blit(calculation2,(width//2 + 25 + k, height + (Height - height)//4 + l- 20))

    else:

        minus1 = myfont.render(" - 1", False, (0, 0, 0))
```

```
37          screen.blit(minus1,(width//2 + 100 + k, height + (Height - height)//4 -
    15 + l- 20))
38
39          pygame.draw.line(screen, "black", (width//2 + 70 + k, height + (Height -
     height)//4 + 12 + l- 20),(width//2 + 180 + k, height + (Height - height)//4
     + 12 + l- 20))
40
41          if signof(flist[0][1]) == " - ":
42              calculation1 = myfont.render(signof(flist[0][1])+str(abs(flist
    [0][1]))+" + z", False, (0, 0, 0))
43              screen.blit(calculation1,(width//2 + 85 + k, height + (Height -
    height)//4 + 20 + l- 20))
44          else:
45              calculation1 = myfont.render(signof(flist[0][1])+str(abs(flist
    [0][1]))+" + z", False, (0, 0, 0))
46              screen.blit(calculation1,(width//2 + 100 + k, height + (Height -
    height)//4 + 20 + l- 20))
47
48          if len(flist)>=3:
49
50              if signof(flist[2][1]) == " - ":
51                  calculation2 = myfont.render(signof(flist[2][1])+str(abs(flist
    [2][1]))+" + ", False, (0, 0, 0))
52                  screen.blit(calculation2,(width//2 + 20 + k, height + (Height -
    height)//4 + l- 20))
53              else:
54                  calculation2 = myfont.render(signof(flist[2][1])+str(abs(flist
    [2][1]))+" + ", False, (0, 0, 0))
55                  screen.blit(calculation2,(width//2 + 35 + k , height + (Height -
     height)//4 + l- 20))
```

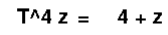
Le détail explicite des fonctions `frac2S()`, `frac3S()` et `frac4S()` peut être retrouvé dans l'annexe D.

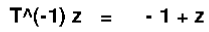
La fonction `drawfrac()` rassemble toutes les visualisations de fractions continues en appelant les justes fonctions au bon moment. Dans notre boucle principale, nous n'appelons donc que cette fonction-ci et non pas les fonctions individuelles `frac1S()`, `frac2S()`, `frac3S()` et `frac4S()`.

```
1  def drawfrac(list1):
2
3      flist = fraclist(list1)
4
5      if len(flist) > 0:
6
7          ftext = "z"
8          d = 0
9
10         for i in range(len(prettylist(list1))):
11             ftext = str(prettylist(list1)[i]) + " " + ftext
```

```
12
13            if flist[i][0]==0:
14                d = d + 20
15            else:
16                if flist[i][1] < 0:
17                    d = d + 70
18                elif flist[i][1] == 1:
19                    d = d + 20
20                else:
21                    d = d + 40
22
23
24        textsurface = myfont.render(ftext, False, (0, 0, 0))
25        screen.blit(textsurface,(width//2 - d - 30 ,height + (Height - height)
    //4- 20))
26
27        equalsign = myfont.render("=", False, (0, 0, 0))
28        screen.blit(equalsign,(width//2- 10, height + (Height - height)//4- 20))
29
30
31        if list1.count("S")==0:
32            calculation = myfont.render(signof(flist[0][1])+str(abs(flist[0][1])
    )+" + z", False, (0, 0, 0))
33            screen.blit(calculation,(width//2 + 30, height + (Height - height)
    //4- 20))
34
35        elif list1.count("S")==1:
36
37            frac1S(list1,0,0)
38
39        elif list1.count("S")==2:
40
41            frac2S(list1,0,0)
42
43        elif list1.count("S")==3:
44
45            frac3S(list1,0,0)
46
47        elif list1.count("S")==4:
48
49            frac4S(list1,0,0)
```

Nous avons deux formes de résultats qui ne contiennent aucun S, qui sont affichés sous cette forme :

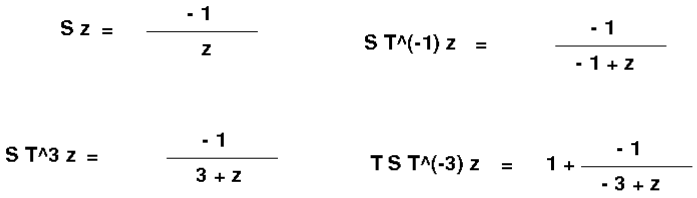




Pour ces illustration, la fonction `drawfrac()` dessine directement sur la fenêtre et n'appelle pas de fonction auxiliaire.

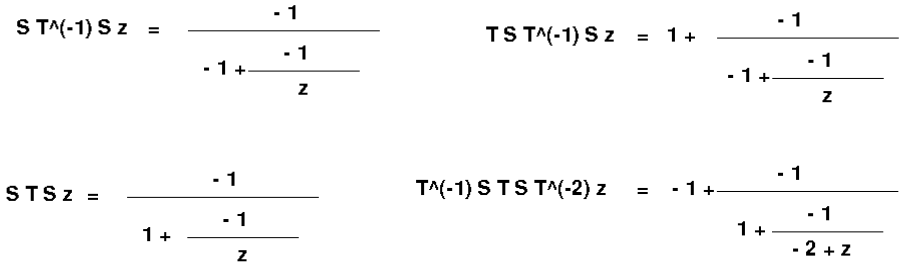
Pour les résultats qui contiennent au moins un S, la fonction `drawfrac()` fait appel aux fonctions `frac1S()`, `frac2S()`, `frac3S()` et `frac4S()`.

Nous avons quatre formes de résultats qui contiennent exactement un S :



Pour ces illustrations, la fonction `frac1S()` est appelée.

Nous avons quatre formes de résultats qui contiennent exactement deux S :



Pour ces illustrations, la fonction `frac2S()` est appelée.

Dans la boucle principale, nous définissons `oldz` et `newz` qui sont des chaînes de caractères qui désignent le point de départ et le point d'arrivée. Si le point de départ est le même que le point d'arrivée, c'est-à-dire si le point de départ se trouve déjà dans le domaine fondamental, alors le point d'arrivée est attribué à "000". De cette manière, nous pouvons facilement différencier le cas où le point de départ se trouve déjà dans le domaine.

```
                    listtostring(way_to_domain,xlist[0],ylist[0],xlist[-1],ylist
[-1])
                    listtostring(prettylist(way_to_domain),xlist[0],ylist[0],
xlist[-1],ylist[-1])

                    oldz = str(round(xlist[0],3))+" + "+str(round(ylist[0],3))+"
i"
                    if len(xlist)!=1:
                        newz = str(round(xlist[-1],3))+" + "+str(round(ylist
[-1],3))+"i"
                    else:
                        newz = "000"
```

Finalement, nous appelons la fonction `drawfrac()` dans la dernière partie de la boucle principale. Nous distinguons le cas où le point initial se trouve déjà dans le domaine fondamental et le cas où le point initial ne se trouve pas encore dans le domaine fondamental.

```
    if oldz != "":
        if newz == "000":
            samez = myfont.render(" z = "+str(oldz)+" se trouve déjà dans D",
False, (0, 0, 0))
            screen.blit(samez,(width//2 - 170 ,height + (Height - height)//4))

        else:
            drawfrac(way_to_domain)

            samez = myfont.render("où z = "+str(oldz)+" est transformé en "+str(
newz), False, (0, 0, 0))
            screen.blit(samez,(width//2 - 200 ,Height - 30))
```

Si notre point se trouve déjà dans le domaine fondamental, nous allons obtenir le message suivant sans aucun calcul additionnel :

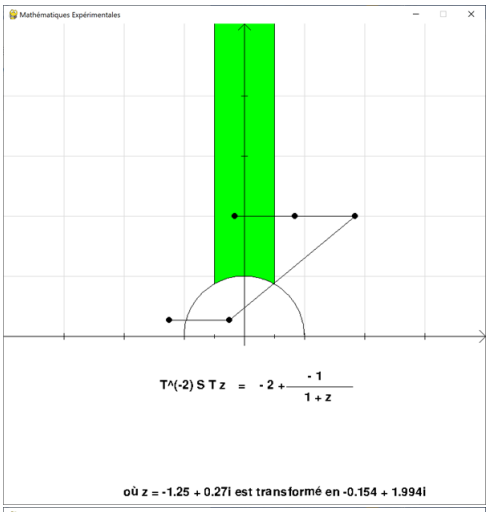
z = 0.1 + 1.6i se trouve déjà dans D

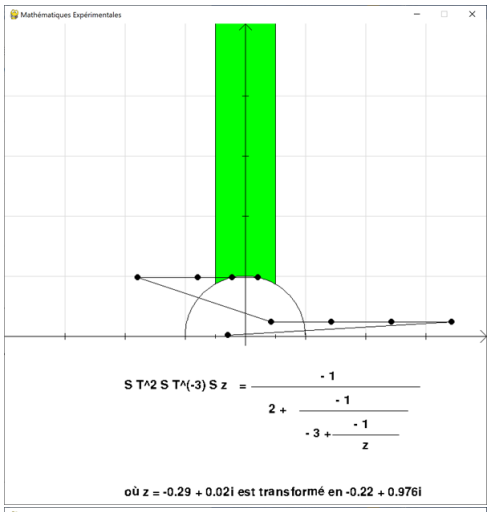
Sinon, notre calcul va être accompagné d'un petit message :

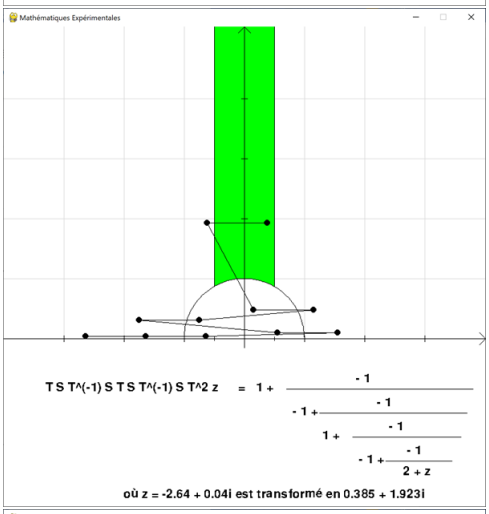
T^2 z = 2 + z

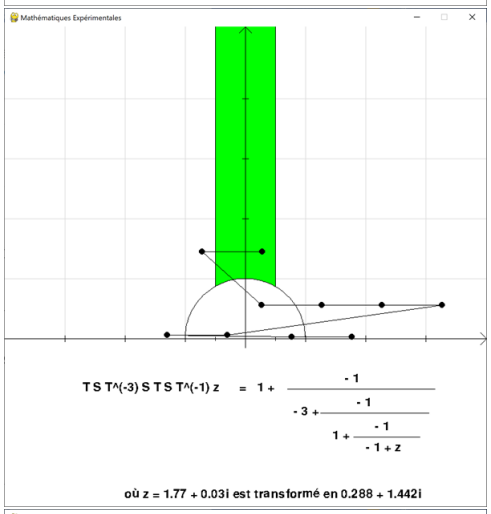
où z = -1.94 + 2.38i est transformé en 0.06 + 2.38i

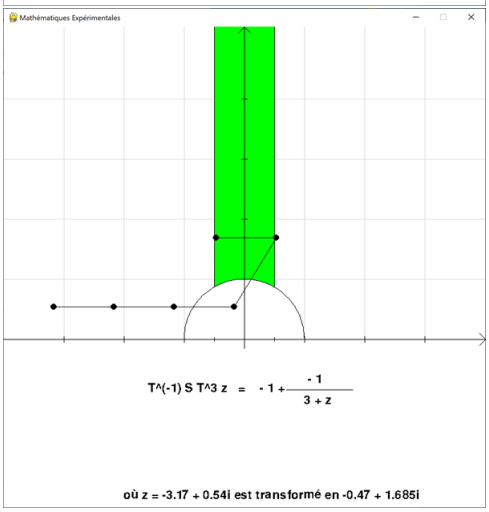
Voici quelques exemples d'application du programme :

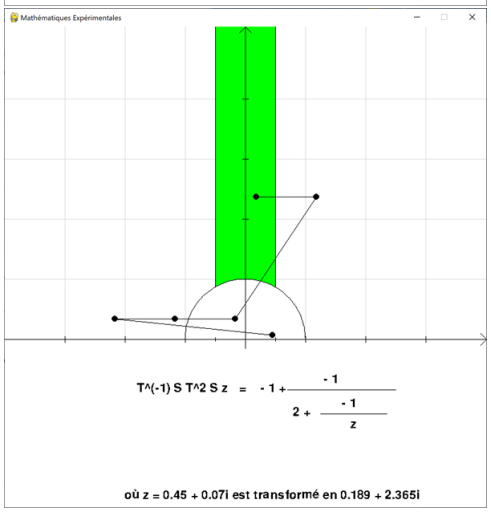












4.5 Programme d'animation avec Pygame sur les surfaces de Hecke

Ce programme est très similaire au programme de la section 4.3, mais au lieu de la matrice T il travaille avec la matrice

$$T_q := \begin{pmatrix} 1 & \lambda_q \\ 0 & 1 \end{pmatrix} \text{ où } \lambda_q := 2\cos\left(\frac{\pi}{q}\right).$$

La matrice S reste inchangée.
Nous avons alors que pour un $z = x + yi \in \mathfrak{H}$,

$$\begin{aligned} T_q z &= \begin{pmatrix} 1 & \lambda_q \\ 0 & 1 \end{pmatrix} z \\ &= \frac{1 \times z + \lambda_q}{0 \times z + 1} \\ &= z + \lambda_q \\ &= (x + \lambda_q) + yi \end{aligned}$$

Nous utilisons donc le programme de la section 4.3 comme base et ne changeons et ajoutons que quelques parties précises.

Nous ajoutons tout d'abord une commande `input` qui permet à l'utilisateur d'entrer un entier positif qui sera attribué à q et avec lequel notre programme continuera de calculer. Nous définissons ensuite notre λ_q que nous allons appeler a au long du programme.

```
q = int(input("Veuillez entrer un entier positif q: "))
if q != 0:
    a = 2*cos(np.pi/abs(q))
```

Dans la console, l'utilisateur peut alors entrer l'entier positif de son choix pour q. Le message suivant s'affiche :

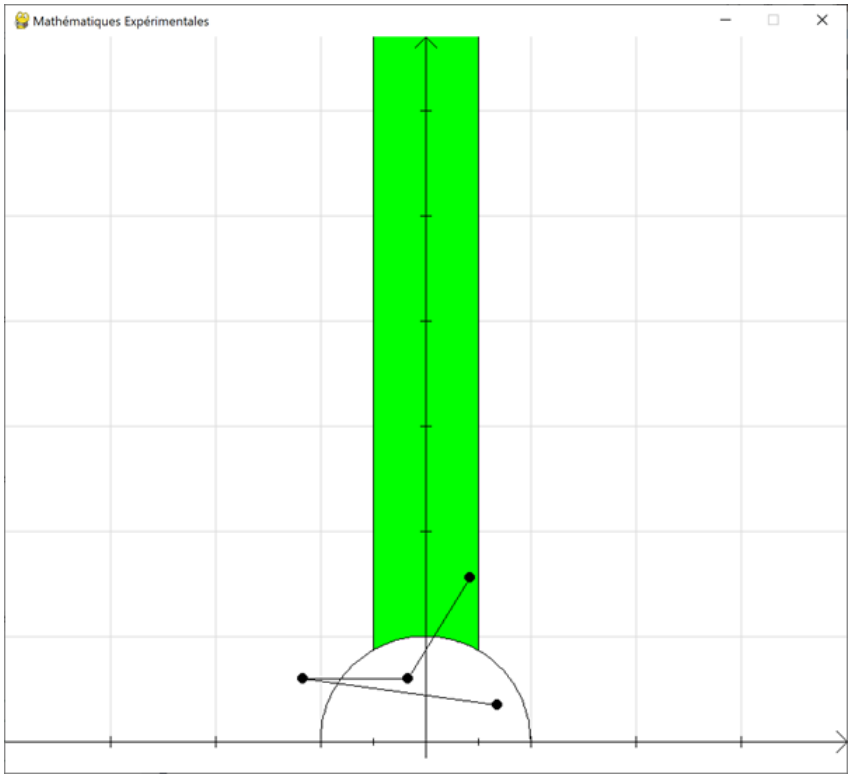
```
Veuillez entrer un entier positif q:
```

Parmi tout le programme, nous devons adapter le dessin en fonction de q. Une fois cela est fait, nous devons aussi altérer la fonction `indomain()`, car le domaine lui-aussi change en fonction de q.

```
    def indomain(x,y):
        return ((-a/2 < x <0 and x**2 + y**2 > 1) or (0 <= x <= a/2 and x**2 + y
    **2 >= 1))
```

Une fois que tous les petits détails du dessin ont été changés, le programme est prêt pour l'utilisateur.

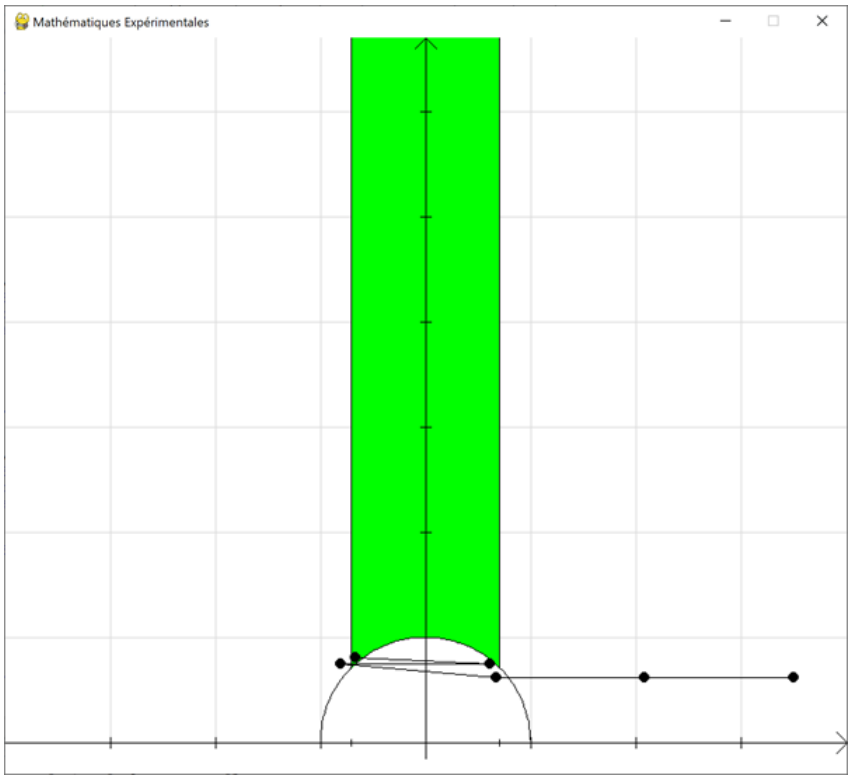
Disons que l'utilisateur entre 3 en tant que q. Un affichage similaire se produit :



Dans la console, nous pouvons relire les détails :

```
Veuillez entrer un entier positif q: 3
0.68 + 0.35i
-1.163 + 0.6i
-0.163 + 0.6i
0.423 + 1.56i
STqS(0.68 + 0.35i ) = 0.423 + 1.556i
```

Si l'utilisateur entre 4, il peut observer l'affichage suivant :



Dans la console, nous pouvons relire les détails :

```
Veuillez entrer un entier positif q: 4
3.5 + 0.62i
2.086 + 0.62i
0.672 + 0.62i
-0.804 + 0.74i
0.61 + 0.74i
-0.661 + 0.8i
STqSTq^(-1)Tq^(-1)(3.5 + 0.62i) = -0.661 + 0.804i
```

Conclusion

Nous avons réussis à mettre en oeuvre et à visualiser l'interaction entre les fractions continues et la géométrie. En effet, nous avons exploré brièvement les surfaces triangulaires de Hecke ainsi que les fractions continues. Nous avons réussi a démontrer que tout point dans le demi-plan de Poincaré peut être ramené dans le domaine fondamental par une combinaison des deux matrices S et T. Nous avons également trouvé l'algorithme qui nous donne cette combinaison.

Au fur et à mesure les programmes ont été améliorés pour devenir de plus en plus autonome. Le premier programme demande les choix des actions tandis que le dernier programme nous donne directement la solution finale tout en détaillant les fractions continues.

Enfin, en réalisant ce projet, nous nous sommes initié à deux nouvelles librairies qui nous ont permis d'animer et de visualiser nos algorithmes : `Pygame` et `Matplotlib`.

Nous tenons à remercier nos superviseurs Gabor Wiese et Lassina Dembélé de nous avoir guidée tout au long de ce projet.

Annexe

A Programme d'introduction avec Matplotlib

```
import numpy as np
from math import sqrt
import matplotlib.pyplot as plt
from pylab import *
from random import randint,random,uniform

#Créer une fenêtre
fig = plt.figure(figsize=(6, 6))

#Créer des axes
ax = fig.add_subplot(1,1,1)
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
plt.grid()
plt.ylim([0,5])
plt.xlim([-2.5,2.5])

# Créer et afficher un titre pour le graphique
plt.title('Mathématiques Expérimentales', fontsize=8)

# Créer un cercle et l'ajouter au graphique
circle = plt.Circle((0,0),1,fill=False, color="midnightblue")
ax.add_artist(circle)

# Colorier le domaine fondamental
x = np.arange(-0.5,0.5,0.0001)
domaine = ax.fill_between(x,sqrt(1-x**2),10**10 , alpha=0.2, color="red")

# Générer un point aléatoire de départ
xpoint = uniform(-2,2)
ypoint = random()

# Créer deux listes de stockage
xpoint_list = [xpoint]
ypoint_list = [ypoint]

# Dessiner les points sur le graphique
ax.scatter(xpoint_list,ypoint_list)

# Afficher le point généré dans la console
print(str(round(xpoint,2)) + " + " + str(round(ypoint,2)) + "i")

def S_Spiegelung(val):
    newx = - xpoint_list[-1]/(xpoint_list[-1]**2 + ypoint_list[-1]**2)
```

```
47      newy = ypoint_list[-1]/(xpoint_list[-1]**2 + ypoint_list[-1]**2)
48      xpoint_list.append(newx)
49      ypoint_list.append(newy)
50      ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="gray")
51      ax.scatter(xpoint_list[-1],ypoint_list[-1])
52      if (-0.5 < xpoint_list[-1] < 0 and ypoint_list[-1]> sqrt(1-xpoint_list
        [-1]**2)) or (0 <= xpoint_list[-1] <= 0.5 and ypoint_list[-1]>= sqrt(1-
        xpoint_list[-1]**2)):
53          domaine.set_color("green")
54      else:
55          domaine.set_color("red")
56      print("S")
57      print(str(round(xpoint_list[-1],2)) + " + " + str(round(ypoint_list[-1],2))
        + "i")
58
59  def T_Translation(val):
60      newx = xpoint_list[-1] + 1
61      newy = ypoint_list[-1]
62      xpoint_list.append(newx)
63      ypoint_list.append(newy)
64      ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="gray")
65      ax.scatter(xpoint_list[-1],ypoint_list[-1])
66      if (-0.5 < xpoint_list[-1] < 0 and ypoint_list[-1]> sqrt(1-xpoint_list
        [-1]**2)) or (0 <= xpoint_list[-1] <= 0.5 and ypoint_list[-1]>= sqrt(1-
        xpoint_list[-1]**2)):
67          domaine.set_color("green")
68      else:
69          domaine.set_color("red")
70      print("T")
71      print(str(round(xpoint_list[-1],2)) + " + " + str(round(ypoint_list[-1],2))
        + "i")
72
73  def T_TranslationInverse(val):
74      newx = xpoint_list[-1] - 1
75      newy = ypoint_list[-1]
76      xpoint_list.append(newx)
77      ypoint_list.append(newy)
78      ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="gray")
79      ax.scatter(xpoint_list[-1],ypoint_list[-1])
80      if (-0.5 < xpoint_list[-1] < 0 and ypoint_list[-1]> sqrt(1-xpoint_list
        [-1]**2)) or (0 <= xpoint_list[-1] <= 0.5 and ypoint_list[-1]>= sqrt(1-
        xpoint_list[-1]**2)):
81          domaine.set_color("green")
82      else:
83          domaine.set_color("red")
84      print("T(-1)")
85      print(str(round(xpoint_list[-1],2)) + " + " + str(round(ypoint_list[-1],2))
        + "i")
86
87  # Générer le boutton S
88  s_axes = plt.axes([0.3, 0.01, 0.1, 0.05])
```

```
89 s_button = Button(s_axes, 'S',color="lightsteelblue")
90 s_button.on_clicked(S_Spiegelung)
91
92 # Générer le boutton T
93 t_axes = plt.axes([0.6, 0.01, 0.1, 0.05])
94 t_button = Button(t_axes, 'T',color="lightsteelblue")
95 t_button.on_clicked(T_Translation)
96
97 # Générer le boutton T^(-1)
98 tinv_axes = plt.axes([0.45, 0.01, 0.1, 0.05])
99 tinv_button = Button(tinv_axes, 'T^(-1)',color="lightsteelblue")
100 tinv_button.on_clicked(T_TranslationInverse)
101
102 plt.show()
```

B Programme pour la compréhension de l'algorithme avec Matplotlib

```
import numpy as np
from math import sqrt
import matplotlib.pyplot as plt
from pylab import *
from random import randint,random

# Créer une fenêtre
fig = plt.figure(figsize=(6, 6))

# Créer des axes
ax = fig.add_subplot(1,1,1)

ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
plt.grid()
plt.ylim([0,5])
plt.xlim([-2.5,2.5])

# Créer un cercle et l'ajouter au graphique
circle = plt.Circle((0,0),1,fill=False, color="midnightblue")
ax.add_artist(circle)

# Colorier le domaine fondamental
x = np.arange(-0.5,0.5,0.0001)
domaine = ax.fill_between(x,sqrt(1-x**2),10**10 , alpha=0.2, color="green")

way_to_domain = ["z"]
way_detail = "z"

def point_in_domain(x,y):
    if (-0.5 < x < 0 and y > sqrt(1-x**2)) or (0 <= x <= 0.5 and y >= sqrt(1-x
    **2)):
        domaine.set_color("green")
        return True
    else:
        domaine.set_color("red")
        return False

xpoint = uniform(-2,2)
ypoint = random()
xpoint_list = [xpoint]
ypoint_list = [ypoint]
ax.scatter(xpoint_list,ypoint_list)
point_in_domain(xpoint_list[-1],ypoint_list[-1])
print("z = " +str(round(xpoint,2)) + " + " + str(round(ypoint,2)) + " i ")
```

```
47
48 def new_try(val):
49     ax.scatter(xpoint_list,ypoint_list,color="gainsboro")
50     global way_detail
51     xpoint_list.clear()
52     ypoint_list.clear()
53     xpoint_list.append(uniform(-2,2))
54     ypoint_list.append(random())
55     ax.scatter(xpoint_list,ypoint_list)
56     way_to_domain.clear()
57     way_to_domain.append("z")
58     way_detail = "z"
59     point_in_domain(xpoint_list[-1],ypoint_list[-1])
60     print()
61     print("z = " +str(round(xpoint,2)) + " + " + str(round(ypoint,2)) + " i ")
62
63 def S_Spiegelung(val):
64     newx = - xpoint_list[-1]/(xpoint_list[-1]**2 + ypoint_list[-1]**2)
65     newy = ypoint_list[-1]/(xpoint_list[-1]**2 + ypoint_list[-1]**2)
66     xpoint_list.append(newx)
67     ypoint_list.append(newy)
68     ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="grey")
69     ax.scatter(xpoint_list[-1],ypoint_list[-1])
70     way_to_domain.append("S")
71     point_in_domain(xpoint_list[-1],ypoint_list[-1])
72
73 def T_Translation(val):
74     newx = xpoint_list[-1] + 1
75     newy = ypoint_list[-1]
76     xpoint_list.append(newx)
77     ypoint_list.append(newy)
78     ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="grey")
79     ax.scatter(xpoint_list[-1],ypoint_list[-1])
80     way_to_domain.append("T")
81     point_in_domain(xpoint_list[-1],ypoint_list[-1])
82
83 def T_TranslationInverse(val):
84     newx = xpoint_list[-1] - 1
85     newy = ypoint_list[-1]
86     xpoint_list.append(newx)
87     ypoint_list.append(newy)
88     ax.scatter(xpoint_list[:-1],ypoint_list[:-1],color="gray")
89     ax.scatter(xpoint_list[-1],ypoint_list[-1])
90     way_to_domain.append("T^(-1)")
91     point_in_domain(xpoint_list[-1],ypoint_list[-1])
92
93 def way(val):
94     if not((-0.5 < xpoint_list[-1] < 0 and ypoint_list[-1]> sqrt(1-xpoint_list
    [-1]**2)) or (0 <= xpoint_list[-1] <= 0.5 and ypoint_list[-1]>= sqrt(1-
    xpoint_list[-1]**2))):
95         if sqrt(xpoint_list[-1]**2 + ypoint_list[-1]**2) < 1:
```

```
            S_Spiegelung(1)
        else:
            if xpoint_list[-1] > 0:
                T_TranslationInverse(1)
            else:
                T_Translation(1)

        way_detail = ""
        for i in range(len(way_to_domain)):
            way_detail = way_to_domain[i] + way_detail

        print(way_detail + " = " + str(round(xpoint_list[-1],2)) + " + " + str(
    round(ypoint_list[-1],2)) + "i")

# Générer le bouton "Next Step"
next_axes = plt.axes([0.465, 0.01, 0.1, 0.05])
next_button = Button(next_axes, 'NEXT',color="lightsteelblue")
next_button.on_clicked(way)

# Générer le bouton "New Point"
renew_axes = plt.axes([0.413, 0.9, 0.2, 0.05])
renew_button = Button(renew_axes, 'New Point',color="lavender")
renew_button.on_clicked(new_try)

plt.show()
```

C Programme d'animation avec Pygame

```
import pygame, sys
from pygame.locals import *
import numpy as np
from math import sqrt, asin, sin
from random import randint, random, uniform

#Création de la surface de dessin
pygame.init()
width = 801
height = 700
bottom = 30
unity = 100
size = (width,height)
screen = pygame.display.set_mode(size)

def S(x,y):
    newx = -x/(x**2 + y**2)
    newy = y/(x**2 + y**2)
    return (newx,newy)

def T(x,y):
    return (x+1,y)

def Tinverse(x,y):
    return (x-1,y)

def drawgrid(width,height):
    pygame.draw.line(screen,"black",(widthpixel(0.5),heightpixel(-0.03)),(widthpixel(0.5),heightpixel(0.03)))
    pygame.draw.line(screen,"black",(widthpixel(-0.5),heightpixel(-0.03)),(widthpixel(-0.5),heightpixel(0.03)))

    for m in range(1,(width//100)//2):
        pygame.draw.line(screen,"gainsboro",(widthpixel(m),height-bottom ),(widthpixel(m),0))
        pygame.draw.line(screen,"gainsboro",(widthpixel(-m),height-bottom ),(widthpixel(-m),0))
        pygame.draw.line(screen,"black",(widthpixel(m),heightpixel(-0.05)),(widthpixel(m),heightpixel(0.05)))
        pygame.draw.line(screen,"black",(widthpixel(-m),heightpixel(-0.05)),(widthpixel(-m),heightpixel(0.05)))

    for n in range(1,height//100):
        pygame.draw.line(screen,"gainsboro",(0,heightpixel(n)),(width//2 - unity*0.5,heightpixel(n)))
        pygame.draw.line(screen,"gainsboro",(width//2 + unity*0.5,heightpixel(n)),(width,heightpixel(n)))
        pygame.draw.line(screen,"black",(widthpixel(-0.05),heightpixel(n)),(widthpixel(0.05),heightpixel(n)))
```

```
41
42 def widthpixel(x):
43     return width//2 + x*unity
44
45 def heightpixel(y):
46     return height - bottom - y*unity
47
48 def xaxis(x):
49     return (x-width//2)/unity
50
51 def yaxis(y):
52     return -(y-height+bottom)/unity
53
54 def distance(x1,y1,x2,y2):
55     return sqrt((x1-x2)**2 + (y1-y2)**2)
56
57 def combinedlistprint(list1,list2):
58     combinedlist = []
59     if len(list1) == len(list2):
60         for i in range(len(list1)):
61             print(str(round(list1[i],3))+" + " + str(round(list2[i],2)) + "i")
62
63 def listtosting(way_to_domain,oldx,oldy,newx,newy):
64     if oldx != newx or oldy != newy:
65         way = "( "+str(round(oldx,3))+" + "+str(round(oldy,3))+"i)"
66         for i in range(len(way_to_domain)):
67             way = str(way_to_domain[i]) + way
68         print(way + " = " + str(round(newx,3)) + " + " +str(round(newy,3))+ "i")
69
70 def indomain(x,y):
71     return ((-0.5 < x <0 and x**2 + y**2 > 1) or (0 <= x <= 0.5 and x**2 + y**2
    >= 1))
72
73 def update_points(oldx,oldy,newx,newy,i):
74     if oldx != newx :
75         m = (oldy - newy)/(oldx - newx)
76         b = newy - m*newx
77         step = distance(oldx,oldy,newx,newy)/100
78         if oldx < newx :
79             movingx = oldx + step*i
80             movingy = m*movingx + b
81             return (widthpixel(movingx),heightpixel(movingy))
82         else:
83             movingx = oldx - step*i
84             movingy = m*movingx + b
85             return (widthpixel(movingx),heightpixel(movingy))
86
87 #Titre de la fenêtre
88 pygame.display.set_caption("Mathématiques Expérimentales")
89
90 screen.fill("white")
```

```
91
92 #Fréquence d'image
93 FPS = 60
94 clock = pygame.time.Clock()
95
96 xlist = []
97 ylist = []
98
99 i = 0
100 j = 1
101
102 #Boucle principale
103 done = False
104 while not done:
105     for event in pygame.event.get():
106         if event.type == QUIT:
107             done = True
108
109         elif event.type == MOUSEBUTTONDOWN:
110             if event.button == 1:
111                 x_mouse = pygame.mouse.get_pos()[0]
112                 y_mouse = pygame.mouse.get_pos()[1]
113
114                 if yaxis(y_mouse)>0:
115                     x = xaxis(x_mouse)
116                     y = yaxis(y_mouse)
117                     xlist.clear()
118                     ylist.clear()
119                     xlist.append(x)
120                     ylist.append(y)
121                     way_to_domain = []
122                     screen.fill("white")
123
124                     while not indomain(xlist[-1],ylist[-1]):
125                         currentx = xlist[-1]
126                         currenty = ylist[-1]
127
128                         if sqrt(currentx**2 + currenty**2) < 1:
129                             xlist.append(S(currentx,currenty)[0])
130                             ylist.append(S(currentx,currenty)[1])
131                             pygame.draw.circle(screen,"black",(widthpixel(xlist
    [-1]),heightpixel(ylist[-1])),5,0)
132                             way_to_domain.append("S")
133
134                         else:
135                             if currentx > 0:
136                                 xlist.append(Tinverse(currentx,currenty)[0])
137                                 ylist.append(Tinverse(currentx,currenty)[1])
138                                 pygame.draw.circle(screen,"black",(widthpixel(
    xlist[-1]),heightpixel(ylist[-1])),5,0)
139                                 way_to_domain.append("T^(-1)")
```

```
140                             else:
141                                 xlist.append(T(currentx,currenty)[0])
142                                 ylist.append(T(currentx,currenty)[1])
143                                 pygame.draw.circle(screen,"black",(widthpixel(
    xlist[-1]),heightpixel(ylist[-1])),5,0)
144                                 way_to_domain.append("T")
145 
146                     combinedlistprint(xlist,ylist)
147                     listtosting(way_to_domain,xlist[0],ylist[0],xlist[-1],ylist
    [-1])
148                     print()
149 
150     if len(xlist) > 1 and j < len(xlist):
151         oldx = xlist[j-1]
152         oldy = ylist[j-1]
153         newx = xlist[j]
154         newy = ylist[j]
155 
156         xmove = update_points(oldx,oldy,newx,newy,i)[0]
157         ymove = update_points(oldx,oldy,newx,newy,i)[1]
158 
159         if indomain(xaxis(xmove),yaxis(ymove)) and j == (len(xlist)-1):
160             screen.fill("white")
161             pygame.draw.rect(screen,"green",(width//2 - unity*0.5,0,unity,height
    ))
162             pygame.draw.circle(screen,"white",(width//2,height-bottom),unity)
163             drawgrid(width,height)
164 
165         else:
166             screen.fill("white")
167             pygame.draw.rect(screen,"red",(width//2 - unity*0.5,0,unity,height))
168             pygame.draw.circle(screen,"white",(width//2,height-bottom),unity)
169             drawgrid(width,height)
170 
171         if oldx < newx :
172             if xmove <= width//2 + newx*unity:
173                 pygame.draw.circle(screen,"black",(xmove,ymove),5,0)
174                 pygame.draw.line(screen, "black", (widthpixel(oldx),heightpixel(
    oldy)),(xmove,ymove))
175                 i = i + 1
176             else:
177                 i = 0
178                 j = j+1
179 
180         else:
181             if width//2 + newx*unity <= xmove :
182                 pygame.draw.circle(screen,"black",(xmove,ymove),5,0)
183                 pygame.draw.line(screen, "black", (widthpixel(oldx),heightpixel(
    oldy)),(xmove,ymove))
184                 i = i + 1
185             else:
```

```
                i = 0
                j = j+1

        for k in range(j):
            pygame.draw.circle(screen,"black",(widthpixel(xlist[k]),heightpixel(
    ylist[k])),5,0)

        if j > 1:
            for k in range(j-1):
                pygame.draw.line(screen, "black", (widthpixel(xlist[k]),
    heightpixel(ylist[k])),(widthpixel(xlist[k+1]),heightpixel(ylist[k+1])))

    elif len(xlist) == 1:
        screen.fill("white")
        drawgrid(width,height)
        pygame.draw.rect(screen,"green",(width//2 - unity*0.5,0,unity,height))
        pygame.draw.circle(screen,"white",(width//2,height-bottom),unity)
        pygame.draw.circle(screen,"black",(widthpixel(xlist[-1]),heightpixel(
    ylist[-1])),5,0)

    elif j >= len(xlist) :
        i = 0
        j = 1
        xlist.clear()
        ylist.clear()

    #Dessiner le graphique
    pygame.draw.line(screen,"black",(width//2,0),(width//2,height-bottom+15))
    pygame.draw.line(screen,"black",(0,height-bottom),(width,height-bottom))
    pygame.draw.line(screen,"black",(width//2,0),(width//2+10,10))
    pygame.draw.line(screen,"black",(width//2,0),(width//2-10,10))
    pygame.draw.line(screen,"black",(width,height-bottom),(width-10,height-
    bottom+10))
    pygame.draw.line(screen,"black",(width,height-bottom),(width-10,height-
    bottom-10))
    pygame.draw.arc(screen,"black",(width//2-unity,height-bottom-unity,unity*2,
    unity*2),0,3.1416)
    pygame.draw.line(screen,"black",(width//2+unity*0.5,height-bottom-unity
    *0.866),(width//2+unity*0.5,0))
    pygame.draw.line(screen,"black",(width//2-unity*0.5,height-bottom-unity
    *0.866),(width//2-unity*0.5,0))

    pygame.display.update()
    clock.tick(FPS)

#Fermer la fenêtre et quitter le programme
pygame.quit()
sys.exit()
```

D Fractions continues

```
import pygame, sys
from pygame.locals import *
import numpy as np
from math import sqrt, asin, sin
from random import randint, random, uniform

#Création de la surface de dessin
pygame.init()
pygame.font.init()
width = 801
height = 550
Height = 800
bottom = 30
unity = 100
size = (width,Height)
screen = pygame.display.set_mode(size)
myfont = pygame.font.SysFont('freesansbold.ttf', 30)

def S(x,y):
    newx = -x/(x**2 + y**2)
    newy = y/(x**2 + y**2)
    return (newx,newy)

def T(x,y):
    return (x+1,y)

def Tinverse(x,y):
    return (x-1,y)

def drawgrid(width,height):
    pygame.draw.line(screen,"black",(widthpixel(0.5),heightpixel(-0.03)),(widthpixel(0.5),heightpixel(0.03)))
    pygame.draw.line(screen,"black",(widthpixel(-0.5),heightpixel(-0.03)),(widthpixel(-0.5),heightpixel(0.03)))

    for m in range(1,(width//100)//2):
        pygame.draw.line(screen,"gainsboro",(widthpixel(m),height-bottom ),(widthpixel(m),0))
        pygame.draw.line(screen,"gainsboro",(widthpixel(-m),height-bottom ),(widthpixel(-m),0))
        pygame.draw.line(screen,"black",(widthpixel(m),heightpixel(-0.05)),(widthpixel(m),heightpixel(0.05)))
        pygame.draw.line(screen,"black",(widthpixel(-m),heightpixel(-0.05)),(widthpixel(-m),heightpixel(0.05)))

    for n in range(1,height//100):
        pygame.draw.line(screen,"gainsboro",(0,heightpixel(n)),(width//2 - unity*0.5,heightpixel(n)))
        pygame.draw.line(screen,"gainsboro",(width//2 + unity*0.5,heightpixel(n)
```

```
    ),(width,heightpixel(n)))
43          pygame.draw.line(screen,"black",(widthpixel(-0.05),heightpixel(n)),(
    widthpixel(0.05),heightpixel(n)))
44
45  def widthpixel(x):
46      return width//2 + x*unity
47
48  def heightpixel(y):
49      return height - bottom - y*unity
50
51  def xaxis(x):
52      return (x-width//2)/unity
53
54  def yaxis(y):
55      return -(y-height+bottom)/unity
56
57  def distance(x1,y1,x2,y2):
58      return sqrt((x1-x2)**2 + (y1-y2)**2)
59
60  def combinedlistprint(list1,list2):
61      combinedlist = []
62      if len(list1) == len(list2):
63          for i in range(len(list1)):
64              print(str(round(list1[i],3))+" + " + str(round(list2[i],2)) + "i")
65
66  def listtostring(way_to_domain,oldx,oldy,newx,newy):
67      if oldx != newx or oldy != newy:
68          way = "( "+str(round(oldx,3))+" + "+str(round(oldy,3))+"i)"
69          for i in range(len(way_to_domain)):
70              way = str(way_to_domain[i]) + " " +way
71          print(way + " = " + str(round(newx,3)) + " + " +str(round(newy,3))+ "i")
72
73  def indomain(x,y):
74      return ((-0.5 < x <0 and x**2 + y**2 > 1) or (0 <= x <= 0.5 and x**2 + y**2
    >= 1))
75
76  def update_points(oldx,oldy,newx,newy,i):
77      if oldx != newx :
78          m = (oldy - newy)/(oldx - newx)
79          b = newy - m*newx
80          step = distance(oldx,oldy,newx,newy)/100
81          if oldx < newx :
82              movingx = oldx + step*i
83              movingy = m*movingx + b
84              return (widthpixel(movingx),heightpixel(movingy))
85          else:
86              movingx = oldx - step*i
87              movingy = m*movingx + b
88              return (widthpixel(movingx),heightpixel(movingy))
89
90  def prettylist(list1):
```

```
91      list2 = []
92      i = 0
93      while i < len(list1) and len(list1) != 0 :
94          a = 1
95          j = i + 1
96
97          while j < len(list1):
98
99              if list1[i] == list1[j]:
100                 a = a + 1
101                 j = j + 1
102             else:
103                 j = len(list1)
104
105         if a > 1:
106             if list1[i] == "T^(-1)":
107                 list2.append("T^(-"+str(a)+")")
108             else:
109                 list2.append(list1[i]+"^"+str(a))
110         else:
111             list2.append(list1[i])
112
113         i = i + a
114
115     return list2
116
117 def fraclist(list1):
118     list2 = []
119     i = 0
120     while i < len(list1) and len(list1) != 0 :
121         a = 1
122         j = i + 1
123
124         while j < len(list1):
125
126             if list1[i] == list1[j]:
127                 a = a + 1
128                 j = j + 1
129             else:
130                 j = len(list1)
131
132         if list1[i] == "T^(-1)":
133             #T^(-1) correspond à 1 avec un exposant négatif
134             list2.append((1,-int(a)))
135         elif list1[i] == "T":
136             #T correspond à 1 avec un exposant positif
137             list2.append((1,int(a)))
138         elif list1[i] == "S":
139             #S correspond à 0
140             list2.append((0,int(a)))
141
```

```
142         i = i + a
143 
144     return list2
145 
146 def signof(number):
147     if number < 0:
148         return " - "
149     else:
150         return ""
151 
152 
153 def frac1S(list1,k,l):
154 
155     flist = fraclist(list1)
156 
157     if flist[0][0]==0:
158 
159         if len(flist) == 1:
160             calculation1 = myfont.render("z", False, (0, 0, 0))
161             screen.blit(calculation1,(width//2 + 90 + k, height + (Height -
    height)//4 + 20 + l - 20))
162 
163             minus1 = myfont.render(" - 1", False, (0, 0, 0))
164             screen.blit(minus1,(width//2 + 70 + k, height + (Height - height)//4
     - 15 + l - 20))
165 
166             pygame.draw.line(screen, "black", (width//2 + 35 + k, height + (
    Height - height)//4 + 12 + l- 20),(width//2 + 145 + k, height + (Height -
    height)//4 + 12 - 20 + l))
167 
168         else:
169             calculation1 = myfont.render("z", False, (0, 0, 0))
170             screen.blit(calculation1,(width//2 + 120 + k, height + (Height -
    height)//4 + 20 + l - 20))
171 
172             minus1 = myfont.render(" - 1", False, (0, 0, 0))
173             screen.blit(minus1,(width//2 + 100 + k, height + (Height - height)
    //4 - 15 + l- 20))
174 
175             pygame.draw.line(screen, "black", (width//2 + 70 + k, height + (
    Height - height)//4 + 12 + l- 20),(width//2 + 180 + k, height + (Height -
    height)//4 + 12 + l- 20))
176 
177             if len(flist)>=2:
178 
179                 if signof(flist[1][1]) == " - ":
180                     calculation2 = myfont.render(signof(flist[1][1])+str(abs(
    flist[1][1]))+" + ", False, (0, 0, 0))
181                     screen.blit(calculation2,(width//2 + 20 + k , height + (
    Height - height)//4 + l- 20))
182                 else:
```

```
183                     calculation2 = myfont.render(signof(flist[1][1])+str(abs(
    flist[1][1]))+" + ", False, (0, 0, 0))
184                     screen.blit(calculation2,(width//2 + 25 + k, height + (
    Height - height)//4 + l- 20))
185 
186     else:
187 
188         minus1 = myfont.render(" - 1", False, (0, 0, 0))
189         screen.blit(minus1,(width//2 + 100 + k, height + (Height - height)//4 -
    15 + l- 20))
190 
191         pygame.draw.line(screen, "black", (width//2 + 70 + k, height + (Height -
     height)//4 + 12 + l- 20),(width//2 + 180 + k, height + (Height - height)//4
     + 12 + l- 20))
192 
193         if signof(flist[0][1]) == " - ":
194             calculation1 = myfont.render(signof(flist[0][1])+str(abs(flist
    [0][1]))+" + z", False, (0, 0, 0))
195             screen.blit(calculation1,(width//2 + 85 + k, height + (Height -
    height)//4 + 20 + l- 20))
196         else:
197             calculation1 = myfont.render(signof(flist[0][1])+str(abs(flist
    [0][1]))+" + z", False, (0, 0, 0))
198             screen.blit(calculation1,(width//2 + 100 + k, height + (Height -
    height)//4 + 20 + l- 20))
199 
200         if len(flist)>=3:
201 
202             if signof(flist[2][1]) == " - ":
203                 calculation2 = myfont.render(signof(flist[2][1])+str(abs(flist
    [2][1]))+" + ", False, (0, 0, 0))
204                 screen.blit(calculation2,(width//2 + 20 + k, height + (Height -
    height)//4 + l- 20))
205             else:
206                 calculation2 = myfont.render(signof(flist[2][1])+str(abs(flist
    [2][1]))+" + ", False, (0, 0, 0))
207                 screen.blit(calculation2,(width//2 + 35 + k , height + (Height -
     height)//4 + l- 20))
208 
209 
210 def frac2S(list1,k,l):
211 
212     flist = fraclist(list1)
213 
214     if len(flist) > 4:
215         if flist[4][0] == 0:
216             del flist[4:]
217         else:
218             del flist[5:]
219 
220     if flist[-1][0]==0:
```

```
221
222         minus1 = myfont.render(" - 1", False, (0, 0, 0))
223         screen.blit(minus1,(width//2 + 110 + k, height + (Height - height)//4 -
    15 + l- 20))
224
225         pygame.draw.line(screen, "black", (width//2 + 30 + k, height + (Height -
     height)//4 + 12 + l- 20),(width//2 + 220 + k, height + (Height - height)//4
     + 12 + l- 20))
226
227         frac1S(list1,20+k,40+l)
228
229     else:
230
231         minus1 = myfont.render(" - 1", False, (0, 0, 0))
232         screen.blit(minus1,(width//2 + 125 + k, height + (Height - height)//4 -
    15 + l- 20))
233
234         pygame.draw.line(screen, "black", (width//2 + 70 + k, height + (Height -
     height)//4 + 12 + l- 20),(width//2 + 250 + k, height + (Height - height)//4
     + 12 + l- 20))
235
236         frac1S(list1,55+k,40+l)
237
238         if signof(flist[-1][1]) == " - ":
239             calculation1 = myfont.render(signof(flist[-1][1])+str(abs(flist
    [-1][1]))+" + ", False, (0, 0, 0))
240             screen.blit(calculation1,(width//2 + 20 + k, height + (Height -
    height)//4 + l- 20))
241         else:
242             calculation1 = myfont.render(signof(flist[-1][1])+str(abs(flist
    [-1][1]))+" + ", False, (0, 0, 0))
243             screen.blit(calculation1,(width//2 + 20 + k, height + (Height -
    height)//4  + l- 20))
244
245
246 def frac3S(list1,k,l):
247
248     flist = fraclist(list1)
249
250     if len(flist) > 6:
251         if flist[6][0] == 0:
252             del flist[6:]
253         else:
254             del flist[7:]
255
256     if flist[-1][0]==0:
257
258         minus1 = myfont.render(" - 1", False, (0, 0, 0))
259         screen.blit(minus1,(width//2 + 120 + k, height + (Height - height)//4 -
    15 + l- 20))
260
```

```
261         pygame.draw.line(screen, "black", (width//2 + 10 + k, height + (Height -
     height)//4 + 12 + l- 20),(width//2 + 290 + k, height + (Height - height)//4
     + 12 + l- 20))
262 
263         frac2S(list1,20+k,40+l)
264 
265     else:
266 
267         minus1 = myfont.render(" - 1", False, (0, 0, 0))
268         screen.blit(minus1,(width//2 + 160 + k, height + (Height - height)//4 -
    15 + l- 20))
269 
270         pygame.draw.line(screen, "black", (width//2 + 70 + k, height + (Height -
     height)//4 + 12 + l- 20),(width//2 + 315 + k, height + (Height - height)//4
     + 12 + l- 20))
271 
272         frac2S(list1,55+k,40+l)
273 
274         if signof(flist[-1][1]) == " - ":
275             calculation1 = myfont.render(signof(flist[-1][1])+str(abs(flist
    [-1][1]))+" + ", False, (0, 0, 0))
276             screen.blit(calculation1,(width//2 + 20 + k, height + (Height -
    height)//4 + l- 20))
277         else:
278             calculation1 = myfont.render(signof(flist[-1][1])+str(abs(flist
    [-1][1]))+" + ", False, (0, 0, 0))
279             screen.blit(calculation1,(width//2 + 20 + k, height + (Height -
    height)//4  + l- 20))
280 
281 
282 def frac4S(list1,k,l):
283 
284     flist = fraclist(list1)
285 
286     if flist[-1][0]==0:
287 
288         minus1 = myfont.render(" - 1", False, (0, 0, 0))
289         screen.blit(minus1,(width//2 + 135 + k, height + (Height - height)//4 -
    15 + l- 20))
290 
291         pygame.draw.line(screen, "black", (width//2 + 10 + k, height + (Height -
     height)//4 + 12 + l- 20),(width//2 + 350 + k, height + (Height - height)//4
     + 12 + l- 20))
292 
293         frac3S(list1,20+k,40+l)
294 
295     else:
296 
297         minus1 = myfont.render(" - 1", False, (0, 0, 0))
298         screen.blit(minus1,(width//2 + 180 + k, height + (Height - height)//4 -
    15 + l- 20))
```

```
299
300         pygame.draw.line(screen, "black", (width//2 + 70 + k, height + (Height -
        height)//4 + 12 + l- 20),(width//2 + 380 + k, height + (Height - height)//4
        + 12 + l- 20))
301
302         frac3S(list1,55+k,40+l)
303
304         if signof(flist[-1][1]) == " - ":
305             calculation1 = myfont.render(signof(flist[-1][1])+str(abs(flist
        [-1][1]))+" + ", False, (0, 0, 0))
306             screen.blit(calculation1,(width//2 + 20 + k, height + (Height -
        height)//4 + l- 20))
307         else:
308             calculation1 = myfont.render(signof(flist[-1][1])+str(abs(flist
        [-1][1]))+" + ", False, (0, 0, 0))
309             screen.blit(calculation1,(width//2 + 20 + k, height + (Height -
        height)//4  + l- 20))
310
311 def drawfrac(list1):
312
313     flist = fraclist(list1)
314
315     if len(flist) > 0:
316
317         ftext = "z"
318         d = 0
319
320         for i in range(len(prettylist(list1))):
321             ftext = str(prettylist(list1)[i]) + " " + ftext
322
323             if flist[i][0]==0:
324                 d = d + 20
325             else:
326                 if flist[i][1] < 0:
327                     d = d + 70
328                 elif flist[i][1] == 1:
329                     d = d + 20
330                 else:
331                     d = d + 40
332
333
334         textsurface = myfont.render(ftext, False, (0, 0, 0))
335         screen.blit(textsurface,(width//2 - d - 30 ,height + (Height - height)
        //4- 20))
336
337         equalsign = myfont.render("=", False, (0, 0, 0))
338         screen.blit(equalsign,(width//2- 10, height + (Height - height)//4- 20))
339
340
341         if list1.count("S")==0:
342             calculation = myfont.render(signof(flist[0][1])+str(abs(flist[0][1])
```

```
    )+" + z", False, (0, 0, 0))
343             screen.blit(calculation,(width//2 + 30, height + (Height - height)
    //4- 20))
344 
345         elif list1.count("S")==1:
346 
347             frac1S(list1,0,0)
348 
349         elif list1.count("S")==2:
350 
351             frac2S(list1,0,0)
352 
353         elif list1.count("S")==3:
354 
355             frac3S(list1,0,0)
356 
357         elif list1.count("S")==4:
358 
359             frac4S(list1,0,0)
360 
361 
362 #Titre de la fenêtre
363 pygame.display.set_caption("Mathématiques Expérimentales ")
364 screen.fill("white")
365 
366 #Fréquence d'image
367 FPS = 60
368 clock = pygame.time.Clock()
369 
370 xlist = []
371 ylist = []
372 way_to_domain = []
373 
374 i = 0
375 j = 1
376 
377 oldz = ""
378 newz = ""
379 
380 #Boucle principale
381 done = False
382 while not done:
383     for event in pygame.event.get():
384         if event.type == QUIT:
385             done = True
386 
387         elif event.type == MOUSEBUTTONDOWN:
388             if event.button == 1:
389                 x_mouse = pygame.mouse.get_pos()[0]
390                 y_mouse = pygame.mouse.get_pos()[1]
391 
```

```
392                 if yaxis(y_mouse)>0:
393                     x = xaxis(x_mouse)
394                     y = yaxis(y_mouse)
395                     xlist.clear()
396                     ylist.clear()
397                     xlist.append(x)
398                     ylist.append(y)
399                     way_to_domain = []
400                     screen.fill("white")
401 
402                     while not indomain(xlist[-1],ylist[-1]):
403                         currentx = xlist[-1]
404                         currenty = ylist[-1]
405 
406                         if sqrt(currentx**2 + currenty**2) < 1:
407                             xlist.append(S(currentx,currenty)[0])
408                             ylist.append(S(currentx,currenty)[1])
409                             pygame.draw.circle(screen,"black",(widthpixel(xlist
    [-1]),heightpixel(ylist[-1])),5,0)
410                             way_to_domain.append("S")
411 
412                         else:
413                             if currentx > 0:
414                                 xlist.append(Tinverse(currentx,currenty)[0])
415                                 ylist.append(Tinverse(currentx,currenty)[1])
416                                 pygame.draw.circle(screen,"black",(widthpixel(
    xlist[-1]),heightpixel(ylist[-1])),5,0)
417                                 way_to_domain.append("T^(-1)")
418                             else:
419                                 xlist.append(T(currentx,currenty)[0])
420                                 ylist.append(T(currentx,currenty)[1])
421                                 pygame.draw.circle(screen,"black",(widthpixel(
    xlist[-1]),heightpixel(ylist[-1])),5,0)
422                                 way_to_domain.append("T")
423 
424                     combinedlistprint(xlist,ylist)
425                     listtostring(way_to_domain,xlist[0],ylist[0],xlist[-1],ylist
    [-1])
426                     listtostring(prettylist(way_to_domain),xlist[0],ylist[0],
    xlist[-1],ylist[-1])
427 
428                     oldz = str(round(xlist[0],3))+" + "+str(round(ylist[0],3))+"
    i"
429                     if len(xlist)!=1:
430                         newz = str(round(xlist[-1],3))+" + "+str(round(ylist
    [-1],3))+"i"
431                     else:
432                         newz = "000"
433 
434                     print()
435 
```

```
436     if len(xlist) > 1 and j < len(xlist):
437         oldx = xlist[j-1]
438         oldy = ylist[j-1]
439         newx = xlist[j]
440         newy = ylist[j]
441 
442         xmove = update_points(oldx,oldy,newx,newy,i)[0]
443         ymove = update_points(oldx,oldy,newx,newy,i)[1]
444 
445         if indomain(xaxis(xmove),yaxis(ymove)) and j == (len(xlist)-1):
446             screen.fill("white")
447             pygame.draw.rect(screen,"green",(width//2 - unity*0.5,0,unity,height
    ))
448             pygame.draw.circle(screen,"white",(width//2,height-bottom),unity)
449             drawgrid(width,height)
450 
451         else:
452             screen.fill("white")
453             pygame.draw.rect(screen,"red",(width//2 - unity*0.5,0,unity,height))
454             pygame.draw.circle(screen,"white",(width//2,height-bottom),unity)
455             drawgrid(width,height)
456 
457         if oldx < newx :
458             if xmove <= width//2 + newx*unity:
459                 pygame.draw.circle(screen,"black",(xmove,ymove),5,0)
460                 pygame.draw.line(screen, "black", (widthpixel(oldx),heightpixel(
    oldy)),(xmove,ymove))
461                 i = i + 1
462             else:
463                 i = 0
464                 j = j+1
465 
466         else:
467             if width//2 + newx*unity <= xmove :
468                 pygame.draw.circle(screen,"black",(xmove,ymove),5,0)
469                 pygame.draw.line(screen, "black", (widthpixel(oldx),heightpixel(
    oldy)),(xmove,ymove))
470                 i = i + 1
471             else:
472                 i = 0
473                 j = j+1
474 
475         for k in range(j):
476             pygame.draw.circle(screen,"black",(widthpixel(xlist[k]),heightpixel(
    ylist[k])),5,0)
477 
478         if j > 1:
479             for k in range(j-1):
480                 pygame.draw.line(screen, "black", (widthpixel(xlist[k]),
    heightpixel(ylist[k])),(widthpixel(xlist[k+1]),heightpixel(ylist[k+1])))
481 
```

```
482     elif len(xlist) == 1:
483         screen.fill("white")
484         drawgrid(width,height)
485         pygame.draw.rect(screen,"green",(width//2 - unity*0.5,0,unity,height))
486         pygame.draw.circle(screen,"white",(width//2,height-bottom),unity)
487         pygame.draw.circle(screen,"black",(widthpixel(xlist[-1]),heightpixel(
    ylist[-1])),5,0)
488 
489     elif j >= len(xlist) :
490         i = 0
491         j = 1
492         xlist.clear()
493         ylist.clear()
494 
495     #Dessiner le graphique
496     pygame.draw.line(screen,"black",(width//2,0),(width//2,height-bottom+15))
497     pygame.draw.line(screen,"black",(0,height-bottom),(width,height-bottom))
498     pygame.draw.line(screen,"black",(width//2,0),(width//2+10,10))
499     pygame.draw.line(screen,"black",(width//2,0),(width//2-10,10))
500     pygame.draw.line(screen,"black",(width,height-bottom),(width-10,height-
    bottom+10))
501     pygame.draw.line(screen,"black",(width,height-bottom),(width-10,height-
    bottom-10))
502     pygame.draw.arc(screen,"black",(width//2-unity,height-bottom-unity,unity*2,
    unity*2),0,3.1416)
503     pygame.draw.line(screen,"black",(width//2+unity*0.5,height-bottom-unity
    *0.866),(width//2+unity*0.5,0))
504     pygame.draw.line(screen,"black",(width//2-unity*0.5,height-bottom-unity
    *0.866),(width//2-unity*0.5,0))
505 
506 
507     if oldz != "":
508         if newz == "000":
509             samez = myfont.render(" z = "+str(oldz)+" se trouve déjà dans D",
    False, (0, 0, 0))
510             screen.blit(samez,(width//2 - 170 ,height + (Height - height)//4))
511 
512         else:
513             drawfrac(way_to_domain)
514 
515             samez = myfont.render("où z = "+str(oldz)+" est transformé en "+str(
    newz), False, (0, 0, 0))
516             screen.blit(samez,(width//2 - 200 ,Height - 30))
517 
518 
519     pygame.display.update()
520     clock.tick(FPS)
521 
522 #Fermer la fenêtre et quitter le programme
523 pygame.quit()
524 sys.exit()
```

E Programme d'animation avec Pygame sur les surfaces de Hecke

```
import pygame, sys
from pygame.locals import *
import numpy as np
from math import sqrt, asin, sin, cos
from random import randint, random, uniform

#Création de la surface de dessin
q = int(input("Veuillez entrer un entier positif q: "))
if q != 0:
    a = 2*cos(np.pi/q)

    pygame.init()
    width = 801
    height = 700
    bottom = 30
    unity = 100
    size = (width,height)
    screen = pygame.display.set_mode(size)

    def S(x,y):
        newx = -x/(x**2 + y**2)
        newy = y/(x**2 + y**2)
        return (newx,newy)

    def T(x,y):
        return (x+a,y)

    def Tinverse(x,y):
        return (x-a,y)

    def drawgrid(width,height):
        pygame.draw.line(screen,"black",(widthpixel(a/2),heightpixel(-0.03)),(widthpixel(a/2),heightpixel(0.03)))
        pygame.draw.line(screen,"black",(widthpixel(-a/2),heightpixel(-0.03)),(widthpixel(-a/2),heightpixel(0.03)))

        for m in range(1,(width//100)//2):
            pygame.draw.line(screen,"gainsboro",(widthpixel(m),height-bottom ),(widthpixel(m),0))
            pygame.draw.line(screen,"gainsboro",(widthpixel(-m),height-bottom ),(widthpixel(-m),0))
            pygame.draw.line(screen,"black",(widthpixel(m),heightpixel(-0.05)),(widthpixel(m),heightpixel(0.05)))
            pygame.draw.line(screen,"black",(widthpixel(-m),heightpixel(-0.05)),(widthpixel(-m),heightpixel(0.05)))

        for n in range(1,height//100):
            pygame.draw.line(screen,"gainsboro",(0,heightpixel(n)),(width//2 - unity*a/2,heightpixel(n)))
```

```
43          pygame.draw.line(screen,"gainsboro",(width//2 + unity*a/2,
    heightpixel(n)),(width,heightpixel(n)))
44          pygame.draw.line(screen,"black",(widthpixel(-0.05),heightpixel(n)),(
    widthpixel(0.05),heightpixel(n)))
45
46      def widthpixel(x):
47          return width//2 + x*unity
48
49      def heightpixel(y):
50          return height - bottom - y*unity
51
52      def xaxis(x):
53          return (x-width//2)/unity
54
55      def yaxis(y):
56          return -(y-height+bottom)/unity
57
58      def distance(x1,y1,x2,y2):
59          return sqrt((x1-x2)**2 + (y1-y2)**2)
60
61      def combinedlistprint(list1,list2):
62          combinedlist = []
63          if len(list1) == len(list2):
64              for i in range(len(list1)):
65                  print(str(round(list1[i],3))+" + " + str(round(list2[i],2)) + "i
    ")
66
67      def listtosting(way_to_domain,oldx,oldy,newx,newy):
68          if oldx != newx or oldy != newy:
69              way = "("+str(round(oldx,3))+" + "+str(round(oldy,3))+"i)"
70              for i in range(len(way_to_domain)):
71                  way = str(way_to_domain[i]) + way
72              print(way + " = " + str(round(newx,3)) + " + " +str(round(newy,3))+
    "i ")
73
74
75
76      def indomain(x,y):
77          return ((-a/2 < x <0 and x**2 + y**2 > 1) or (0 <= x <= a/2 and x**2 + y
    **2 >= 1))
78
79      def update_points(oldx,oldy,newx,newy,i):
80          if oldx != newx :
81              m = (oldy - newy)/(oldx - newx)
82              b = newy - m*newx
83              step = distance(oldx,oldy,newx,newy)/100
84              if oldx < newx :
85                  movingx = oldx + step*i
86                  movingy = m*movingx + b
87                  return (widthpixel(movingx),heightpixel(movingy))
88              else:
```

```
                movingx = oldx - step*i
                movingy = m*movingx + b
                return (widthpixel(movingx),heightpixel(movingy))


    #Titre de la fenêtre
    pygame.display.set_caption("Mathématiques Expérimentales")

    screen.fill("white")

    #Fréquence d'image
    FPS = 60
    clock = pygame.time.Clock()

    xlist = []
    ylist = []

    i = 0
    j = 1

    #Boucle principale
    done = False
    while not done:
        for event in pygame.event.get():
            if event.type == QUIT:
                done = True

            elif event.type == MOUSEBUTTONDOWN:
                if event.button == 1:
                    x_mouse = pygame.mouse.get_pos()[0]
                    y_mouse = pygame.mouse.get_pos()[1]
                    x = xaxis(x_mouse)
                    y = yaxis(y_mouse)
                    xlist.clear()
                    ylist.clear()
                    xlist.append(x)
                    ylist.append(y)
                    way_to_domain = []
                    screen.fill("white")


                    while not indomain(xlist[-1],ylist[-1]):
                        currentx = xlist[-1]
                        currenty = ylist[-1]
                        if sqrt(currentx**2 + currenty**2) < 1:
                            xlist.append(S(currentx,currenty)[0])
                            ylist.append(S(currentx,currenty)[1])
                            pygame.draw.circle(screen,"black",(widthpixel(xlist
[-1]),heightpixel(ylist[-1])),5,0)
                            way_to_domain.append("S")
                        else:
```

```
139                             if currentx > 0:
140                                 xlist.append(Tinverse(currentx,currenty)[0])
141                                 ylist.append(Tinverse(currentx,currenty)[1])
142                                 pygame.draw.circle(screen,"black",(widthpixel(
    xlist[-1]),heightpixel(ylist[-1])),5,0)
143                                 way_to_domain.append("T^(-1)")
144                             else:
145                                 xlist.append(T(currentx,currenty)[0])
146                                 ylist.append(T(currentx,currenty)[1])
147                                 pygame.draw.circle(screen,"black",(widthpixel(
    xlist[-1]),heightpixel(ylist[-1])),5,0)
148                                 way_to_domain.append("T")
149
150
151                     combinedlistprint(xlist,ylist)
152                     listtosting(way_to_domain,xlist[0],ylist[0],xlist[-1],ylist
    [-1])
153                     print()
154
155         if len(xlist) > 1 and j < len(xlist):
156             oldx = xlist[j-1]
157             oldy = ylist[j-1]
158             newx = xlist[j]
159             newy = ylist[j]
160
161             xmove = update_points(oldx,oldy,newx,newy,i)[0]
162             ymove = update_points(oldx,oldy,newx,newy,i)[1]
163
164             if indomain(xaxis(xmove),yaxis(ymove)) and j == (len(xlist)-1):
165                 screen.fill("white")
166                 pygame.draw.rect(screen,"green",(width//2 - unity*a/2,0,unity*a,
    height))
167                 pygame.draw.circle(screen,"white",(width//2,height-bottom),unity
    )
168                 drawgrid(width,height)
169
170             else:
171                 screen.fill("white")
172                 pygame.draw.rect(screen,"red",(width//2 - unity*a/2,0,unity*a,
    height))
173                 pygame.draw.circle(screen,"white",(width//2,height-bottom),unity
    )
174                 drawgrid(width,height)
175
176
177             if oldx < newx :
178                 if xmove <= width//2 + newx*unity:
179                     pygame.draw.circle(screen,"black",(xmove,ymove),5,0)
180                     pygame.draw.line(screen, "black", (widthpixel(oldx),
    heightpixel(oldy)),(xmove,ymove))
181                     i = i + 1
```

```
182                 else:
183                     i = 0
184                     j = j+1
185 
186             else:
187                 if width//2 + newx*unity <= xmove :
188                     pygame.draw.circle(screen,"black",(xmove,ymove),5,0)
189                     pygame.draw.line(screen, "black", (widthpixel(oldx),
    heightpixel(oldy)),(xmove,ymove))
190                     i = i + 1
191                 else:
192                     i = 0
193                     j = j+1
194 
195             for k in range(j):
196                 pygame.draw.circle(screen,"black",(widthpixel(xlist[k]),
    heightpixel(ylist[k])),5,0)
197 
198 
199             if j > 1:
200                 for k in range(j-1):
201                     pygame.draw.line(screen, "black", (widthpixel(xlist[k]),
    heightpixel(ylist[k])),(widthpixel(xlist[k+1]),heightpixel(ylist[k+1])))
202 
203         elif len(xlist) == 1:
204             screen.fill("white")
205             drawgrid(width,height)
206             pygame.draw.rect(screen,"green",(width//2 - unity*a/2,0,unity*a,
    height))
207             pygame.draw.circle(screen,"white",(width//2,height-bottom),unity)
208             pygame.draw.circle(screen,"black",(widthpixel(xlist[-1]),heightpixel
    (ylist[-1])),5,0)
209 
210         elif j >= len(xlist) :
211             i = 0
212             j = 1
213             xlist.clear()
214             ylist.clear()
215 
216         #Dessiner le graphique
217         pygame.draw.line(screen,"black",(width//2,0),(width//2,height-bottom+15)
    )
218         pygame.draw.line(screen,"black",(0,height-bottom),(width,height-bottom))
219         pygame.draw.line(screen,"black",(width//2,0),(width//2+10,10))
220         pygame.draw.line(screen,"black",(width//2,0),(width//2-10,10))
221         pygame.draw.line(screen,"black",(width,height-bottom),(width-10,height-
    bottom+10))
222         pygame.draw.line(screen,"black",(width,height-bottom),(width-10,height-
    bottom-10))
223         pygame.draw.arc(screen,"black",(width//2-unity,height-bottom-unity,unity
    *2,unity*2),0,3.1416)
```

```
224         pygame.draw.line(screen,"black",(width//2+unity*a/2,height-bottom-unity*
    sqrt(1-a**2/4)),(width//2+unity*a/2,0))
225         pygame.draw.line(screen,"black",(width//2-unity*a/2,height-bottom-unity*
    sqrt(1-a**2/4)),(width//2-unity*a/2,0))
226 
227         pygame.display.update()
228         clock.tick(FPS)
229 
230     #Fermer la fenêtre et quitter le programme
231     pygame.quit()
232     sys.exit()
```