Fonctions_Complexes_4

April 12, 2019

1 Fonctions d'une variable complexe (IV)

2 La sphère de Riemann

Marc Lorenzi
10 avril 2019

```
In [1]: import matplotlib.pyplot as plt
        import cmath
        import math
        import colorsys
        import random

In [2]: plt.rcParams['figure.figsize'] = (10, 10)
```

2.1 1. Une quasi-bijection entre la sphère et le plan

2.1.1 1.1 La sphère de Riemann $\mathcal{S}^2$

Soit $R > 0$ un réel fixé. Soit $\mathcal{S}^2$ l'ensemble des points $(X, Y, Z) \in \mathbb{R}^3$ tels que $X^2 + Y^2 + Z^2 = R^2$. L'ensemble $\mathcal{S}^2$ est la sphère de centre O et de rayon R, nous l'appellerons la **sphère de Riemann**.

Remarque : Dans tous les livres parlant de fonctions de la variable complexe, le réel R vaut 1. Pour des raisons qui apparaîtront plus tard, nous généralisons un peu ...

Notons $N = (0, 0, R)$ le **pôle nord** de $\mathcal{S}^2$. Soi $M = (X, Y, Z) \in \mathcal{S}^2$ un point différent de N. La droite (NM) coupe le plan xOy, d'équation $z = 0$, en un unique point. Quel est ce point ? Notons le $P = (x, y, 0)$. Il existe un réel λ tel que $\overrightarrow{NP} = \lambda\overrightarrow{NM}$ ce qui donne, en termes de coordonnées,

$$x - 0 = \lambda(X - 0), y - 0 = \lambda(Y - 0), 0 - R = \lambda(Z - R)$$

De la dernière égalité, on déduit

$$\lambda = \frac{R}{R - Z}$$

Puis, en reportant :

$$x = \frac{RX}{R - Z}, y = \frac{RY}{R - Z}$$

En identifiant le plan xOy au plan complexe $\mathbb{C}$, nous venons donc de définir une fonction $\varphi : \mathcal{S}^2 \setminus \{N\} \to \mathbb{C}$, définie par

$$\varphi(X, Y, Z) = \frac{R}{R - Z}(X + iY)$$

```
In [3]: def projection_C(X, Y, Z, R):
            x = R * X / (R - Z)
            y = R * Y / (R - Z)
            return x + 1j * y
```

```
In [4]: projection_C(1, 0, 0, 1)
```

```
Out[4]: (1+0j)
```

2.1.2 1.2 La réciproque de φ

IL est facile de voir que la fonction φ est bijective. Pour cela, prenons $z = x + iy \in \mathbb{C}$. Déterminons tous les points $M = (X, Y, Z) \in \mathcal{S}^2$, $M \neq N$, tels que $\varphi(M) = z$. Un tel point convient si et seulement si $(R - Z)x = RX$ et $(R - Z)y = RY$. Élevons au carré, additionnons, et rappelons-nous que $X^2 + Y^2 + Z^2 = R^2$. il vient

$$(R - Z)^2(x^2 + y^2) = R^2(R^2 - Z^2)$$

ou encore, puisque $Z \neq R$,

$$(R - Z)(x^2 + y^2) = R^2(R + Z)$$

On en tire facilement

$$Z = R\frac{x^2 + y^2 - R^2}{x^2 + y^2 + R^2}$$

Calculer X et Y est maintenant une formalité. Remarquons que $R - Z = \frac{2R^3}{x^2+y^2+R^2}$. On en déduit

$$X = \frac{2R^2x}{x^2 + y^2 + R^2} \quad \text{et} \quad Y = \frac{2R^2y}{x^2 + y^2 + R^2}$$

```
In [5]: def anti_projection(z, R):
            x, y = z.real, z.imag
            r = x ** 2 + y ** 2 + R ** 2
            return (2 * R ** 2 * x / r, 2 * R ** 2 * y / r, R - 2 * R ** 3 / r)
```

Si on compose `anti_projection` et `projection` on doit trouver l'identité.

```
In [6]: R = 7
        X, Y, Z = anti_projection(1.234 - 5.678 * 1j, R)
        projection_C(X, Y, Z, R)
```

```
Out[6]: (1.234-5.677999999999999j)
```

Enfin, disons l'identité à ε près :-).

2.1.3 1.3 Le point à l'infini du plan complexe

Le pauvre pôle nord de la sphère n'a pas d'image par φ. Soit $M = (X, Y, Z) \in \mathcal{S}^2$ un point différent de N. Que fait $\varphi(M)$ lorsque M s'approche indéfiniment de N, c'est à dire lorsque $Z \to R$? On a

$$|\varphi(M)|^2 = \frac{R^2}{(R-Z)^2}(X^2+Y^2) = \frac{R^2(R^2-Z^2)}{(R-Z)^2} = \frac{R^2(R+Z)}{R-Z}$$

Le numérateur de cette fraction tend vers $2R^3$ et son dénominateur tend vers 0. Ainsi, $|\varphi(M)|$ tend vers $+\infty$ lorsque M tend vers N : Le nombre complexe $\varphi(M)$ "part à l'infini".

Avez-vous noté les guillemets ? Nous allons les supprimer :-). Rajoutons au plan complexe un point que nous noterons ∞. Notons $\hat{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$ notre "nouveau" plan : nous l'appellerons le **plan complexe étendu**. Maintenant, il est facile de prolonger φ en une bijection de $\mathcal{S}^2$ sur $\hat{\mathbb{C}}$ en posant $\varphi(N) = \infty$. Maintenant, lorsque M tend vers N, nous avons $\varphi(M)$ qui tend vers l'infini. Qui tend ? Il faudrait pour cela pouvoir mesurer la distance entre $\varphi(M)$ et ∞, non ? Eh bien nous allons pour cela définir une distance adéquate.

2.1.4 1.4 Une distance bornée sur le plan complexe

Pour $z, z' \in \hat{\mathbb{C}}$, posons $\delta(z, z') = d(M, M')$ où M et M' sont les antécédents de z et z' par φ et d est la distance usuelle dans l'espace. Si vous préférez :

$$\delta(z, z') = d(\varphi^{-1}(z), \varphi^{-1}(z'))$$

Comme d est une distance et φ est bijective, la fonction δ est une distance dans le plan complexe étendu. Et comme M et M' sont sont la sphère $\mathcal{S}^2$, leur distance est inférieure ou égale au diamètre de la sphère, à savoir $2R$.

Si vous le désirez, nous pouvons calculer explicitement $\delta(z, z')$. Il suffit de reprendre les expressions de M et M' déjà trouvées. Je vous épargne les calculs. On obtient les résultats suivants :

- Si $z, z' \in \mathbb{C}$, alors

$$\delta(z, z') = \frac{2R^2|z'-z|}{\sqrt{|z|^2+R^2}\sqrt{|z'|^2+R^2}}$$

- Si $z \in \mathbb{C}$, alors

$$\delta(z, \infty) = \frac{2R^2}{\sqrt{|z|^2+R^2}}$$

Remarquons avec la dernière égalité que si $|z|$ tend vers $+\infty$, alors $\delta(z, \infty)$ tend vers 0. Nous pourrons donc enlever les modules et tout simplement dire que $z \to \infty$.

```
In [7]: def distance_sphere(z1, z2, R):
            r1 = math.sqrt(abs(z1) ** 2 + R ** 2)
            r2 = math.sqrt(abs(z2) ** 2 + R ** 2)
            return 2 * R ** 2 * abs(z2 - z1) / (r1 * r2)

In [10]: distance_sphere(1, 1j, 1)
```

```
Out[10]: 1.414213562373095
```

Notez l'exemple ci-dessous :

```
In [9]: distance_sphere(-1000, 1000, 1)

Out[9]: 0.003999996000004
```

Nous constatons que $\delta(-1000, 1000)$ est petit. Pourquoi ? Chacun de ces deux nombres est "grand", donc proche de ∞. Par l'inégalité triangulaire appliquée à δ, ils sont donc proches l'un de l'autre.

Remarque : Pour les amateurs de topologie, nous disposons maintenant de **deux** distances dans le plan complexe : la distance usuelle $d(z, z') = |z' - z|$ et la distance δ. On peut montrer que ces deux distances sont **équivalentes**, elles définissent la même topologie sur $\mathbb{C}$.

Par ailleurs, $\hat{\mathbb{C}}$, muni de la distance δ, devient un **espace métrique**. La sphère $\mathcal{S}^2$, elle-aussi, est un espace métrique. On peut démontrer que la bijection $\varphi : \mathcal{S}^2 \to \hat{\mathbb{C}}$ est continue, et que sa réciproque est aussi continue. C'est ce que l'on appelle un **homéomorphisme** entre $\mathcal{S}^2$ et $\hat{\mathbb{C}}$. Nous pourrions aussi bien dire maintenant que $\hat{\mathbb{C}}$ **EST** la sphère de Riemann.

Je n'entrerai pas, excepté dans une ou deux occasions, dans les détails topologiques dans ce qui va suivre.

2.2 2. Dessiner la sphère

2.2.1 2.1 Lancer de rayons

Comment dessiner la sphère de Riemann ? Imaginez un appareil photo placé sur l'axe OX, disons au point $\Omega = (\omega, 0, 0)$ où $\omega > 0$ est un réel fixé. L'appareil photo pointe vers la sphère de Riemann. À l'intérieur de l'appareil se trouve un ensemble de capteurs que nous représenterons par un plan vertical, parallèle à yOz, formé de pixels. Les rayons lumineux issus de la sphère frappent les pixels qui enregistrent la couleur de ces rayons. Nous avons obtenu une image.

En fait, nous allons faire exactement le contraire : c'est la technique du **lancer de rayons**. Nous allons lancer des rayons (c'est à dire des demi-droites) partant du point Ω vers tous les pixels de l'appareil. Si un tel rayon frappe la sphère, on colorie le pixel avec la couleur du point frappé sur la sphère.

Facile à dire, facile à faire ? Au travail. Mais commençons par un peu de géométrie.

2.2.2 2.2 Intersection d'un rayon et de la sphère

Pour simplifier un peu nos calculs, nous supposons que le plan des pixels est le plan d'équation $x = 0$. Nous appellerons ce plan le **plan de projection**.

Rappelons que $\Omega = (\omega, 0, 0)$. Soit $M = (0, y, z)$ un point dans le plan de projection. Soit $\mathcal{D}$ la droite (ΩM). Un point P est sur $\mathcal{D}$ si et seulement si il existe un réel t tel que $\overrightarrow{\Omega P} = t\overrightarrow{\Omega M}$, ou encore $P = \Omega + t\overrightarrow{\Omega M}$.

En termes de coordonnées, cela donne $P = (\omega - t\omega, ty, tz)$ où $t \in \mathbb{R}$. Ce point est sur la sphère $\mathcal{S}^2$ si et seulement si

$$\omega^2(1-t)^2 + t^2y^2 + t^2z^2 = R^2$$

ou encore

$$(\omega^2 + y^2 + z^2)t^2 - 2\omega^2 t + \omega^2 - R^2 = 0$$

Nous avons là une équation du second degré en t. Son discriminant est

$$D = 4(\omega^4 + (\omega^2 + y^2 + z^2)(R^2 - \omega^2)) = 4((y^2 + z^2)(R^2 - \omega^2) + R^2\omega^2)$$

Évidemment, l'écran d'un appareil photo n'est pas un plan infini avec une densité infinie de pixels. Dans la fonction ci-dessous, nous supposons que l'écran est le carré $[-\frac{6}{5}R, \frac{6}{5}R[^2$, et qu'il contient n pixels par ligne et par colonne. Le pixel ligne i, colonne j est d'abord converti en un couple (y, z) de réels, qui sont des coordonnées dans le plan d'équation $x = 0$. Puis on résout l'équation ci-dessus.

- Si le discriminant est strictement négatif, le rayon ne frappe pas la sphère et on renvoie la liste vide.
- Sinon, le rayon frappe la sphère en un ou deux points. On renvoie une liste contenant un unique triplet de réels qui est le point frappé par le rayon : il s'agit du point qui est le plus proche de l'observateur, c'est à dire celui pour lequel t est le plus petit.

Comme j'imagine que vous savez résoudre une équation du second degré, je ne détaille pas plus avant la fonction `intersect_sphere`.

```
In [11]: def intersect_sphere(i, j, n, R, omega):
             y = -1.2 * R + 2.4 * R * j / n
             z = 1.2 * R - 2.4 * R * i / n
             D = (y ** 2 + z ** 2) * (R ** 2 - omega ** 2) + R ** 2 * omega ** 2
             if D < 0: return []
             else:
                 d = math.sqrt(D)
                 t = (omega ** 2 - d) / (y ** 2 + z ** 2 + omega ** 2)
                 return [((1 - t) * omega, t * y, t * z)]
```

2.2.3 2.3 Dessiner

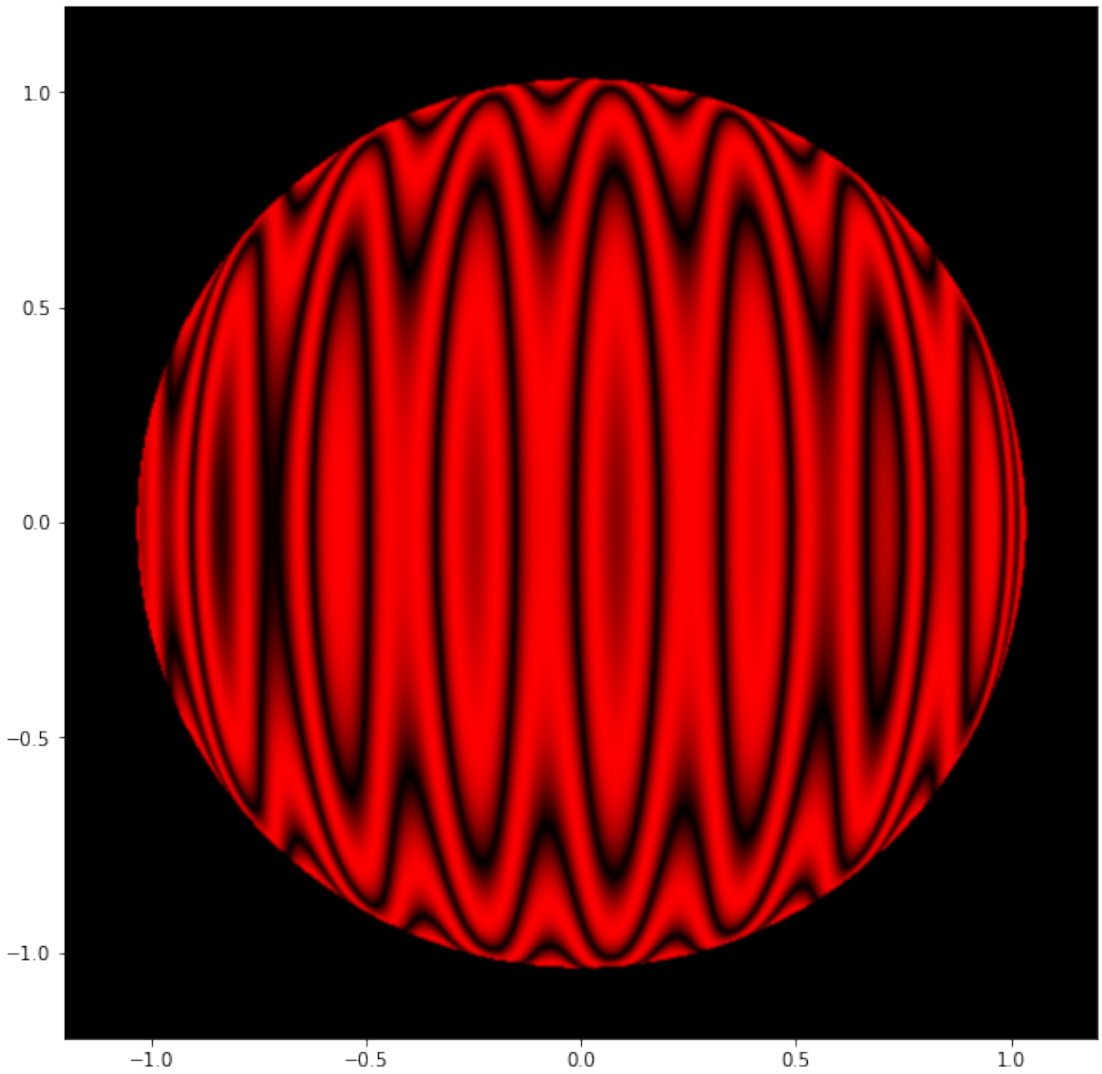
La fonction de dessin est maintenant immédiate. Elle prend un paramètre `methode` qui est une fonction qui renvoie une couleur (une couleur doit être un triplet (r, g, b) de réels entre 0 et 1). Plus les paramètres nécessaires n, R et `omega`. Si vous voulez éviter les surprises, prenz toujours `omega`$> R$.

```
In [12]: def plot_riemann0(methode, n=500, R=5, omega=20):
             ws = [[(0, 0, 0) for i in range(n)] for j in range(n)]
             for i in range(n):
                 for j in range(n):
                     s = intersect_sphere(i, j, n, R, omega)
                     if s != []:
                         X, Y, Z = s[0]
                         ws[i][j] = methode(X, Y, Z, R)
             plt.imshow(ws, interpolation='bicubic', extent=[-1.2, 1.2, -1.2, 1.2])
```

Tentons avec une méthode bête de coloration. On renvoie une couleur dépendant de X et Y.

```
In [17]: def methode_naive(X, Y, Z, R):
             c1 = abs(math.sin(10 * (X + math.sin(5 * Y)) / R))
             c2 = 1.
             c3 = 1.
             return (c1, 0., 0.)

In [18]: plot_riemann0(methode_naive)
```



2.2.4 2.4 Rotations

Voilà une bonne chose de faite, mais ce serait bien si nous pouvions faire tourner la sphère. Dans l'espace, on tourne autour d'une droite, d'un certain angle.

Voici tout d'abord une fonction `rotation_Oz`. Elle prend en paramètres trois réels x, y, z et un angle t. Il renvoie l'image du point (x, y, z) par la rotation d'axe Oz et d'angle t.

```
In [19]: def rotation_Oz(x, y, z, t):
             c, s = math.cos(t), math.sin(t)
             x1 = c * x - s * y
             y1 = s * x + c * y
             return (x1, y1, z)
```

```
In [20]: rotation_Oz(1, 1, 1, math.pi / 4)
```

```
Out[20]: (1.1102230246251565e-16, 1.414213562373095, 1)
```

Et voici de même une fonction qui fait tourner autour de l'axe Oy.

```
In [21]: def rotation_Oy(x, y, z, t):
             c, s = math.cos(t), math.sin(t)
             z1 = c * z - s * x
             x1 = s * z + c * x
             return (x1, y, z1)
```

```
In [22]: rotation_Oy(1, 1, 1, math.pi / 4)
```

```
Out[22]: (1.414213562373095, 1, 1.1102230246251565e-16)
```

2.2.5 2.5 Dessiner (bis)

Réécrivons notre fonction de tracé en lui ajoutant deux paramètres `roty` et `rotz`. Ces deux paramètres contiennent les angles dont on veut faire tourner la sphère, **exprimés en degrés**. Soyons plus précis.

1. On dessine la sphère en la coloriant avec la fonction `methode`.
2. On la fait tourner autour de Oz d'un angle `rotz`.
3. On la fait tourner autour de Oy d'un angle `roty`.

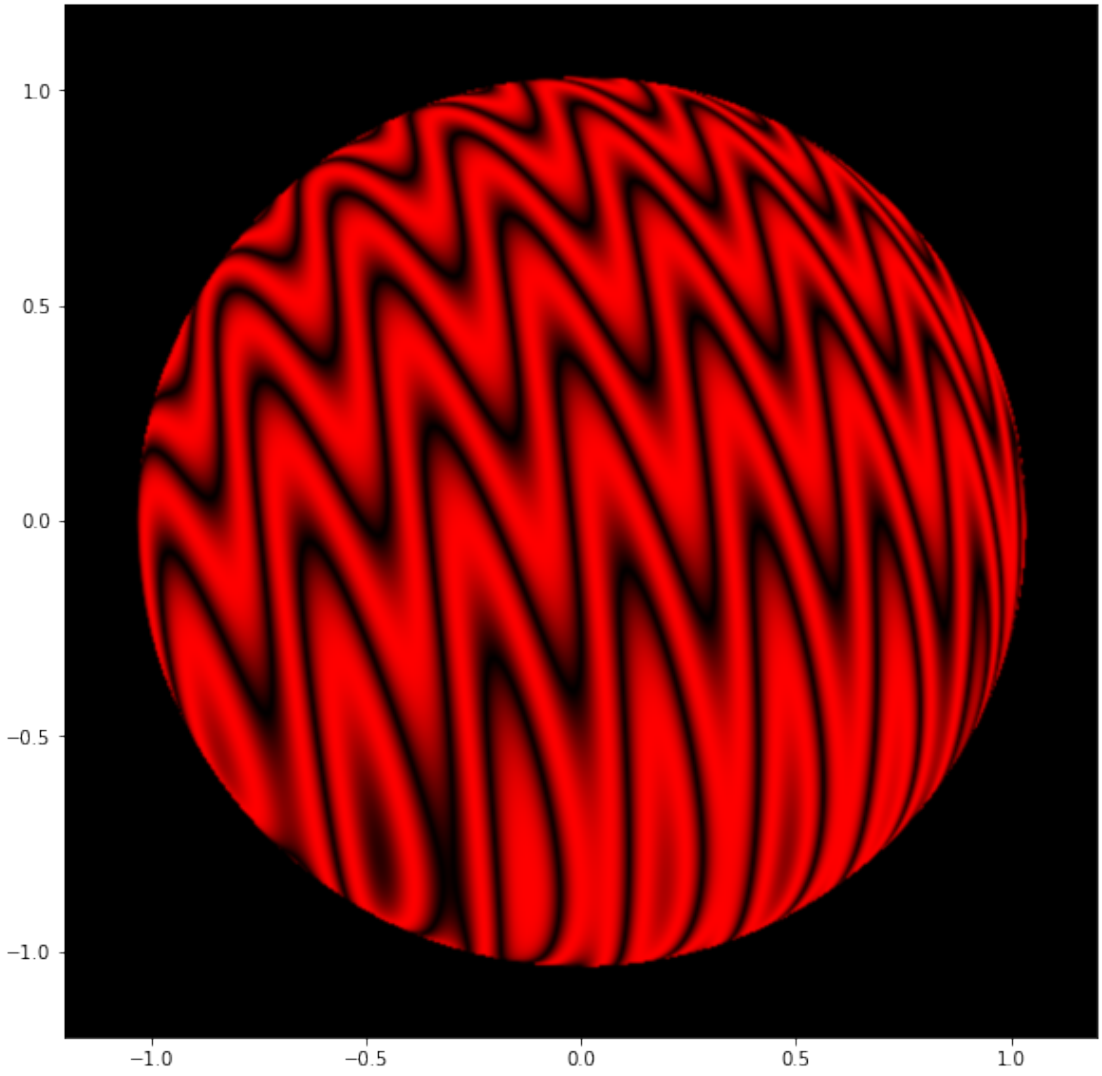
Bon, ça c'est ce qu'on veut, mais on se rend vite compte que notre algorithme de lancer de rayons ne va pas du tout apprécier. Alors nous allons procéder ... à l'envers !

1. On fait tourner la sphère autour de Oy d'un angle $-$`roty`.
2. On fait tourner la sphère autour de Oz d'un angle $-$`rotz`.
3. On dessine la sphère en la coloriant avec la fonction `methode`.

Voici notre nouvelle fonction de tracé.

```
In [23]: def plot_riemann1(methode, roty=0, rotz=0, n=500, R=5, omega=20):
             ws = [[(0, 0, 0) for i in range(n)] for j in range(n)]
             for i in range(n):
                 for j in range(n):
                     s = intersect_sphere(i, j, n, R, omega)
                     if s != []:
                         X0, Y0, Z0 = s[0]
                         X, Y, Z = rotation_Oy(X0, Y0, Z0, -roty * math.pi / 180)
                         X, Y, Z = rotation_Oz(X, Y, Z, -rotz * math.pi / 180)
                         ws[i][j] = methode(X, Y, Z, R)
             plt.imshow(ws, interpolation='bicubic', extent=[-1.2, 1.2, -1.2, 1.2])
```

```
In [24]: plot_riemann1(methode_naive, roty=45, rotz=20)
```



Maintenant que nous savons dessiner la sphère $\mathcal{S}^2$ nous allons utiliser notre tout nouvel outil pour dessiner le graphe de fonctions de $\mathbb{C}$ vers $\mathbb{C}$.

À partir de maintenant, je vais supposer que vous avez lu au moins le premier notebook sur les fonctions holomorphes.

2.2.6 2.6 La grille de coordonnées

Ce serait bien de pouvoir tracer sur la sphère une grille de coordonnées, un peu comme le fait `pyplot` avec la fonction `pyplot.grid`. Mais c'est quoi une grille de coordonnées sur la sphère de Riemann ? Voici l'idée : nous allons prendre une grille de coordonnées (droites horizontales et verticales) dans le plan complexe $\mathbb{C}$. Puis nous allons dé-projeter cette grille sur la sphère $\mathcal{S}^2$. Que devient une droite de $\mathbb{C}$ lorsqu'on la dé-projette ? Eh bien une droite passe par ∞. On devrait donc obtenir une courbe qui passe par le pôle nord de la sphère.

On peut montrer, et je ne le ferai pas ici, que ces courbes sont des cercles passant par le pôle nord N.

Tout d'abord, comment tracer une "droite" ? La fonction `tracer_droite` ci-dessous fait le travail et trace la "droite" passant par $z \in \mathbb{C}$ et dirigée par $u \in \mathbb{C}^*$. Elle prend en paramètres

- Une matrice `ws` représentant les pixels de l'écran.
- Deux nombres complexes z et $u \neq 0$: z est un point de la droite et u est un vecteur directeur de celle-ci.
- Deux paramètres `roty` et `rotz` de rotation.
- R, le rayon de la sphère, et `omega`, l'abscisse de l'observateur.

La fonction déprojette un certain nombre de points de la droite puis projette les dé-projetés sur "l'écran" `ws` en mettant le pixel correspondant à la couleur blanche.

Exercice : Soit $\Omega = (\omega, 0, 0)$. Soit $M = (X, Y, Z)$. Que vaut le produit scalaire $< \overrightarrow{\Omega O}, \overrightarrow{OM} >$? Expliquez la raison d'être du test `if X * (X - omega) + Y ** 2 + Z ** 2 < 0:`.

La fonction `scale` est une bijection de $]-1, 1[$ sur $\mathbb{R}$.

```
In [25]: def scale(t):
             return math.tan(math.pi * t / 2) ** 3
```

```
In [26]: def tracer_droite(ws, z, u, roty, rotz, R, omega, n):
             for t in range(-999, 1000):
                 w = z + u * scale(t / 1000)
                 X, Y, Z = anti_projection(w, R)
                 X, Y, Z = rotation_Oz(X, Y, Z, rotz * math.pi / 180)
                 X, Y, Z = rotation_Oy(X, Y, Z, roty * math.pi / 180)
                 if X * (X - omega) + Y ** 2 + Z ** 2 <= 0:
                     i, j = pixel(X, Y, Z, R, omega, n)
                     if i >= 0 and i < n and j >= 0 and j < n:
                         color = 1.
                         ws[i][j] = (color, color, color)
```

```
In [27]: def pixel(X, Y, Z, R, omega, n):
             y = omega * Y / (omega - X)
             z = omega * Z / (omega - X)
             j = int(0.5 + n * (y + 1.2 * R) / (2.4 * R))
             i = int(0.5 + n * (1.2 * R - z) / (2.4 * R))
             return (i, j)
```

Tracer la grille c'est simplement tracer quelques droites verticales et quelques droites horizontales.

```
In [28]: def tracer_grille(ws, roty, rotz, R, omega, n):
             for x in range(8):
                 tracer_droite(ws, 2 ** x, 1j, roty, rotz, R, omega, n)
                 tracer_droite(ws, -2 ** x, 1j, roty, rotz, R, omega, n)
             for y in range(8):
                 tracer_droite(ws, -1j * 2 ** y, 1, roty, rotz, R, omega, n)
                 tracer_droite(ws, 1j * 2 ** y, 1, roty, rotz, R, omega, n)
             tracer_droite(ws, 0, 1j, roty, rotz, R, omega, n)
             tracer_droite(ws, 0, 1, roty, rotz, R, omega, n)
```

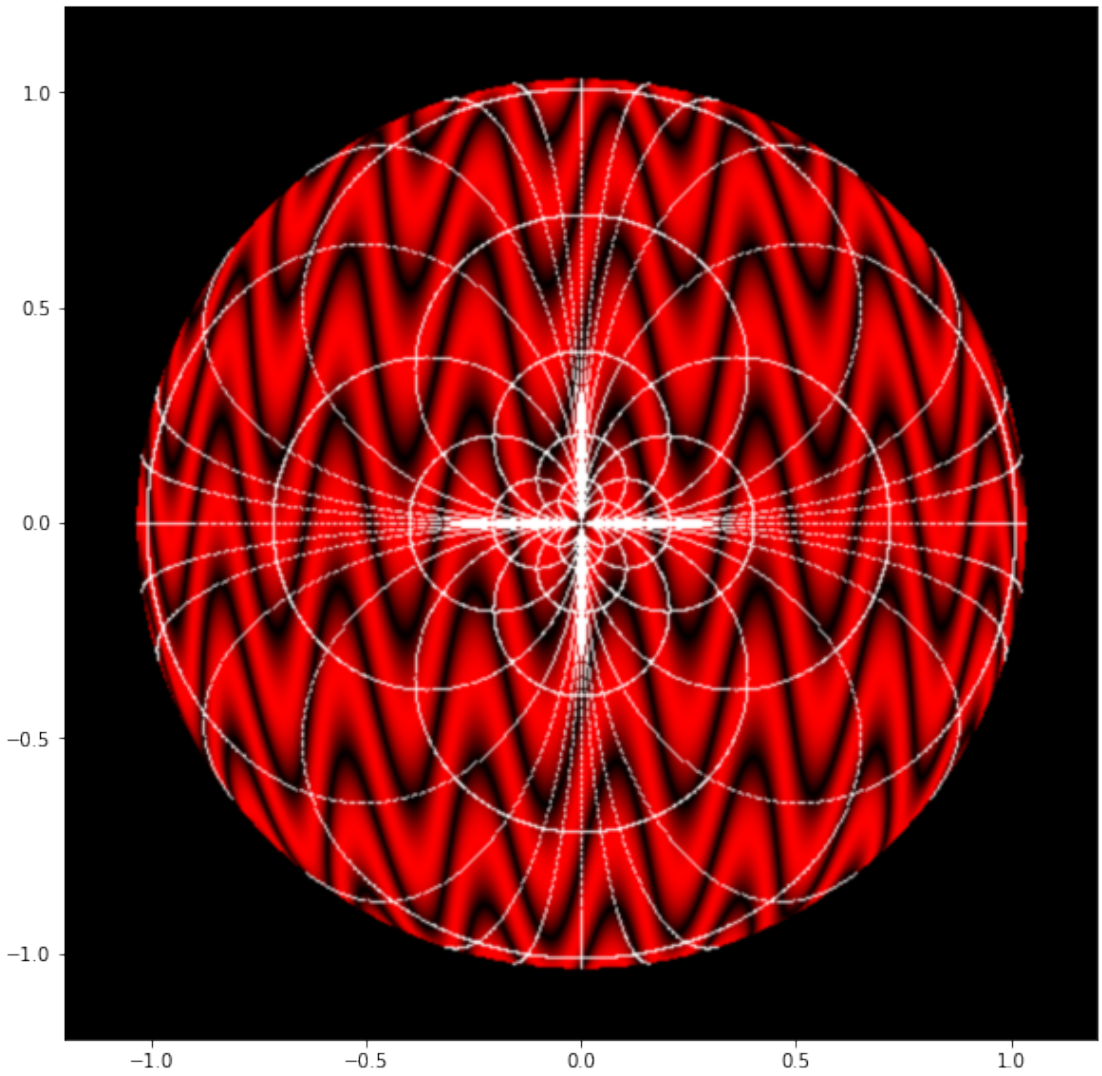
Et voici notre fonction ultime de tracé de $\mathcal{S}^2$.

```
In [29]: def plot_riemann(methode, roty=0, rotz=0, n=500, R=5, omega=20, axes=True):
             ws = [[(0, 0, 0) for i in range(n)] for j in range(n)]
             for i in range(n):
                 for j in range(n):
                     s = intersect_sphere(i, j, n, R, omega)
                     if s != []:
                         X0, Y0, Z0 = s[0]
                         X, Y, Z = rotation_Oy(X0, Y0, Z0, -roty * math.pi / 180)
                         X, Y, Z = rotation_Oz(X, Y, Z, -rotz * math.pi / 180)
                         ws[i][j] = methode(X, Y, Z, R)
             if axes: tracer_grille(ws, roty, rotz, R, omega, n)
             plt.imshow(ws, interpolation='bicubic', extent=[-1.2, 1.2, -1.2, 1.2])
```

Allons-y ! Voici la sphère vue du dessus. La grille de coordonnées est composée de deux familles de cercles passant par le pôle nord. Dans chacune des familles les cercles sont tangents entre-eux au pôle nord. Remarquons également (sans preuve ici) que les cercle d'une famille coupent les cercles de l'autre famlille à angle droit en **DEUX** points : le pôle nord, et un autre point. Si vous pensez que ces cercles sont des droites dans le plan complexe, vous obtenez le résultat suivant :

- Deux droites non parallèles se coupent en **DEUX** points, l'un des deux étant ∞.
- Deux droites parallèles se coupent en **UN** point, ∞, et sont tangentes en ce point.

```
In [32]: plot_riemann(methode_naive, roty=90, rotz=0, axes=True)
```



Notre méthode de tracé de la grille n'est pas idéale. Par exemple, certaines parties des "droites" sont en pointillés. Il faudrait beaucoup plus de travail pour obtenir une représentation plus esthétique. Nous nous contenterons de ce que nous avons fait :-).

2.3 3. Graphe d'une fonction de variable complexe

2.3.1 3.1 Rappels du premier notebook

La fonction ci-dessous renvoie $\frac{1}{2\pi}\arg z$ où l'argument de z est dans $[0, 2\pi[$. Elle renvoie donc un réel entre 0 et 1.

```
In [33]: def arg01(z):
             theta = cmath.phase(z) / (2 * math.pi)
             if theta < 0: return theta + 1
             else: return theta
```

J'ai parlé de la fonction `psi` dans le premier notebook sur les fonctions holomorphes. Elle prend un réel $x > 0$ et deux réels m et d, et renvoie un nombre entre 0 et 1 particulièrement bien adapté pour colorier le graphe d'une fonction complexe.

```
In [34]: def psi(x, m, d):
             if x == 0: return 0
             else:
                 return abs(math.sin(math.pi * math.log(x, 2) / m)) ** d
```

2.3.2 3.2 La méthode de tracé

Les sphères que nous avons tracées pour l'instant étaient toutes grises. L'idée, maintenant, est la suivante. Étant donnée une fonction $f : \mathbb{C} \to \mathbb{C}$, chaque point M de la sphère de Riemann est colorié en fonction de la valeur de $\varphi(M)$.

- On projette le point M sur le plan complexe étendu. Soit z son projeté.
- On calcule $w = f(z)$.
- À partir de w on calcule un triplet représentant une couleur dans le modèle HSV (voir le premier notebook).
- On renvoie ce triplet, ce sera la couleur choisie pour M.

```
In [35]: def methode_hsv(X, Y, Z, R, f, m, d):
             try:
                 z = projection_C(X, Y, Z, R)
                 w = f(z)
                 r = psi(abs(w), m, d)
                 ph = arg01(w)
                 return colorsys.hsv_to_rgb(ph, 1, r)
             except: return (1., 1., 1.)
```

À quoi sert le `try ... except` ? Les raisons pour que les calculs ci-dessous échouent sont nombreuses. L'une d'entre-elles, et non des moindres, est que nous calculons des quantités du genre $f(z)$ pour z voisin de l'infini. Essayez donc d'évaluer `math.exp(1000)`. Et pourtant, 1000 n'est pas un bien grand nombre.

2.3.3 3.3 La fonction de tracé

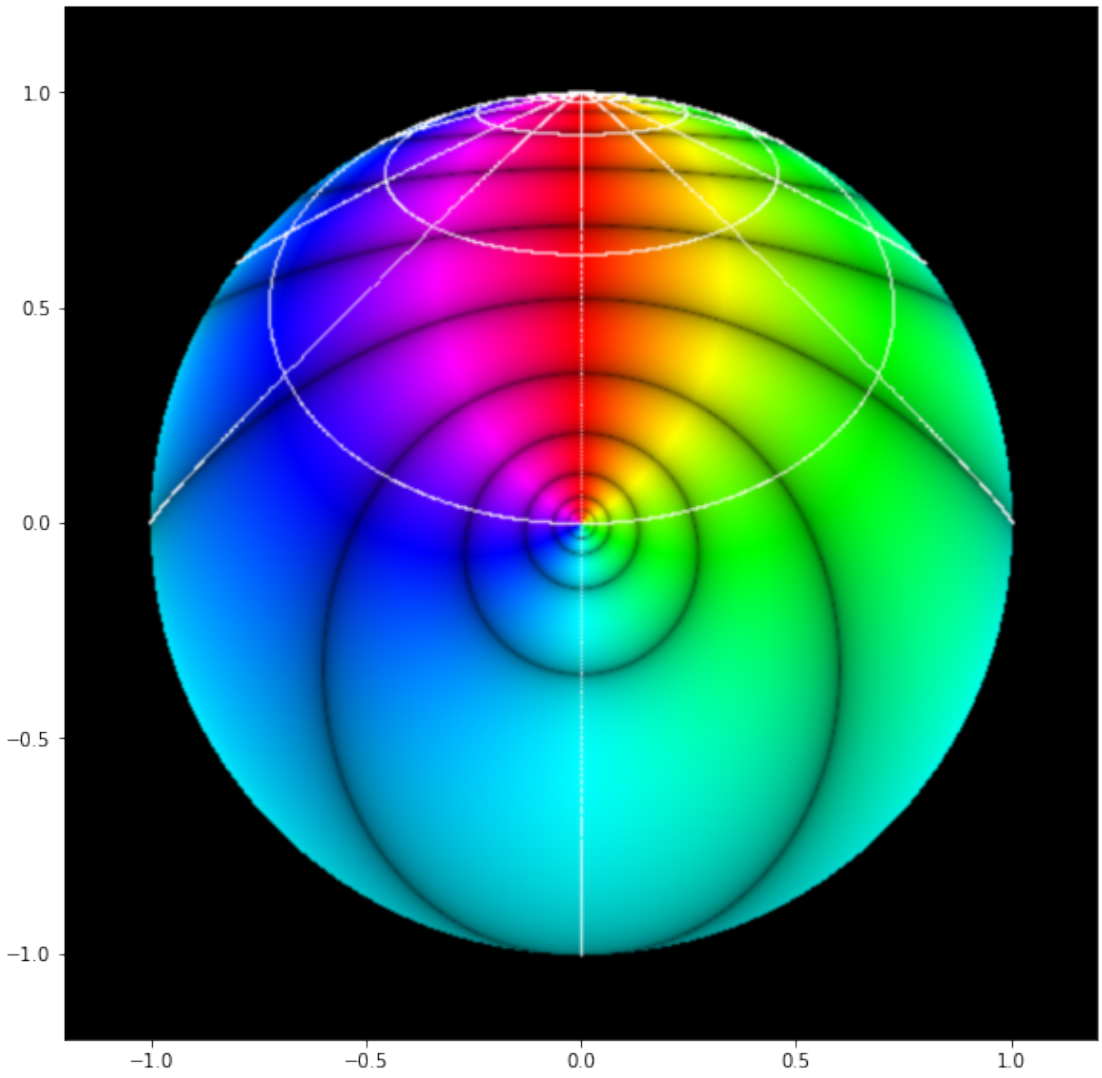
La fonction de tracé d'une fonction $f : \mathbb{C} \to \mathbb{C}$ est maintenant immédiate.

```
In [36]: def plot_complex(f, roty=0, rotz=0, m=1, d=0.2, n=500, R=5, omega=20, axes=True):
             plot_riemann(lambda X, Y, Z, R:methode_hsv(X, Y, Z, R, f, m, d), roty, rotz, n, R,
```

2.3.4 3.4 Un premier test

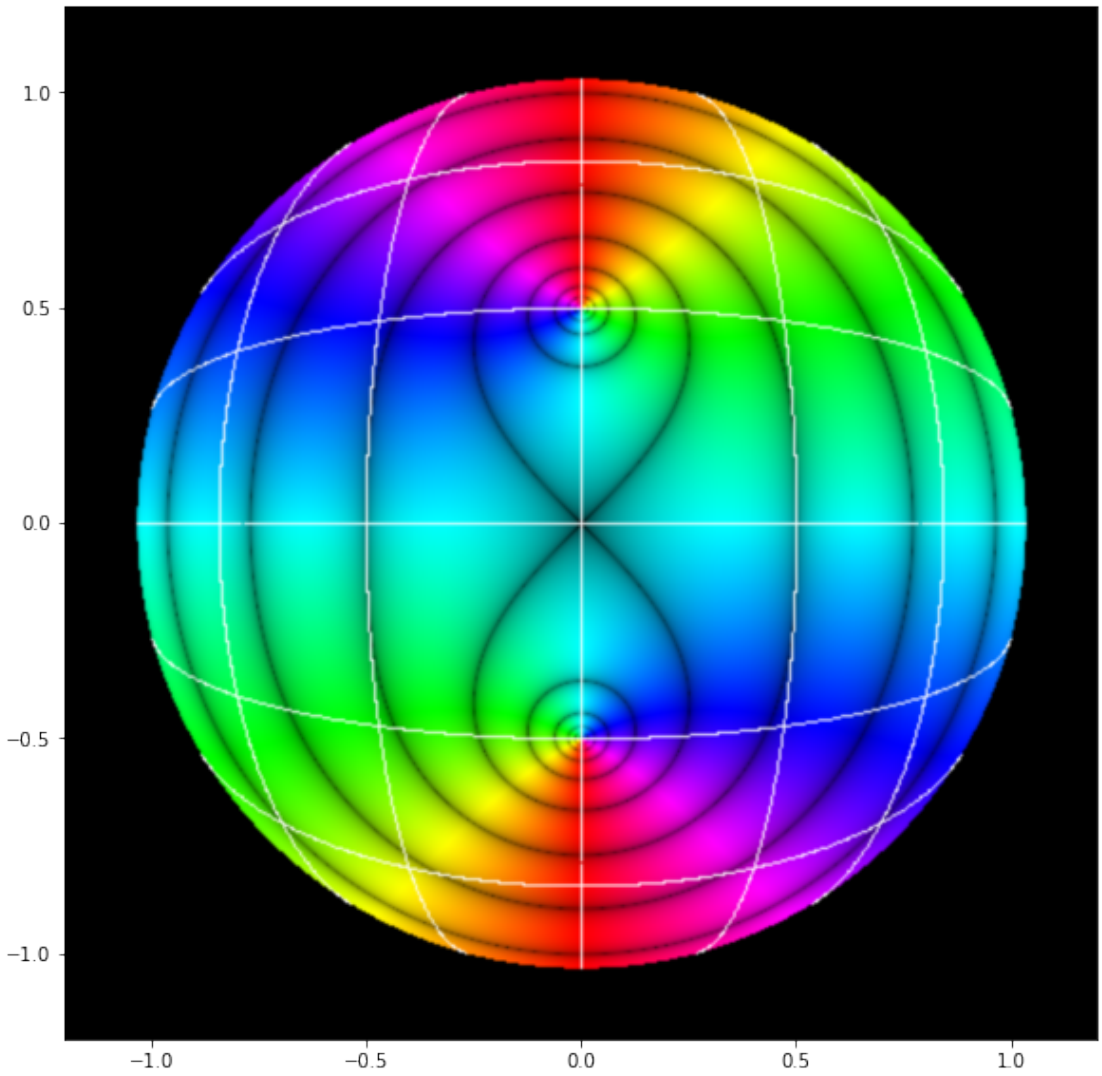
Traçons la fonction $f : z \mapsto z^2 - 1$.

```
In [37]: plot_complex(lambda z:z ** 2 - 1, roty=0, rotz=0, R=1)
```



La sphère de Riemann est coloriée par des bandes horizontales sur chacune desquelles ont lieu des cycle de couleurs. Peut-on voir les deux racines de f en même temps ? Oui, il suffit de prendre R un peu plus grand et effectuer une rotation par rapport à Oy de $-\frac{\pi}{2}$.

```
In [38]: plot_complex(lambda z:z ** 2 - 1, roty=-90, rotz=0, R=5, omega=20)
```

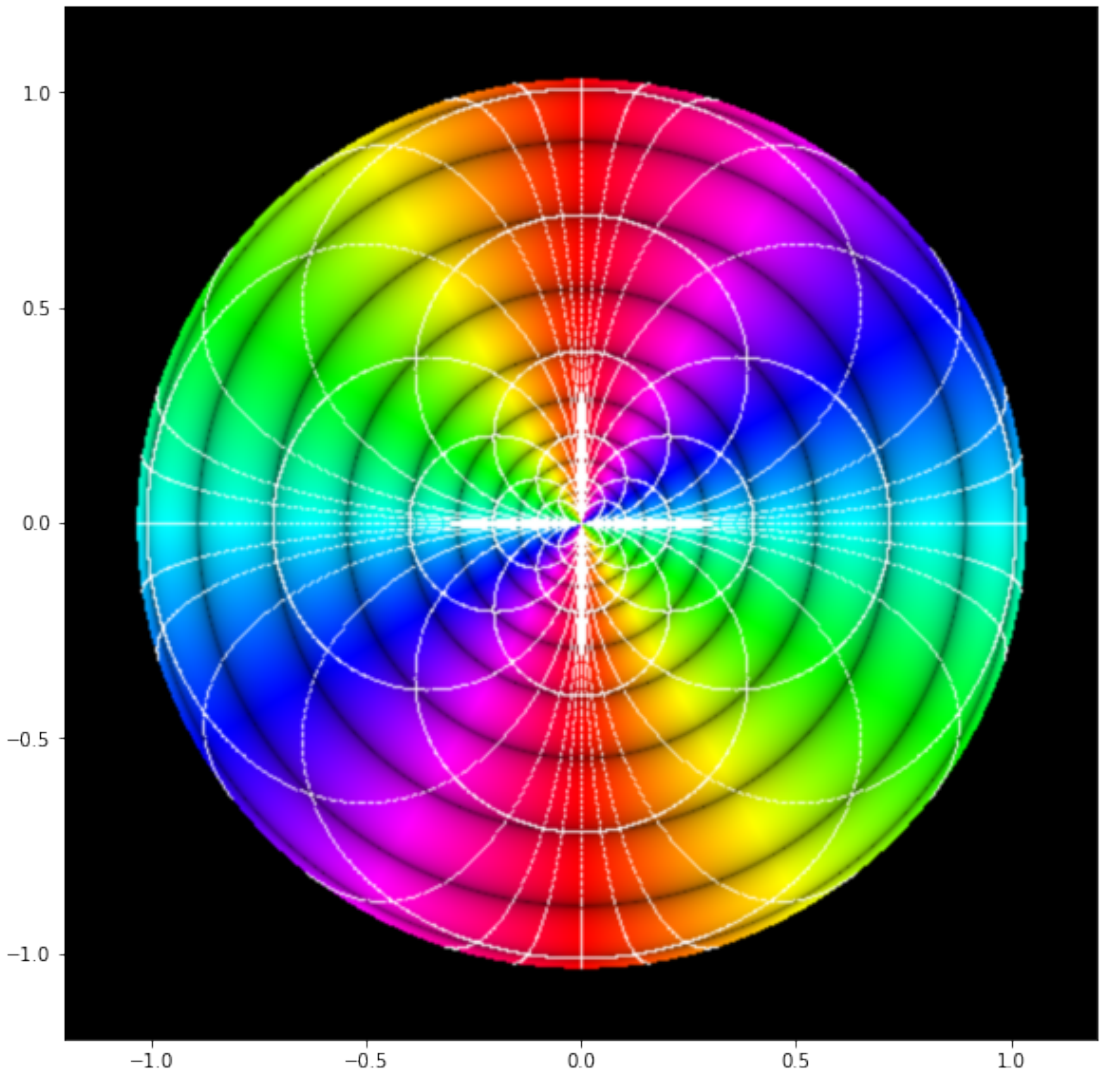


Ah mais oui, c'est ce que nous avions vu dans le premier notebook !
Ah mais non, les couleurs tournent dans le mauvais sens ???
Réfléchissez un peu : la projection de la sphère de Riemann sur le plan complexe renverse les sens de rotation
Conséquence :

- Racine : les couleurs tournent dans le sens trigonométrique inverse.
- Pôle : les couleurs tournent dans le sens trigonométrique direct.

Que se passe -t-il en ∞ ? Tournons de $\frac{\pi}{2}$ autour de Oy, l'infini sera alors juste devant nos yeux.

```
In [39]: plot_complex(lambda z:z ** 2 - 1, roty=90, rotz=0, R=5, omega=20)
```



Deux cycles de couleurs, parcourus dans le sens trigonométrique ? Oserons-nous ?
Propriété : ∞ est un pôle double de f.

2.3.5 3.5 L'infini comme pôle ou racine

Définition : On dit que ∞ est un pôle de f lorsque 0 est un pôle de $g : z \mapsto f(\frac{1}{z})$. L'ordre de ∞ comme pôle de f est alors l'ordre de 0 en tant que pôle de g. On définit de même la notion de racine en ∞ et l'ordre de multiplicité de la racine.

Prenons par exemple $f(z) = z^2 - 1$, prolongée à $\hat{\mathbb{C}}$ par $f(\infty) = \infty$. On a

$$g(z) = \frac{1}{z^2} - 1$$

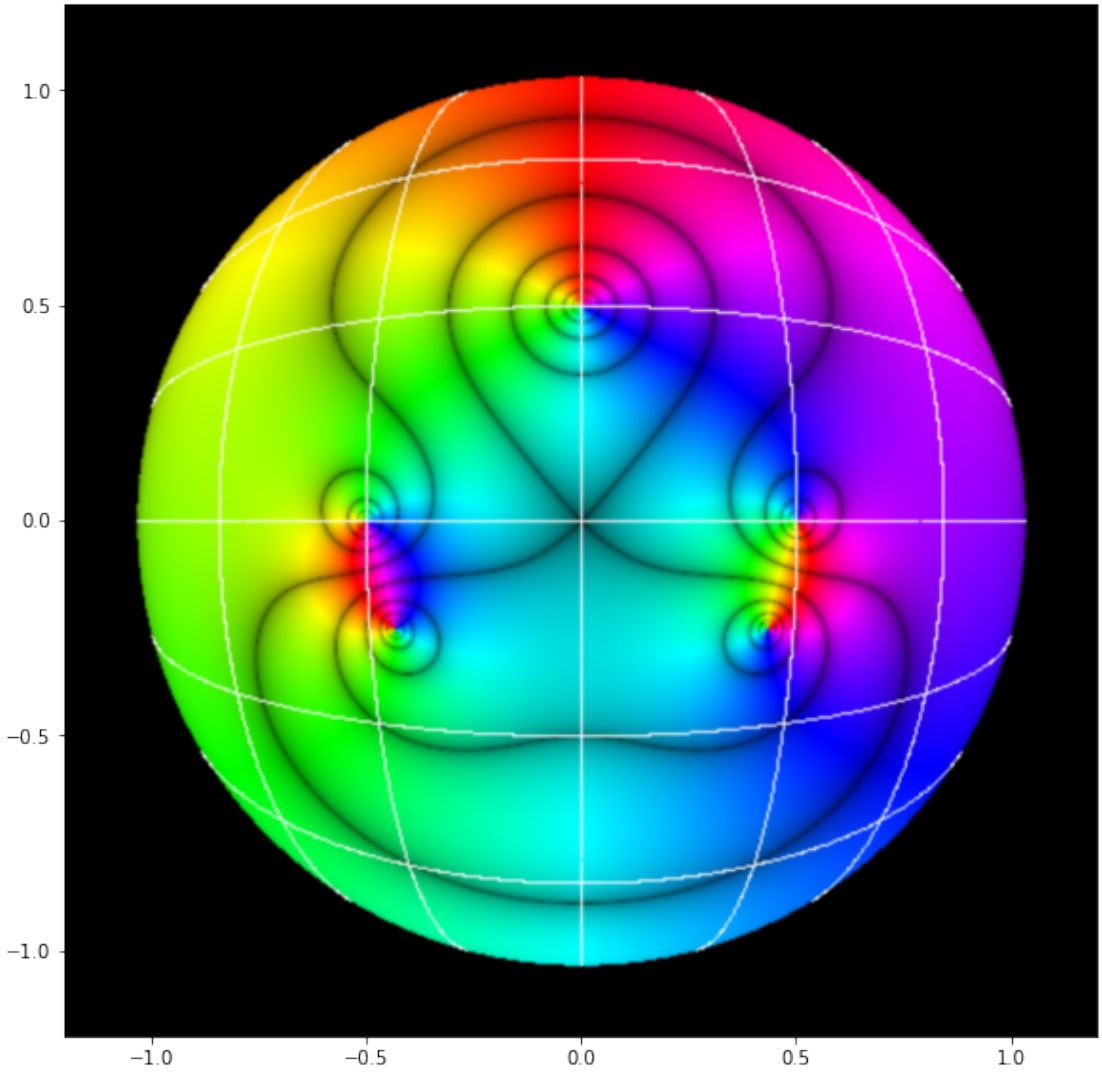
et 0 est un pôle d'ordre 2 de g.

2.3.6 3.6 Un exemple où l'infini est une racine

Prenons $f(z) = \frac{1}{z^3-1}$.

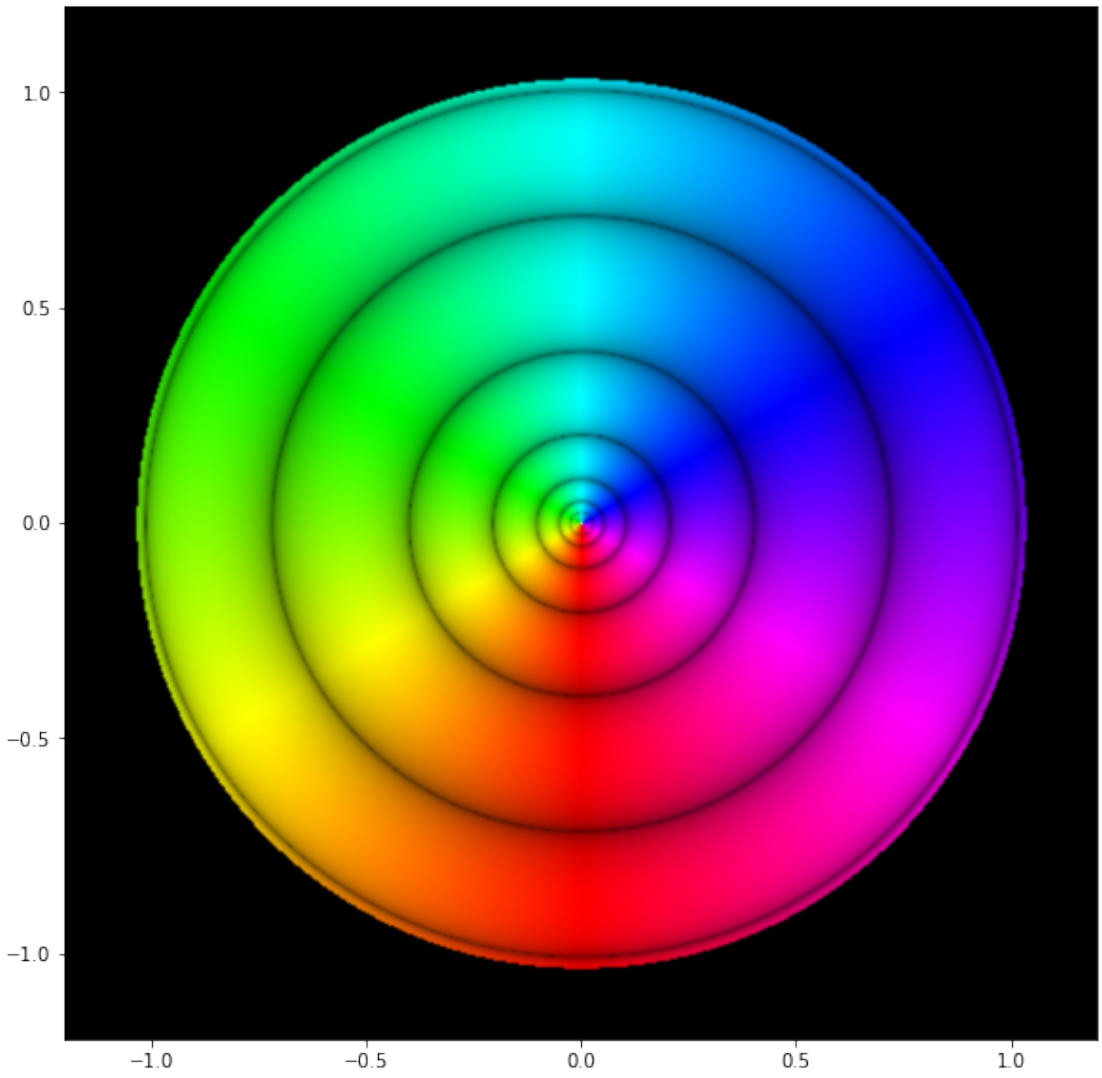
1, j et j^2 sont maintenant trois pôles de f d'ordre 1 : notez le cycle de couleurs dans le sens trigonométrique autour de ces deux pôles. Et $-i$ et i sont deux racines d'ordre 1.

```
In [40]: plot_complex(lambda z:(z ** 2 + 1) / (z ** 3 - 1), roty=-90, rotz=0)
```



Et l'infini ? Eh bien c'est une racine d'ordre 1.

```
In [41]: plot_complex(lambda z:(z ** 2 + 1) / (z ** 3 - 1), roty=90, rotz=0, axes=False)
```



Une preuve ? On a

$$f(\frac{1}{z}) = \frac{\frac{1}{z^2}+1}{\frac{1}{z^3}-1} = \frac{z(1+z^2)}{1-z^3}$$

et 0 est bien racine simple de $z \mapsto f(\frac{1}{z})$.

2.4 4. Et maintenant ?

Je ne vais pas reprendre tous les exemples de fonctions que nous avons tracées dans les trois premiers notebooks :

- polynômes
- fractions rationnelles
- exponentielle

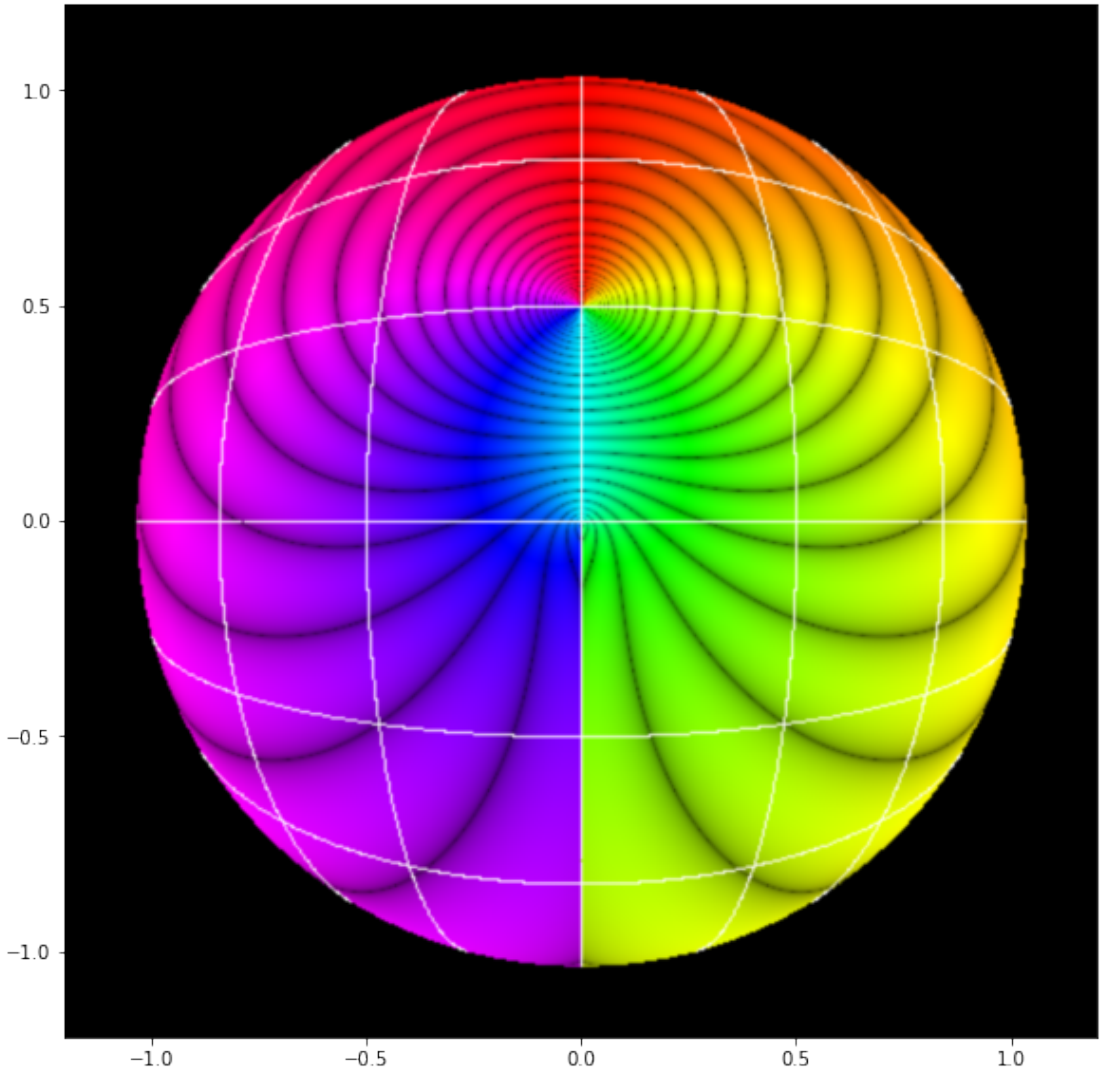
- logarithmes
- puissances
- fonctions trigonométriques

Faites-le. Et regardez l'infini :-).
Histoire de vous donner envie :

2.4.1 4.1 Logarithmes

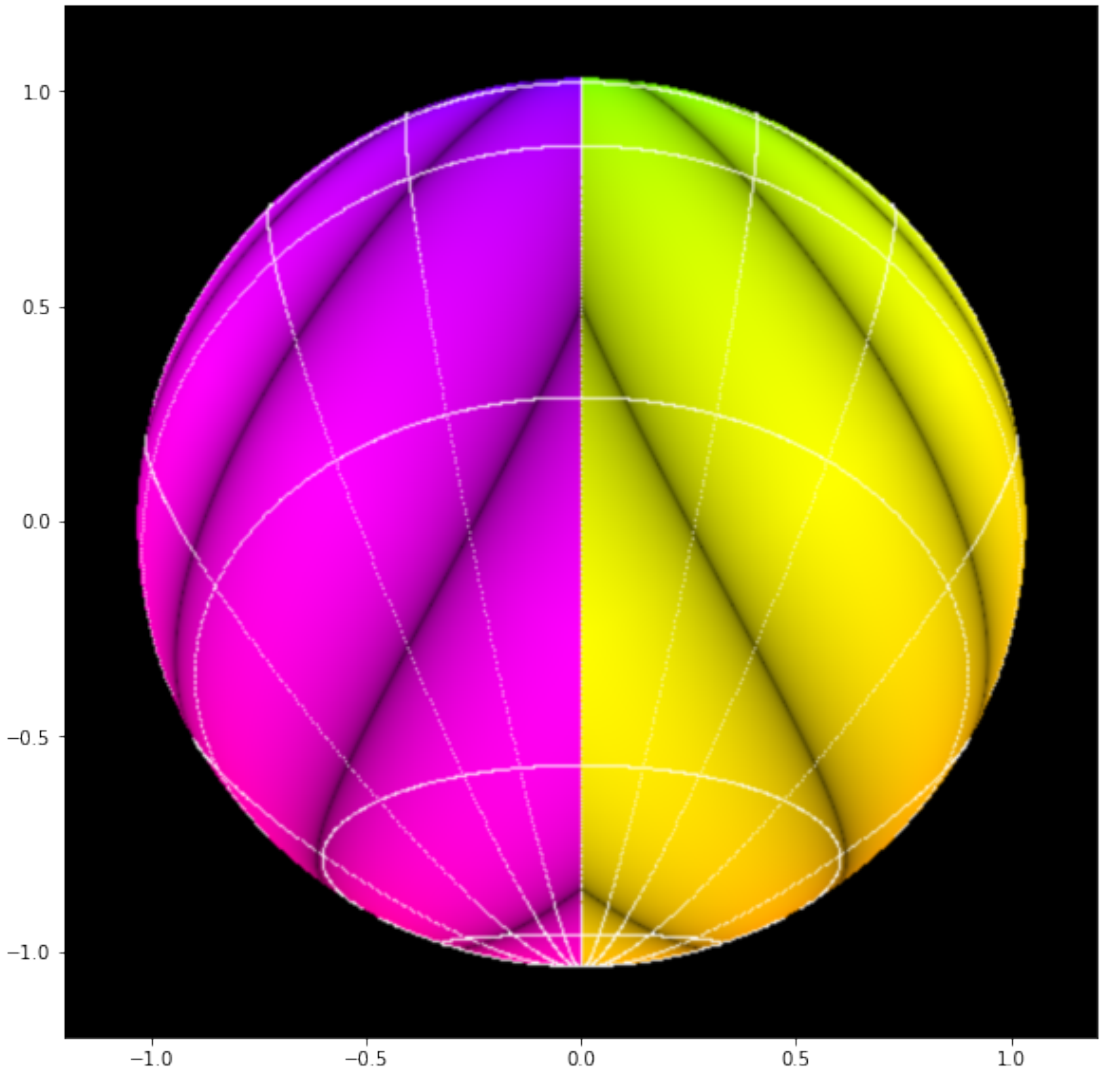
Voici une vue "standard" du logarithme, avec sa racine simple 1, sa singularité non isolée 0 et une partie de la coupure $\mathbb{R}_-$.

```
In [42]: plot_complex(lambda z:cmath.log(z), roty=-90, rotz=0, m=0.25, axes=True)
```



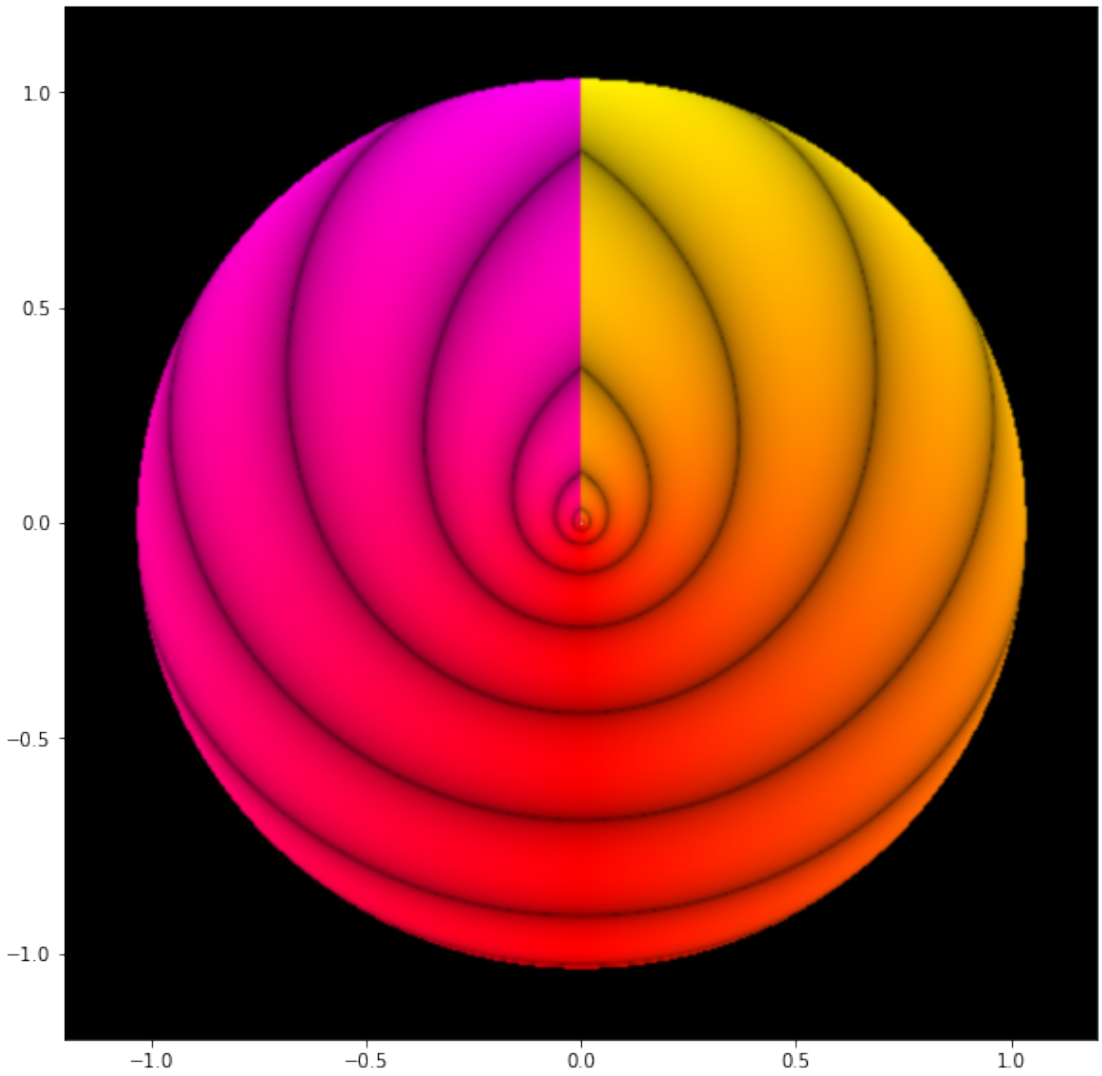
Voici une vue centrée sur (presque tous) les réels négatifs. Eh oui, malgré toute notre bonne volonté il nous est très difficile de voir en même temps le pôle sud et le pôle nord de $\mathcal{S}^2$.

```
In [43]: plot_complex(lambda z:cmath.log(z), roty=180, rotz=0, m=0.25, axes=True)
```



Et enfin, une vue centrée sur ∞. Mettez `axes` à `True` si vous désirez l'affichage de la grille.

```
In [44]: plot_complex(lambda z:cmath.log(z), roty=90, rotz=0, m=0.25, axes=False)
```

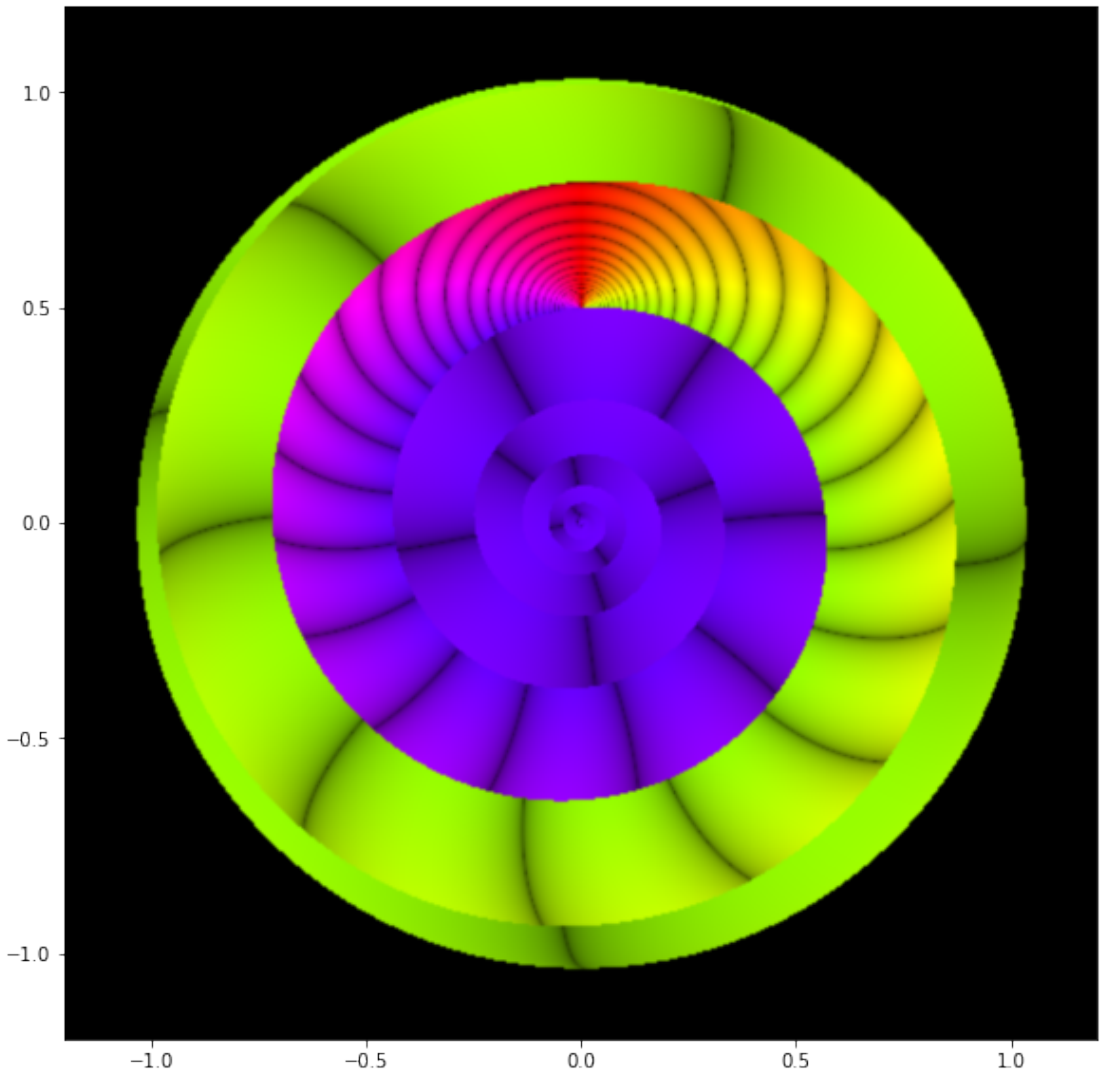


Le logarithme est holomorphe sur $\mathcal{S}^2$ privée d'un demi-cercle, allant du pôle sud (0) au pôle nord (∞). Ceci explique les remarques que j'avais faites en leur temps sur l'existence de déterminations du logarithme. C'est à ce moment que j'avais parlé de logarithmes fous, définis sur $\mathbb{C}$ privé d'une spirale logarithmique. Eh bien cette spirale joint 0 à ∞. Dessinons-le, ce logarithme bizarre.

```
In [45]: def logfou(z, a):
             r = abs(z)
             theta = cmath.phase(z)
             k = math.floor((math.log(r, a)- theta) / (2 * math.pi))
             return math.log(r) + 1j * (theta + 2 * k * math.pi)
```

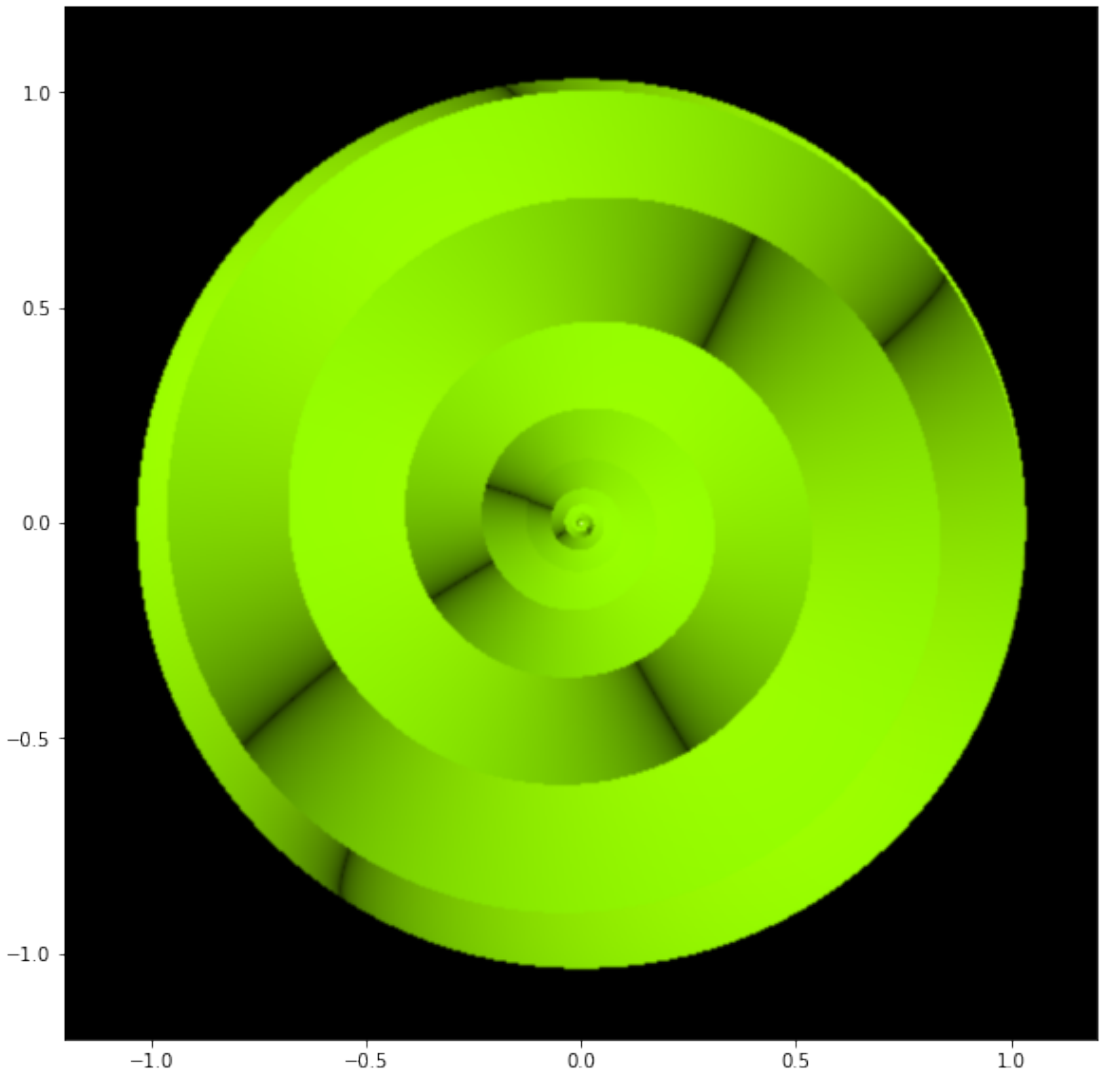
Voici le logarithme fou autour de 0.

```
In [46]: plot_complex(lambda z:logfou(z, 1.1), roty=-90, rotz=0, m=0.25, axes=False)
```



Et le voici autour de ∞ !

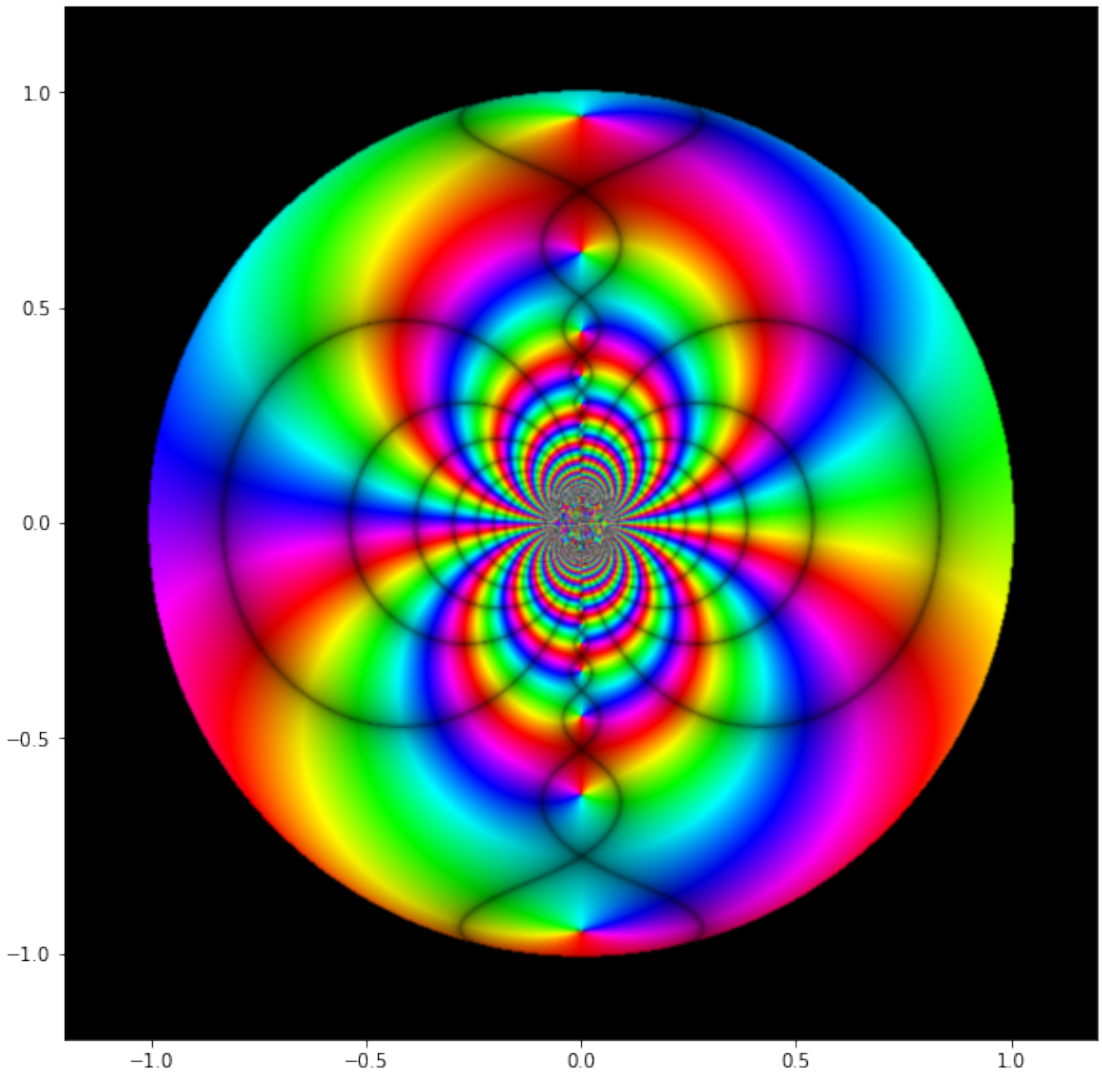
```
In [47]: plot_complex(lambda z:logfou(z, 1.1), roty=90, rotz=0, m=0.2, axes=False, d=0.25)
```



2.4.2 4.2 Sinus

Terminons en beauté par un zoom à l'infini de la fonction sinus : ∞ est une singularité essentielle de sin.

```
In [48]: plot_complex(lambda z: cmath.sin(z), roty=90, rotz=0, m=5, R=2, d=0.2, axes=False, n=60
```



2.4.3 4.3 À vous de jouer . . .

In [ ]: