Lambert

December 17, 2018

1 $x^{x^{x^{\cdots}}} = ?$

Marc Lorenzi
15 décembre 2018

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

In [2]: plt.rcParams['figure.figsize'] = (12, 6)
```

1.1 1. Exponentielle infinie

1.1.1 1.1 Introduction

Soit $x \in \mathbb{R}_+^*$. Quel sens donner à $\varphi(x) = x^{x^{x^{\cdots}}}$? Des points de suspensions avec rien au bout nous suggèrent une limite de suite.

Posons $x^{[0]} = 1$ et, pour tout $n \in \mathbb{N}$, $x^{[n+1]} = x^{x^{[n]}}$. On a $x^{[1]} = x$, $x^{[2]} = x^x$, $x^{[3]} = x^{x^x}$. Notons $x^{[\infty]}$ la limite, si elle existe, de $x^{[n]}$ lorsque n tend vers l'infini.

On a $x^{[n]} \to x^{[\infty]}$. Mais $x^{x^{[n]}} = x^{[n+1]} \to x^{[\infty]}$. On en déduit que

$$x^{x^{[\infty]}} = x^{[\infty]}$$

Posons dorénavant $\varphi(x) = x^{[\infty]}$. Nous venons donc de "définir" une fonction $\varphi : \mathcal{D} \to \mathbb{R}$ par

$$x^{\varphi(x)} = \varphi(x)$$

où $\mathcal{D}$ est l'ensemble des réels $x > 0$ tel que notre suite converge. Pourvu que $\mathcal{D}$ soit non vide, sinon ce notebook risque de tourner court :-).

Exercice : Montrer que $1 \in \mathcal{D}$ et que $\varphi(1) = 1$.

Exercice : Montrer que pour tout $n \in \mathbb{N}$, $0^{[2n]} = 1$ et $0^{[2n+1]} = 0$. En déduire que $0 \notin \mathcal{D}$.

Exercice : Montrer que pour tout $x > 2$, $x \notin \mathcal{D}$. Ainsi, $\mathcal{D} \subset]0, 2[$.

Voici quelques questions auxquelles nous allons répondre :

- Que vaut précisément $\mathcal{D}$?
- Comment calculer efficacement pour $x \in \mathcal{D}$ une valeur approchée de $\varphi(x)$?
- Y-a-t-il des réels $x \in \mathcal{D}$ pour lesquels on peut calculer explicitement $\varphi(x)$?

Nous verrons que la fonction φ est étroitement liée à une autre fonction, la fonction W de Lambert.

1.1.2 1.2 Une valeur approchée

Soit $x > 0$. Le réel $\varphi(x)$ (s'il existe) est un point fixe de la fonction $t \mapsto x^t$. Considérer une suite (u_n) définie par $u_{n+1} = x^{u_n}$ devrait peut-être permettre d'approcher ce point fixe. Une simple boucle `while` ferait-elle l'affaire ?

La fonction `inf_exp` ci-dessous prend un réel x en paramètre. Elle renvoie une valeur que l'on espère approchée de $\varphi(x)$. Le second paramètre, `dbg` permet l'affichage du nombre d'itérations lorsqu'il est mis à `True`. Le troisième paramètre, `tol`, permet de régler la précision du résultat.

```
In [3]: def inf_exp(x, dbg=False, tol=1e-15):
            cnt = 0
            u, v = 1, x
            while abs(v - u) > tol and cnt <= 1e6:
                cnt += 1
                u, v = v, x ** v
            if dbg: print(cnt)
            return v

In [4]: inf_exp(1)

Out[4]: 1

In [5]: y = inf_exp(1.2, True)
        print(y)
        print(1.2 ** y)

23
1.2577345413765262
1.2577345413765264
```

1.1.3 1.3 Quelques tentatives plus ou moins heureuses

Tentative 1 ... On a $\sqrt{2}^2 = 2$. Se pourrait-il donc que $\sqrt{2}^{[\infty]} = 2$?

```
In [6]: x = math.sqrt(2)
        y = inf_exp(x, True)
        print(y)
        print(x ** y)

90
1.999999999999999
1.9999999999999996
```

Cela est très prometteur. Mais bien entendu l'approximation précédente ne le **montre** pas. Elle nous le **suggère**.

Tentative 2 ... On a $\left(e^{\frac{1}{e}}\right)^e = e$. Se pourrait-il que $\left(e^{\frac{1}{e}}\right)^{[\infty]} = e$?

```
In [7]: x = math.e ** (1 / math.e)
        y = inf_exp(x, True, tol=1e-10)
        print(y)
        print(x ** y)

233158
2.7182585121475733
2.7182585122475715
```

Cela se pourrait. Remarquez la convergence terriblement lente. Il va falloir améliorer cela. Tentative 3 ... On a $(e^{-e})^{\frac{1}{e}} = e^{-1} = \frac{1}{e}$. Se pourrait-il que $(e^{-e})^{[\infty]} = \frac{1}{e}$?

```
In [8]: x = math.e ** (- math.e)
        y = inf_exp(x, True, tol=1e-5)
        print(y)
        print(x ** y)

1000001
0.3687811044347396
0.3669788819829381
```

Argh, le million d'itérations a été dépassé ! Mais les valeurs numériques renvoyées sont encourageantes. Relançons avec une tolérance de 10^{-2}.

```
In [9]: x = math.e ** (- math.e)
        y = inf_exp(x, dbg=True, tol=1e-2)
        print(y)
        print(x ** y)

32472
0.362896496593683
0.3728962857462967
```

Ultime tentative ... Et si nous prenions x **complexe** ? Que vaut $i^{i^{i^{\cdots}}}$?

```
In [10]: inf_exp(1j, True)

303


Out[10]: (0.4382829367270322+0.36059247187138604j)
```

Bigre, ça a l'air de converger. Notre tâche va donc être complexe :-). Quels sont les **nombres complexes** z pour lesquels $z^{[\infty]}$ existe ?

1.1.4 1.4 Les résultats sur $\mathbb{R}$

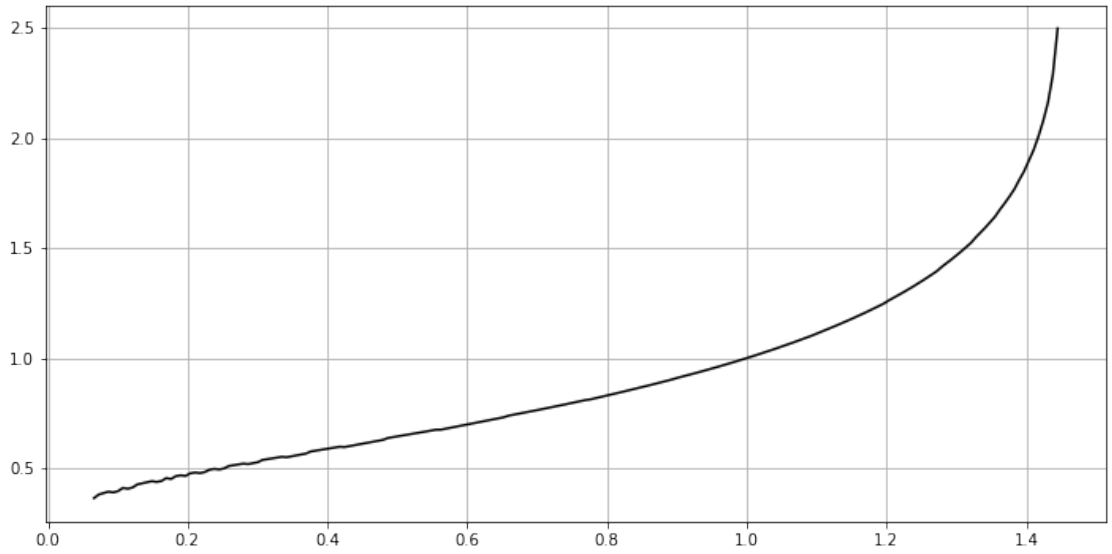
Proposition : L'ensemble de définition de φ est $\mathcal{D} = [e^{-e}, e^{1/e}]$.

Démonstration : La démonstration est un peu longue, quoique sans difficulté particulière. Nous ne la ferons pas ici. Pour être honnêtes, $D = [e^{-e}, e^{1/e}] \cup \{-1\}$.

Contentons nous de tracer la courbe de la fonction φ.

```
In [11]: def subdivision(a, b, n):
             d = (b - a) / n
             return [a + k * d for k in range(n + 1)]
```

```
In [12]: xs = subdivision(math.e ** (-math.e), math.e ** (1 / math.e), 200)
         ys = [inf_exp(x, False, 1e-2) for x in xs]
         plt.plot(xs, ys, 'k')
         plt.grid()
         plt.show()
```



Remarquez les zigzags au voisinage de e^{-e} ... ceci est dû à la mauvaise convergence des suites au voisinage de ce point. Il va nous falloir remédier à cela.

Nous allons voir maintenant que φ est liée à une autre fonction, la **fonction de Lambert**.

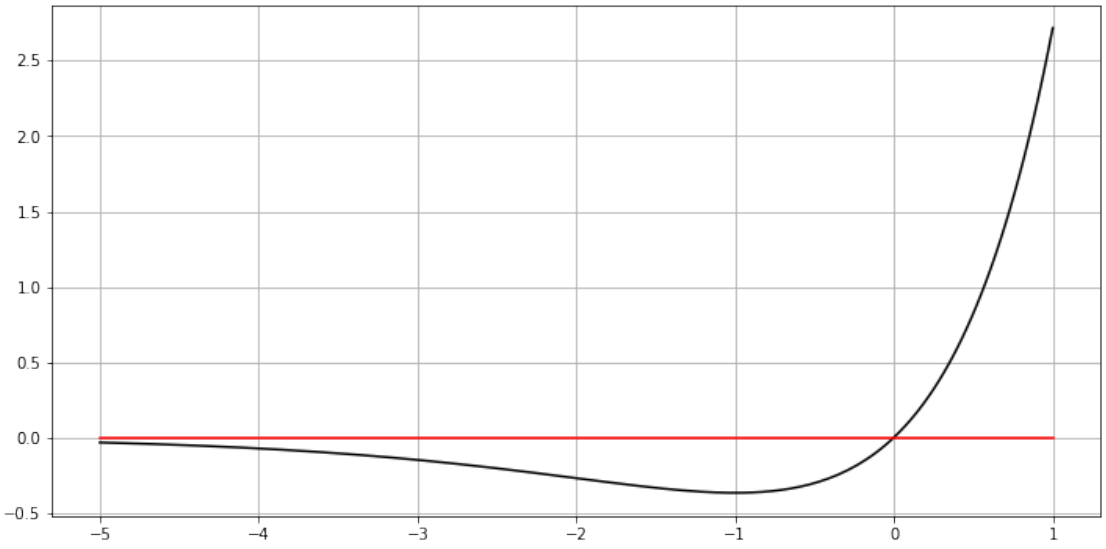
1.2 2. La fonction W de Lambert

1.2.1 2.1 C'est quoi ?

La fonction de Lambert est la fonction W "définie" par $W(xe^x) = x$. Disons pour faire vite que W est la "réciproque" de la fonction $f : x \mapsto xe^x$. Euh, oui, sauf que f n'est pas bijective. Voici le graphe de f.

```
In [13]: xs = subdivision(-5, 1, 200)
         ys = [x * math.exp(x) for x in xs]
```

```
zs = [0 for x in xs]
plt.plot(xs, ys, 'k')
plt.plot(xs, zs, 'r')
plt.grid()
```



Une étude rapide de f montre que f est une bijection de $]-\infty,-1]$ sur $[-\frac{1}{e},0[$, et aussi une bijection de $[-1,+\infty[$ sur $[-\frac{1}{e},+\infty[$.

Notons $f_1 :]-\infty,-1] \to [-\frac{1}{e},0[$ et $f_2 : [-1,+\infty[\to [-\frac{1}{e},+\infty[$ les bijections induites par f. Nous aurons donc deux fonctions de lambert, réciproques de f_1 et f_2 :

Proposition : - $W_1 : [-\frac{1}{e},0[\to]-\infty,-1]$ est une bijection continue strictement décroissante. La fonction W_1 est de classe $\mathcal{C}^\infty$ sur $[-\frac{1}{e},0[$. - $W_2 : [-\frac{1}{e},+\infty[\to [-1,+\infty[$ est une bijection continue strictement croissante. La fonction W_2 est de classe $\mathcal{C}^\infty$ sur $[-\frac{1}{e},+\infty[$.

Démonstration : Il suffit d'appliquer les théorèmes généraux sur les fonctions strictement monotones. Que vaut la dérivée de W_2 ? Pour tout $x \in [-\frac{1}{e},+\infty[$, on a $W_2'(x) = (f_2^{-1})'(x) = \frac{1}{f_2' \circ f_2^{-1}(x)}$. Or, $f_2'(t) = (t+1)e^t = e^t + f_2(t)$. Donc,

$$W_2'(x) = \frac{1}{x + \exp(W_2(x))}$$

Quelques valeurs intéressantes prises par la fonction W_2 :

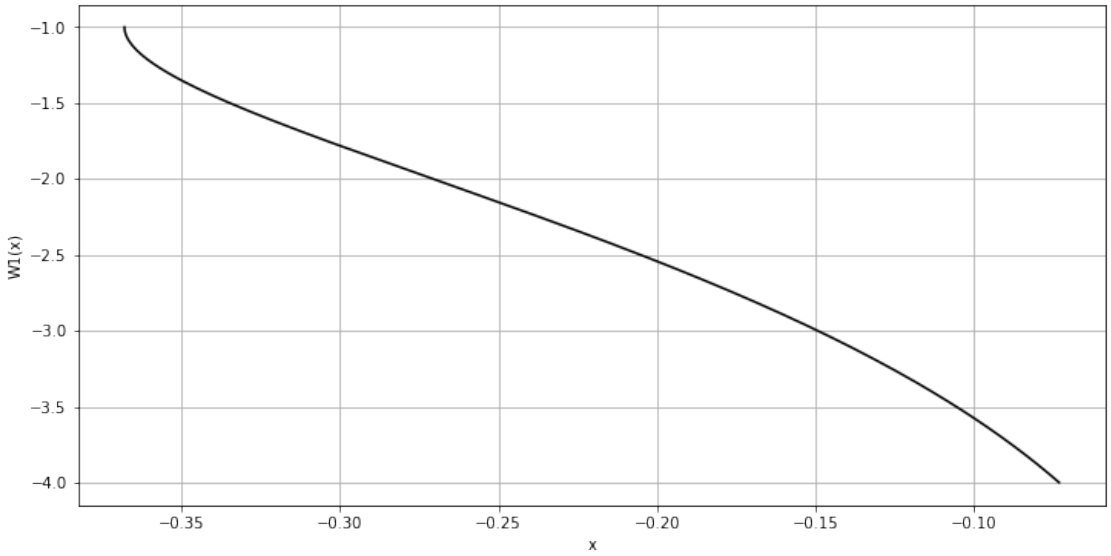
- $W_2(0) = 0$.
- $W_2(-\frac{1}{e}) = -1$.
- $W_2(e) = 1$.

Exercice :

- Que valent $W_2'(0)$? $W_2'(e)$?
- Quelle est la limite de $W_2'(x)$ lorsque x tend vers $-\frac{1}{e}$?
- Quelle est la limite de $W_2'(x)$ lorsque x tend vers $+\infty$?

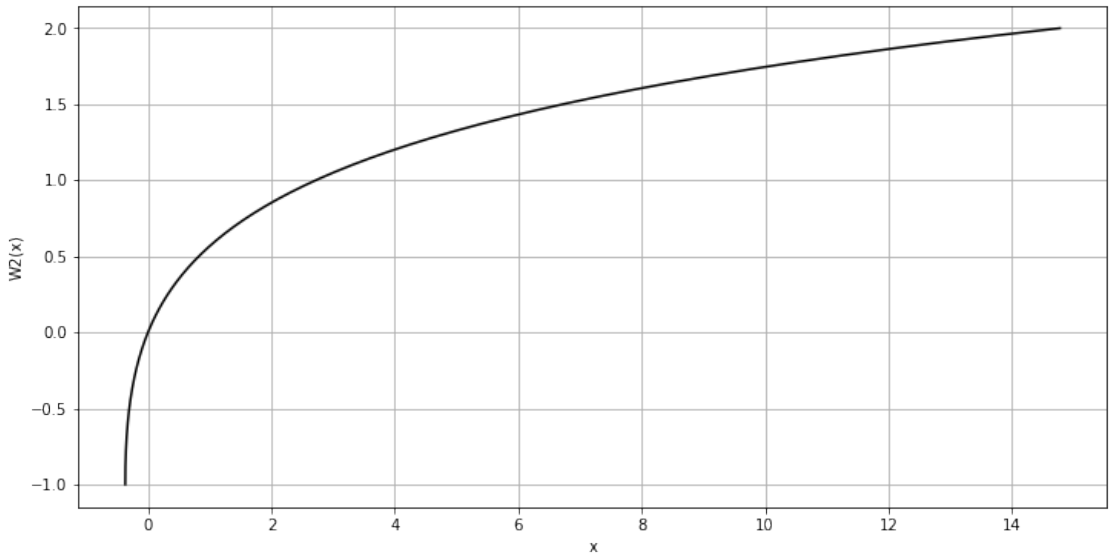
Voici le graphe de W_1 obtenu en "trichant". Il n'y a qu'à tracer f_1 en échangeant abscisses et ordonnées.

```
In [14]: xs = subdivision(-4, -1, 200)
         ys = [x * math.exp(x) for x in xs]
         plt.plot(ys, xs, 'k')
         plt.xlabel('x')
         plt.ylabel('W1(x)')
         plt.grid()
```



Et voici le graphe de W_2, obtenu par la même technique.

```
In [15]: xs = subdivision(-1, 2, 200)
         ys = [x * math.exp(x) for x in xs]
         plt.plot(ys, xs, 'k')
         plt.xlabel('x')
         plt.ylabel('W2(x)')
         plt.grid()
```



Évidemment, nous aimerions bien pouvoir calculer $W_i(x)$, $i = 1, 2$, ou du moins une approximation. Patience, cela va venir.

1.2.2 2.2 Le rapport avec les puissances infinies

Quel est le rapport entre les puissances infinies et la fonctiion W ?

Rappelons la notation $\varphi(x) = x^{[\infty]}$. Soit $x \in [e^{-e}, e^{1/e}]$. On a $x^{\varphi(x)} = \varphi(x)$. Passons aux logarithmes pour en déduire $\varphi(x) \ln x = \ln \varphi(x)$. Si nous posons $\alpha = \ln \varphi(x)$, il vient $e^{\alpha} \ln x = \alpha$, ou encore $-\ln x = -\alpha e^{-\alpha}$. Ainsi, $\alpha = -W(-\ln x)$, d'où

$$\varphi(x) = e^{-W(-\ln x)}$$

Certes, mais de quel W parlons-nous ? S'agit-il de W_1 ou de W_2 ? Comme $e^{-e} \leq x \leq e^{1/e}$, on a $-e \leq \ln x \leq \frac{1}{e}$, donc $-\frac{1}{e} \leq -\ln x \leq e$. Si $-\ln x \geq 0$, c'est à dire $x \leq 1$, aucun doute nous parlons de W_2. Et sinon ? Dans ce cas, $-\ln x$ appartient aux ensembles de définition de W_1 et W_2 et nous voilà bien embêtés. Nous admettrons que la "bonne" fonction est encore W_2.

Proposition :

- $\varphi(e^{-e}) = \frac{1}{e}$
- $\varphi(e^{1/e}) = e$
- $\varphi(\sqrt{2}) = 2$

Démonstration : On a $W_2(-\ln e^{-e}) = W_2(e) = 1$. Donc $\varphi(e^{-e}) = \exp(W_2(-\ln e^{-e})) = e^{-1} = \frac{1}{e}$. Je vous laisse calculer les deux autres valeurs.

Proposition : la fonction φ est une bijection continue strictement croissante de $[e^{-e}, e^{1/e}]$ sur $[\frac{1}{e}, e]$. Elle est de classe $\mathcal{C}^{\infty}$ sur $[e^{-e}, e^{1/e}[$ et dans cet intervalle on a

$$\varphi'(x) = \frac{\varphi(x)^2}{x(1 - \varphi(x) \ln x)}$$

Démonstration : On a $\varphi(x) = e^{-W_2(-\ln x)}$. Il n'y a qu'à utiliser les résultats montrés pour W_2.

Exemples :

- $\varphi'(e^{-e}) = \frac{1}{2}e^{e-2}$.
- $\varphi'(1) = 1$.
- Quand $x \to e^{1/e}$, $\varphi(x) \to \varphi(e^{1/e}) = e$. On voit donc que $\varphi'(x)$ tend vers $+\infty$. Ainsi, φ n'est pas dérivable en $e^{1/e}$ et la courbe de φ a une tangente verticale en ce point.

```
In [16]: math.e ** (math.e - 2) / 2
```

```
Out[16]: 1.0254531863462504
```

Si nous disposons d'un algorithme efficace du calcul de W_i, alors nous pourrons calculer efficacement $x^{[\infty]}$ pour tout x. Nous allons tout d'abord implémenter la méthode de Newton, qui donne d'excellents résultats pour le calcul de W, sauf au voisinage de $-\frac{1}{e}$. Puis nous verrons une méthode encore plus efficace, la méthode de Halley, une sorte de méthode de Newton de Newton.

1.2.3 2.3 Calcul de W par la méthode de Newton

Soit $f : I \to \mathbb{R}$ une fonction dérivable définie sur un intervalle de $\mathbb{R}$. Il s'agit de trouver une racine de f. La méthode de Newton consiste à considérer la suite (u_n) définie par $u_0 \in I$ et, pour tout entier n, $u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}$. Sous certaines conditions que je n'expliciterai pas ici (**voir le notebook sur les racines des équations !**), la suite (u_n) converge vers une racine de f sur l'intervalle I.

Soit $x \in \mathbb{R}$. Notre problème ici est de trouver un réel w tel que $we^w = x$. On considère donc la fonction f définie par $f(w) = we^w - x$. On a $w - \frac{f(w)}{f'(w)} = \frac{w^2e^w+x}{(w+1)e^w}$, après quelques simplifications.

La fonction `flamb` ci-dessous s'impose donc.

```
In [17]: def flamb(w, x):
             e = math.exp(w)
             return (w ** 2 * e + x) / ((w + 1) * e)
```

En itérant la fonction `flamb` nous devrions donc avoir des suites convergeant vers $W(x)$. Euh, oui, mais quel W ? W_1 ou W_2 ? Sans démonstration, l'idée est de choisir le premier terme de la suite dans l'image de la fonction W qui nous intéresse. Ainsi, si nous voulons calculer $W_1(x)$, nous choisissons $u_0 = -2 \in W_1([-\frac{1}{e}, 0[)$ et si nous voulons calculer $W_2(x)$, nous choisissons $u_0 = 2 \in W_1([-\frac{1}{e}, +\infty[)$. Ces choix sont arbitraires et risqués. On pourrait faire mieux (et on le fera plus loin) mais cela nous suffira pour l'instant.

Voici donc la fonction `lambertw`. Elle prend en paramètres un réel x, un numéro i de branche ($i = 1$ ou 2), et un paramètre optionnel `dbg` qui permet d'afficher le nombre d'itérations effectuées. Elle renvoie $W_i(x)$.

```
In [18]: def lambertw(x, branche, dbg=False, tol=1e-14):
             cnt = 0
             if branche == 1: u = -2
             else: u = 2
             #while abs(u - v) > 1e-15:
             while abs(u * math.exp(u) - x) > tol:
                 cnt += 1
                 u = flamb(u, x)
             if dbg: print(cnt)
             return u
```

```
In [19]: lambertw(-0.25, 1, True)
```

```
3
```

```
Out[19]: -2.153292364110349
```

```
In [20]: lambertw(-0.25, 2, True)
```

```
9
```

```
Out[20]: -0.3574029561813889
```

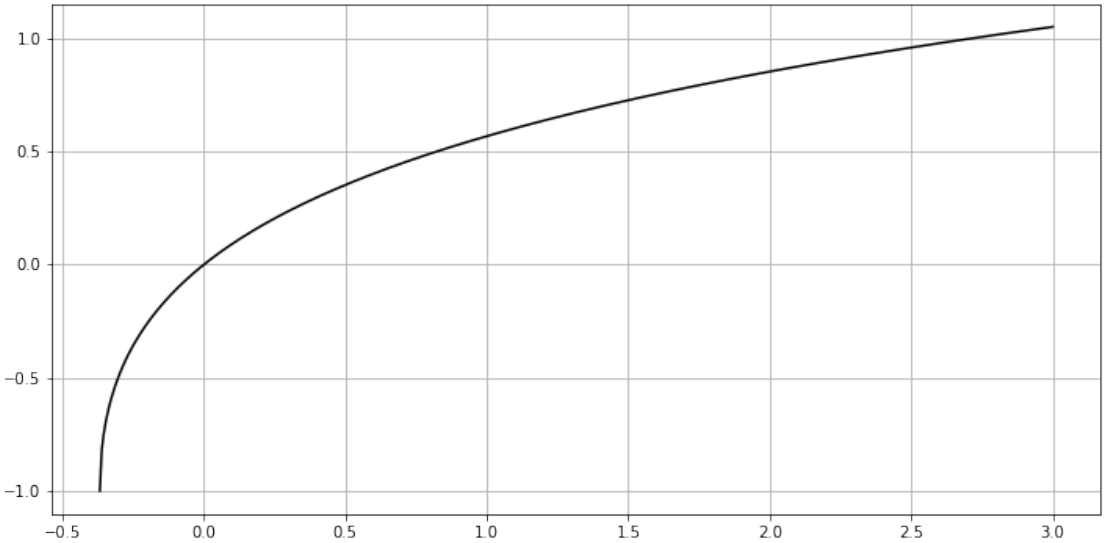
```
In [21]: lambertw(-1 / math.e, 2, True)
```

```
27
```

```
Out[21]: -0.9999998484721851
```

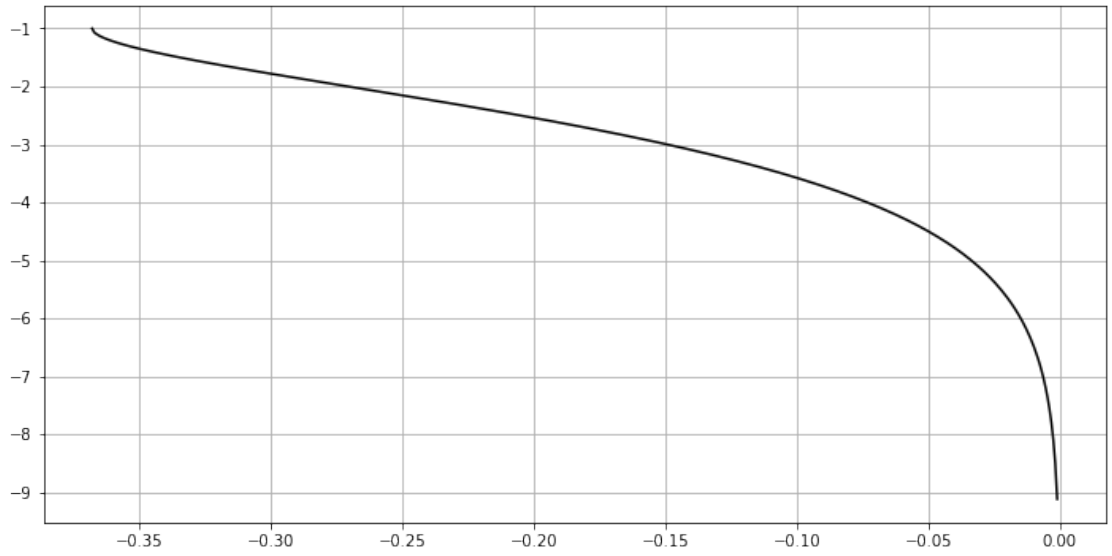
Voici la courbe de la fonction W_2. Et sans tricher cette fois-ci :-).

```
In [22]: xs = subdivision(-1 / math.e, 3, 500)
         ys = [lambertw(x, 2) for x in xs]
         plt.plot(xs, ys, 'k')
         plt.grid()
```



Et voici la courbe de la fonction W_1.

```
In [23]: xs = subdivision(-1 / math.e, -0.001, 500)
         ys = [lambertw(x, 1) for x in xs]
         plt.plot(xs, ys, 'k')
         plt.grid()
```



Parfait. Nous pouvons calculer assez rapidement $W_i(x)$, $i = 1, 2$.

1.2.4 2.3′ Petit aparté pour Romain Blanchard

On désire résoudre l'équation d'inconnues $x, y > 0$, (E) $x^y = y^x$. On vérifie facilement que ceci équivaut à $\frac{\ln x}{x} = \frac{\ln y}{y}$. L'étude de la fonction $t \mapsto \frac{\ln t}{t}$ permet de voir que les couples solutions vérifient $x, y > 1$, l'un des deux réels x, y étant inférieur à e et l'autre supérieur à e.

Posons $x = e^{-u}$ et $y = e^{-v}$. Le couple (x, y) vérifie (E) si et seulement si le couple (u, v) vérifie $ue^u = ve^v$ (exercice). On commence à voir le rapport avec la fonction de Lambert ...

Regardez la courbe de la fonction $f : x \mapsto xe^x$, vue plus haut. Pour tout réel $a \in]-\frac{1}{e}, 0[$ il existe exactement deux réels u et v tels que $f(u) = f(v) = a$. Ces deux réels sont bien sûr $W_1(a)$ et $W_2(a)$.

Pour revenir à l'équation qui nous intéresse, on prend $x \in]1, +\infty[$ arbitraire. On a alors $u = -\ln x < 0$ et $a = ue^u \in]-\frac{1}{e}, 0[$. On a $u = W_1(a)$ ou $u = W_2(a)$. En tout cas, le couple $(u', v') = (W_1(a), W_2(a))$, vérifie que $f(u') = f(v')$ et que l'un des deux réels u' et v' est u. Le couple $(x', y') = (e^{-u'}, e^{-v'})$ est alors une solution de (E) telle que l'un des deux réels x' et y' soit x.

On en déduit la fonction `solxyyx` qui prend en paramètre un réel $x > 1$ et renvoie le couple (x, y) (ou (y, x)) tel que $x^y = y^x$.

```
In [24]: def solxyyx(x):
             u = - math.log(x)
             a = u * math.exp(u)
             u = lambertw(a, 1)
             v = lambertw(a, 2)
```

```
            x = math.exp(-u)
            y = math.exp(-v)
            return(x, y)
```

```
In [25]: x, y = solxyyx(2)
         print('x = ', x)
         print('y = ', y)
         print('x^y = ', x ** y)
         print('y^x = ', y ** x)
```

```
x =  3.999999999999999
y =  1.999999999999999
x^y =  15.999999999999968
y^x =  15.999999999999954
```

1.2.5 2.4 $x^{[\infty]}$, plus efficacement

Rappelons-nous l'inefficacité de notre fonction `inf_exp` au voisinage des points e^{-e} et $e^{1/e}$. Écrivons donc une nouvelle fonction.

```
In [26]: def inf_exp2(x, dbg=False, tol=1e-14):
             return math.exp(-lambertw(-math.log(x), 2, dbg, tol))
```

```
In [27]: inf_exp2(math.sqrt(2), True)
```

```
10
```

```
Out[27]: 2.0000000000000004
```

```
In [28]: x = math.exp(-math.e)
         y = inf_exp2(x, True)
         print(y)
         print(1 / math.e)
```

```
6
0.3678794411714422
0.36787944117144233
```

6 itérations seulement, au lieu de 233158 !

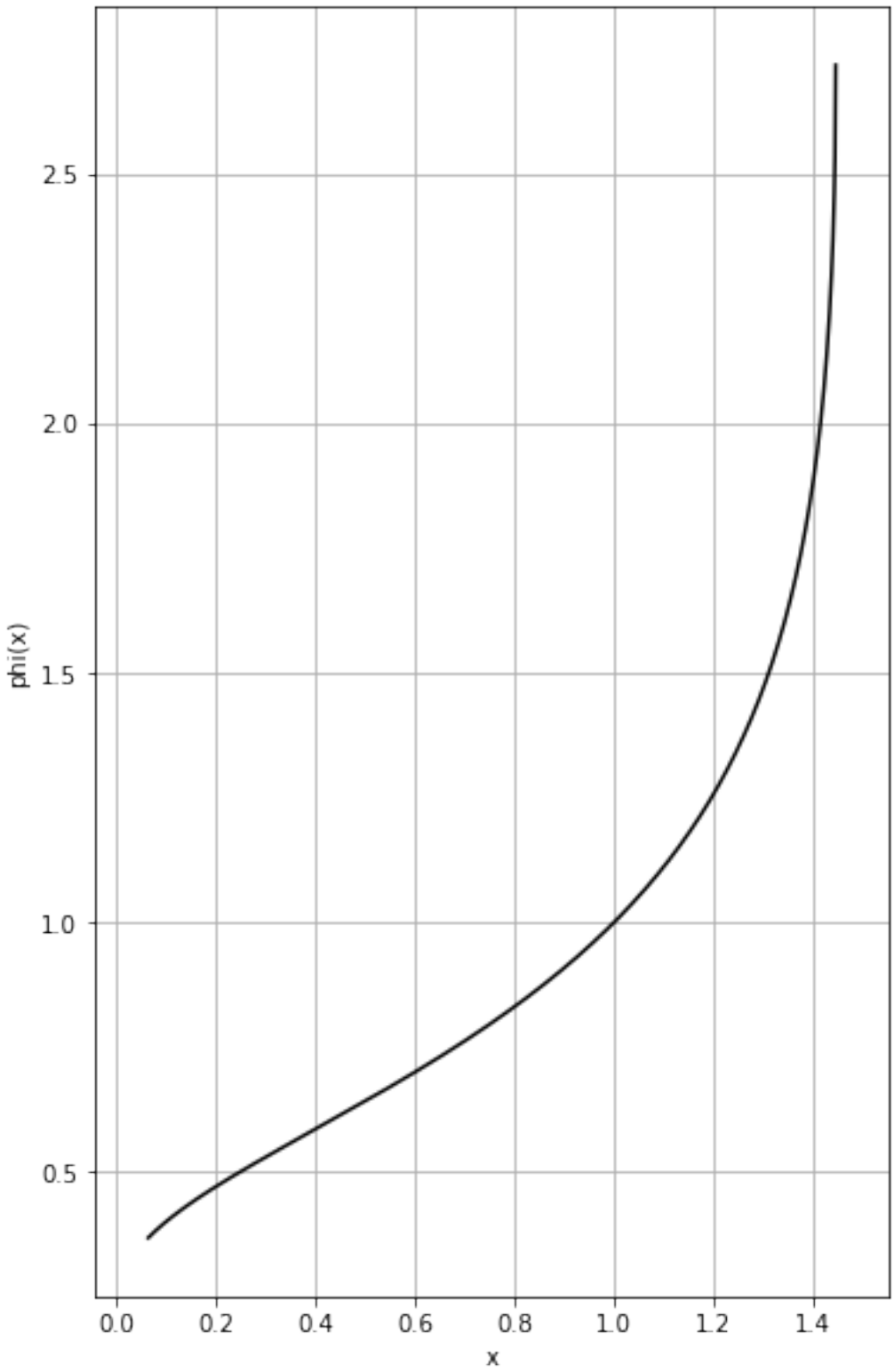
```
In [29]: x = math.exp(1 / math.e)
         y = inf_exp2(x, True)
         print(y)
         print(math.e)
```

```
27
2.718281413402751
2.718281828459045
```

Et ici, 24 itérations. Remarquez tout de même que la précision obtenue n'est pas terrible. La méthode de Newton n'aime pas les points à tangente horizontale, ce qui est le cas ici pour la fonction $x \mapsto xe^x$ en $-1/e$.

Voici enfin la courbe tanta-tendue de $x \mapsto x^{[\infty]}$.

```
In [30]: plt.rcParams['figure.figsize'] = (6, 10)

In [31]: xs = subdivision(math.exp(-math.e), math.exp(1 / math.e), 1000)
         ys = [inf_exp2(x) for x in xs]
         plt.plot(xs, ys, 'k')
         plt.xlabel('x')
         plt.ylabel('phi(x)')
         plt.axis('equal')
         plt.grid()
```




```
In [32]: plt.rcParams['figure.figsize'] = (12, 6)
```

Rappelez-vous la remarque concernant le cafouillis au voisinage de e^{-e}. Nous n'avons plus ce problème. Notre nouvel alorithme se comporte bien sur tout l'ensemble de définition de $x \mapsto x^{[\infty]}$.

1.2.6 2.4 Calcul encore plus efficace de W

Voici un extrait de la documentation de la fonction `lambertw` du module `scipy` :

.....

All branches are supported by lambertw:

`lambertw(z)` gives the principal solution (branch 0) `lambertw(z, k)` gives the solution on branch k The Lambert W function has two partially real branches: the principal branch ($k = 0$) is real for real $z > -1/e$, and the $k = -1$ branch is real for $-1/e < z < 0$. All branches except $k = 0$ have a logarithmic singularity at $z = 0$.

Possible issues

The evaluation can become inaccurate very close to the branch point at $-1/e$. In some corner cases, `lambertw` might currently fail to converge, or can end up on the wrong branch.

Algorithm

Halley's iteration is used to invert $w * exp(w)$, using a first-order asymptotic approximation ($O(log(w))$ or $O(w)$) as the initial estimate.

The definition, implementation and choice of branches is based on [R207].

.....

Comment fonctionne la méthode de Halley ? Soit à résoudre l'équation $f(x) = 0$. On considère la suite définie par son premier terme x_0 et la récurrence

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2f'(x_n)^2 - f(x_n)f''(x_n)}$$

Proposition : On suppose que f est de classe $\mathcal{C}^3$ ur un intervalle I, que $a \in I$ vérifie $f(a) = 0$ et $f'(a) \neq 0$. Alors, pour tout x_0 suffisamment proche de a, la suite (x_n) converge vers a. Il existe un réel $K > 0$ tel que, pour tout n,

$$|x_{n+1} - a| \leq K|x_n - a|^3$$

La présence de la puissance 3 indique une vitesse foudroyante de convergence. La méthode de Halley est une **méthode d'ordre 3**. À titre de comparaison, la méthode de Newton est une méthode d'ordre 2.

Exercice : Prenez $f(w) = we^w - x$. Que devient la méthode de Halley pour f ? Du coup, vous avez compris la fonction ci-dessous.

```
In [33]: def flamb2(w, x):
             e = math.exp(w)
             wx = w * e - x
             return w - wx / (e * (w + 1) - (w + 2) * wx / (2 * (w + 1)))
```

Voici notre nouvelle fonction de Lambert. J'ai effectué quelques modifications :

- La considération à part du cas où $x = -\frac{1}{e}$. Dans ce cas $f'(x) = 0$ et la méthode de Halley échoue.
- Le choix mystérieux du premier terme de la suite, $-1 \pm \sqrt{2(ex+1)}$. Je n'entre pas ici dans les détails. Ce choix est dicté par un développement en série de la fonction de Lambert au voisinage de x.

- Lorsque `dbg` vaut `True`, on renvoie le couple $(W_i(x), n)$ où n est le nombre d'itérations effectuées.

```
In [34]: def lambertw2(x, branch, dbg=False, tol=1e-14):
             cnt = 0
             p = math.sqrt(2 * (math.e * x + 1))
             if p == 0:
                 if dbg: return (-1, 0)
                 else: return -1
             if branch == 1: u = -1 - p
             else: u = -1 + p
             while abs(u * math.exp(u) - x) > tol:
                 cnt += 1
                 u = flamb2(u, x)
             if dbg: return (u, cnt)
             else: return u
```

Vérifions que tout va bien ...

```
In [35]: print(lambertw(-0.25, 1, True))
         print(lambertw2(-0.25, 1, True))
```

```
3
-2.153292364110349
(-2.15329236411035, 3)
```

```
In [36]: print(lambertw(-1 / math.e + 0.001, 2, True))
         print(lambertw2(-1 / math.e + 0.001, 2, True))
```

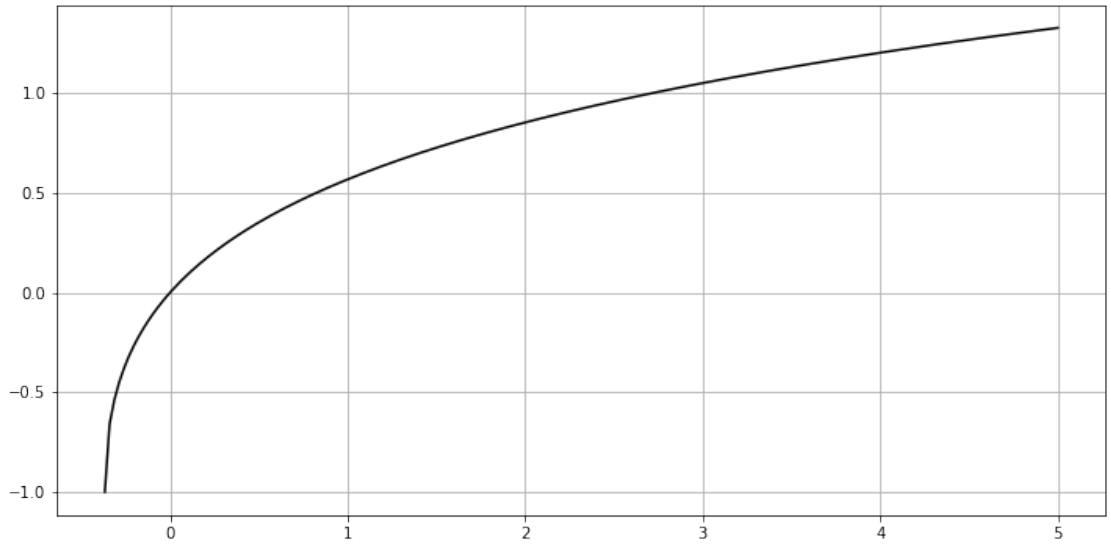
```
12
-0.9280201500545503
(-0.9280201500545691, 2)
```

Dans l'exemple ci-dessous, nous sommes obligés de diminuer la tolérance pour `lambertw`, sinon celle-ci échoue lamentablement. Pour `lambertw2`, en revanche, aucun problème.

```
In [37]: print(lambertw(100, 2, True, tol=1e-12))
         print(lambertw2(100, 2, True))
```

```
9
3.3856301402900506
(3.38563014029005, 14)
```

```
In [38]: xs = subdivision(-1 / math.e, 5, 200)
         ys = [lambertw2(x, 2) for x in xs]
         plt.plot(xs, ys, 'k')
         plt.grid()
```



Et voici notre nouvelle exponentielle infinie.

```
In [39]: def inf_exp3(x, dbg=False, tol=1e-14):
             if dbg:
                 y, v = lambertw2(-math.log(x), 2, dbg, tol)
                 return (math.exp(-y), v)
             else:
                 return math.exp(-lambertw2(-math.log(x), 2, dbg, tol))
```

```
In [40]: x = math.exp(- math.e)
         print(inf_exp3(x, True))
         print(1 / math.e)
```

```
(0.36787944117144233, 5)
0.36787944117144233
```

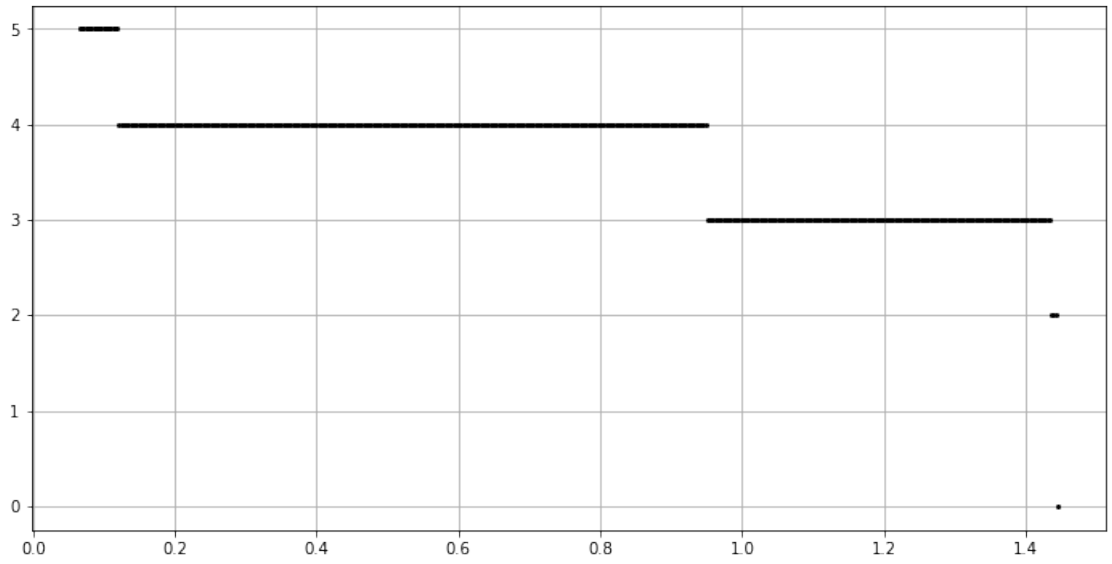
```
In [41]: x = math.exp(1 / math.e)
         y = inf_exp3(x, True)
         print(y)
         print(math.e)
```

```
(2.7182817711755396, 0)
2.718281828459045
```

Notre nouvelle fonction est-elle vraiment efficace ? Traçons, en fonction de x, le nombre d'itérations nécessaires pour calculer $x^{[\infty]}$.

```
In [42]: xs = subdivision(math.exp(-math.e), math.exp(1/math.e), 500)
         ys = []
```

```
for x in xs:
    (_, n) = inf_exp3(x, True, tol=1e-15)
    ys.append(n)
plt.plot(xs, ys, 'ko', markersize=2)
plt.grid()
plt.show()
```



Maximum 5 ! C'est **très** efficace.

1.3 3. Puissances infinies complexes

L'exposé qui suit reste à un niveau élémentaire. Je me contenterai de quelques explications et de dessins. Pas de preuves ...

Remarque : pour tout nombre complexe $z \neq 0$ nous noterons $\arg z$ l'unique argument de z dans l'intervalle $]-\pi, \pi]$.

1.3.1 3.1 Logarithmes complexes

```
In [43]: import cmath
```

Soit $z \in \mathbb{C}^*$. Quels sont les nombres complexes w tels que $e^w = z$? Posons $z = re^{i\theta}$ où $r > 0$ est le module de z et $-\pi < \theta \leq \pi$ est son argument. Posons $w = x + iy$, où $x, y \in \mathbb{R}$. On a $e^w = z$ si et seulement si $e^x e^{iy} = re^{i\theta}$, ou encore $x = \ln r$ et $\exists k \in \mathbb{Z}, y = \theta + 2k\pi$. Ainsi, $w = \ln|z| + i \arg z + 2ik\pi$.

Posons, pour tout $k \in \mathbb{Z}$, $\log_k(z) = \ln|z| + i \arg z + 2ik\pi$. Nous venons de définir pour tout entier k la $k$ième branche du *logarithme complexe*. Pour $k = 0$, nous noterons plus simplement $\log z = \log_0 z$. La fonction log est appelée la *branche principale* du logarithme.

Que dire de la fonction $\log_k$? Eh bien, par définition même, $\log_k : \mathbb{C}^* \to \mathbb{C}$. Et pour tout $z \neq 0$, nous avons $e^{\log_k z} = z$.

Soit maintenant $z = x + iy \in \mathbb{C}$. Que vaut $\log_k e^z$? On a $|e^z| = e^x$ et $\arg e^z = iy - 2i\ell\pi$ où $\ell \in \mathbb{Z}$ est l'unique entier tel que $-\pi < y - 2\ell\pi \leq \pi$. De là, $\log_k e^z = x + iy + 2i(k-\ell)\pi = z + 2i(k-\ell)\pi$, qui est en général différent de z ! Attention, donc, le logarithme n'est pas la réciproque de l'exponentielle. Précisément, c'est un inverse à droite ($\exp \circ \log = id$) mais pas à gauche ($\log \circ \exp \neq id$).

Exercice : Montrer que $\log e^z = z$ si et seulement si la partie imaginaire de z appartient à $]-\pi, \pi]$.

Exemples :

- Si x est un réel strictement positif, alors $|x| = x$ et $\arg x = 0$. Donc $\log x = \ln x$.
- Si x est un réel strictement négatif, alors $|x| = -x$ et $\arg x = \pi$. Donc $\log x = \ln(-x) + i\pi$. Par exemple, $\log(-1) = i\pi$.
- $\log i = i\frac{\pi}{2}$

Remarques :

- Attention au logarithme complexe, il ne vérifie pas les belles propriétés du logarithme réel. Par exemple, $\log((-1)(-1)) = \log 1 = 0$ mais $\log(-1) + \log(-1) = 2i\pi$. On n'a donc PAS la jolie propriété de morphisme du logarithme réel.
- Le logarithme complexe présente une discontinuité sur l'axe des réels négatifs. Ainsi, si $\varepsilon > 0$ est un réel "petit", on a $\log(-1 + i\varepsilon) \simeq i\pi$ alors que $\log(-1 - i\varepsilon) \simeq -i\pi$. les nombres complexes $-1 + i\varepsilon$ et $-1 - i\varepsilon$ sont très proches mais leurs logarithmes diffèrent d'environ $2i\pi$.

```
In [44]: z1 = cmath.log(-1+0.001j)
         z2 = cmath.log(-1-0.001j)
         print(z1)
         print(z2)
         print(z1 - z2)
```

```
(4.999997500001667e-07+3.1405926539231266j)
(4.999997500001667e-07-3.1405926539231266j)
6.281185307846253j
```

Exercice : Montrer que pour tous $z, z' \in \mathbb{C}$, on a $\log(zz') \equiv \log z + \log z'[2i\pi]$.

Voici à titre d'illustration le graphe du logarithme. Mais c'est quoi le graphe d'une fonction de $\mathbb{C}$ vers $\mathbb{C}$? Il y a plusieurs façons de voir les choses. L'idée que nous allons retenir ici est la suivante. Prenons des courbes du plan et regardons leurs images par la fonction log. Nous allons obtenir de nouvelles courbes. En traçant ce nouvelles courbes nous voyons comment le logarithme *déforme* le plan complexe.

Quelles courbes choisir ? Voici quelques choix possibles :

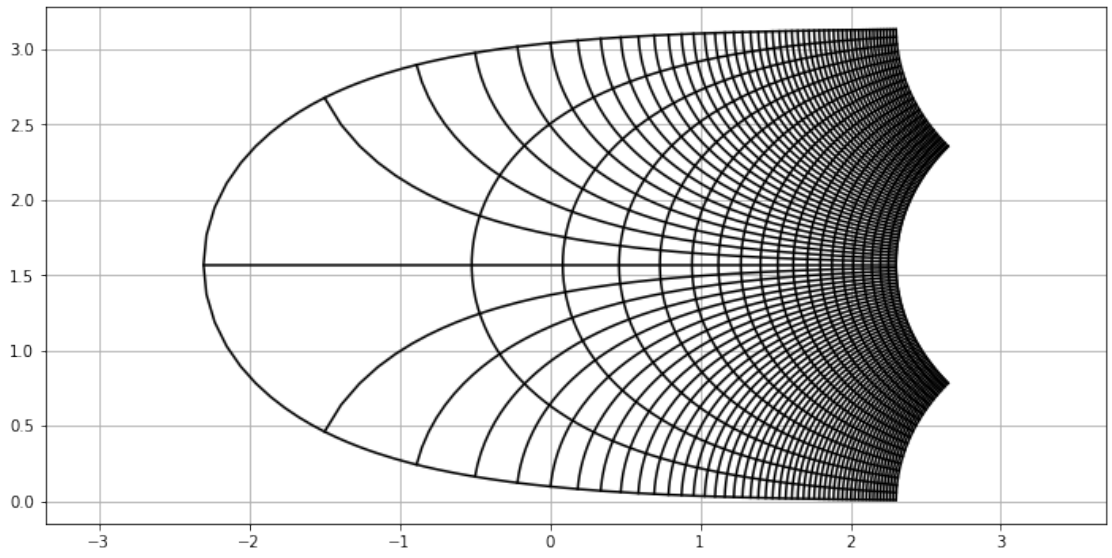
- Les droites parallèles aux axes de coordonnées. C'est le choix "cartésien", que nous allons prendre ici.
- Les cercles centrés en O et les demi-droites issues de O. C'est le choix "polaire".
- D'autres familles de courbes ...

Remarquez que nous prenons à chaque fois **DEUX** familles de courbes qui se coupent à angles droits. Nous y reviendrons plus loin.

Pour éviter le problème de discontinuité du logarithmes aux réels négatifs, nous ne traçons ici que pour $y > 0$. Pour $y < 0$ on obtiendrait une figure symétrique. Pourquoi ? Parce que :

Exercice : Montrer que pour tout $z \in \mathbb{C}^*$ on a $\log \overline{z} = \overline{\log z}$.

```
In [45]: # Images des droites x = cte
         xmin, xmax = -10, 10
         ymin, ymax = 0.1, 10
         for x in subdivision (xmin, xmax, 100):
             xs = []
             ys = []
             for y in subdivision(ymin, ymax, 200):
                 if x != 0 or y != 0:
                     w = cmath.log(x + 1j * y)
                     xs.append(w.real)
                     ys.append(w.imag)
             plt.plot(xs, ys, 'k')
         # Images des droites y = cte
         for y in subdivision (ymin, ymax, 20):
             xs = []
             ys = []
             for x in subdivision(xmin, xmax, 1000):
                 if x != 0 or y != 0:
                     w = cmath.log(x + 1j * y)
                     xs.append(w.real)
                     ys.append(w.imag)
             plt.plot(xs, ys, 'k')


         plt.grid()
         plt.axis('equal')
         plt.show()
```



Que voyons-nous ? Les droites horizontales et verticales se sont transformées en des courbes dont nous pourrions trouver éventuellement l'équation. Je vous laisse faire cet exercice.

Remarque : Les courbes du dessin ont l'air de se couper à angles droits. Est-ce une coïncidence ? Non ! La fonction log "conserve les angles". C'est ce que l'on appelle une **transformation conforme**.

1.3.2 3.2 Puissances complexes

Pour $a, b \in \mathbb{C}$, $a \neq 0$, posons $a^b = e^{b \log a}$.

Exemples :

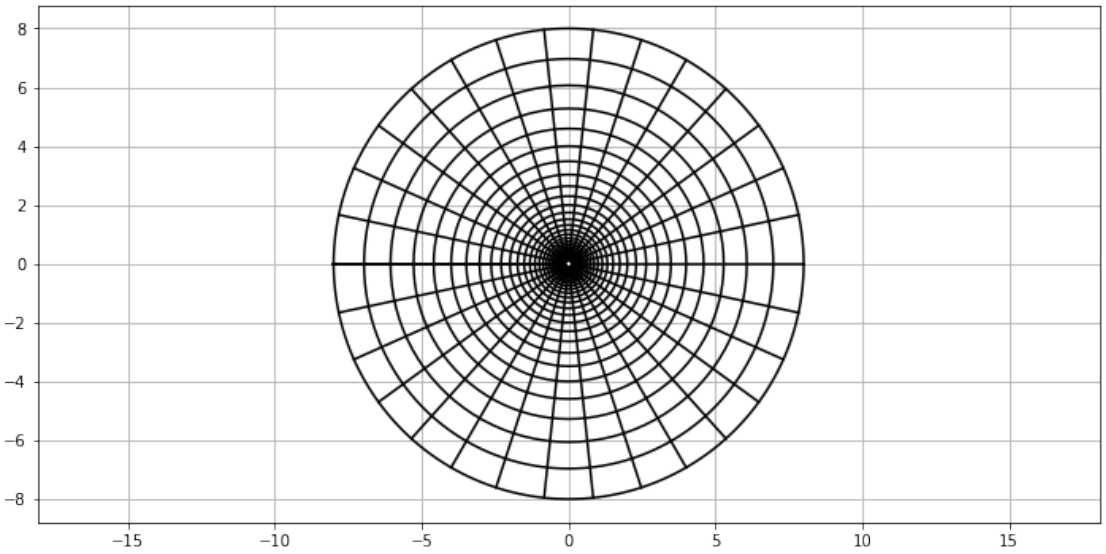
- Si b est entier, $a^b = e^{b \log a} = (e^{\log a})^b = \ldots a^b$, au sens usuel de l'expression. Tout va bien !
- $i^i = e^{i \log i} = e^{-\frac{\pi}{2}} \in \mathbb{R}$.

Ci-dessous le graphe de $z \mapsto 2^z$.

Exercice : Pourquoi ces choix pour `ymin` et `ymax` ?

```
In [46]: # Images des droites x = cte
         xmin, xmax = -3, 3
         ymin, ymax = -math.pi / math.log(2), math.pi / math.log(2)
         for x in subdivision (xmin, xmax, 30):
             xs = []
             ys = []
             for y in subdivision(ymin, ymax, 200):
                 w = 2 ** (x + 1j * y)
                 xs.append(w.real)
                 ys.append(w.imag)
             plt.plot(xs, ys, 'k')
         # Images des droites y = cte
         for y in subdivision (ymin, ymax, 30):
```

```
        xs = []
        ys = []
        for x in subdivision(xmin, xmax, 200):
            w = 2 ** (x + 1j * y)
            xs.append(w.real)
            ys.append(w.imag)
        plt.plot(xs, ys, 'k')

plt.grid()
plt.axis('equal')
plt.show()
```



Exercice : Commenter ce dessin. Demi-droites, cercles, angles droits.

Munis de puissances complexes quelconques, nous pouvons nous attaquer aux puissances infinies ...

1.3.3 3.3 Puissances infinies complexes

Nous admettrons le résultat suivant :

Proposition : Soit $z \in \mathbb{C}$. Le nombre complexe $z^{[\infty]}$ est défini si et seulement si $z \in V$ où

$$V = \{e^{te^{-t}}, |t| < 1 \; ou \; \exists n \geq 1, t^n = 1\}$$

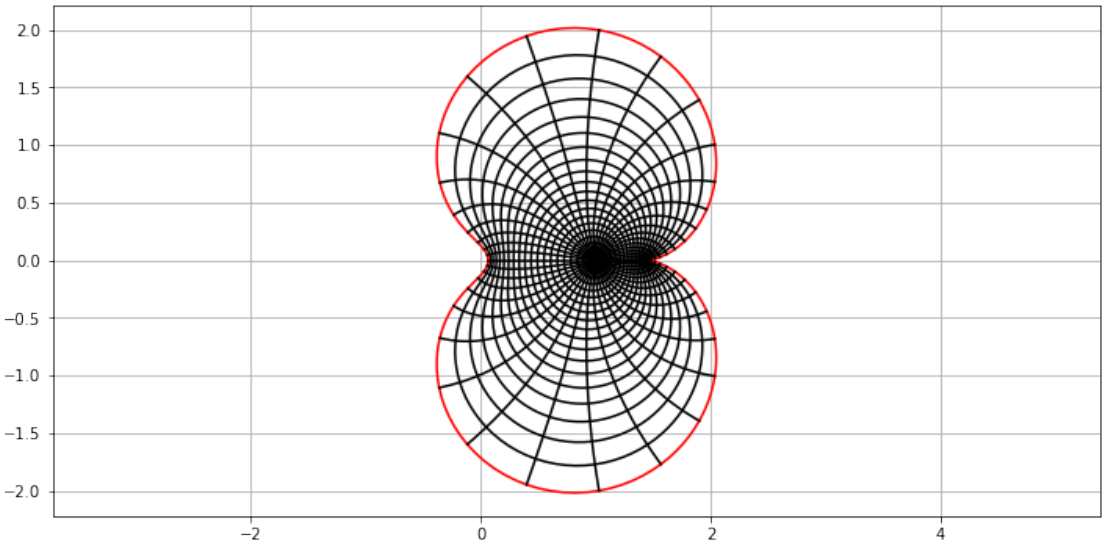
Dessinons l'ensemble V. Plus précisément, dessinons $V_0 = \{e^{te^{-t}}, |t| < 1\}$ et, pour tout $n \geq 1$, les ensembles $V_n = \{e^{te^{-t}}, t^n = 1\}$. Si nous posons $\Phi(z) = e^{ze^{-z}}$, nous avons alors

- $V_0 = \Phi(D)$ où $D = \{z \in \mathbb{C}, |z| < 1\}$ est le disque ouvert de centre O et de rayon 1
- $V_n = \Phi(\mathcal{U}_n)$ où $\mathcal{U}_n = \{z \in \mathbb{C}, z^n = 1\}$ est l'ensemble des racines $n$ièmes de l'unité.

Commençons par V_0. Nous adoptons la méthode déjà vue plus haut, sauf que D étant rond nous travaillons en coordonnées polaires. Nous traçons les images par Φ des cercles de centre O et des demi-droites issues de O.

Dans le dessin ci-dessous, la ligne extérieure rouge correspond à $r = 1$. Elle n'est donc pas incluse dans V_0. En fait, V_0 est la partie du plan située strictement à l'intérieur de cette ligne.

```
In [47]: for r in subdivision(0, 1, 20):
             xs = []
             ys = []
             for theta in subdivision (0, 2 * math.pi, 300):
                 t = r * cmath.exp(1j * theta)
                 w = cmath.exp(t * cmath.exp(-t))
                 xs.append(w.real)
                 ys.append(w.imag)
             if r < 1: col = 'k'
             else: col = 'r'
             plt.plot(xs, ys, col)
         for theta in subdivision(0, 2 * math.pi, 50):
             xs = []
             ys = []
             for r in subdivision (0, 1, 200):
                 t = r * cmath.exp(1j * theta)
                 w = cmath.exp(t * cmath.exp(-t))
                 xs.append(w.real)
                 ys.append(w.imag)
             plt.plot(xs, ys, 'k')
         plt.grid()
         plt.axis('equal')
         plt.show()
```



Ensemble compliqué, certes, mais abordable. Regardez en particulier les deux "pincements", ils se trouvent pile aux réels e^{-e} et $e^{1/e}$.

Et les ensembles V_n ? Ils forment une partie dense de la ligne rouge. En effet, l'ensemble de toutes les racines $n$ièmes de l'unité, pour toutes les valeurs de n, est dense dans le cercle unité $\mathcal{U}$. Il en résulte par continuité de Φ que $\cup_{n\geq 1} V_n$ est dense dans la ligne extérieure, qui n'est autre que $\Phi(\mathcal{U})$!

Nous avons donc bien dessiné l'ensemble V. Nous retrouvons l'intervalle $[e^{-e}, e^{1/e}]$ pour les réels de cet ensemble. Et nous voyons que $i \in V$.

1.3.4 3.4 La fonction W dans $\mathbb{C}$

Réécrivons la fonction de Lambert, ainsi que les puissances infinies pour que tout cela fonctionne avec un paramètre complexe.

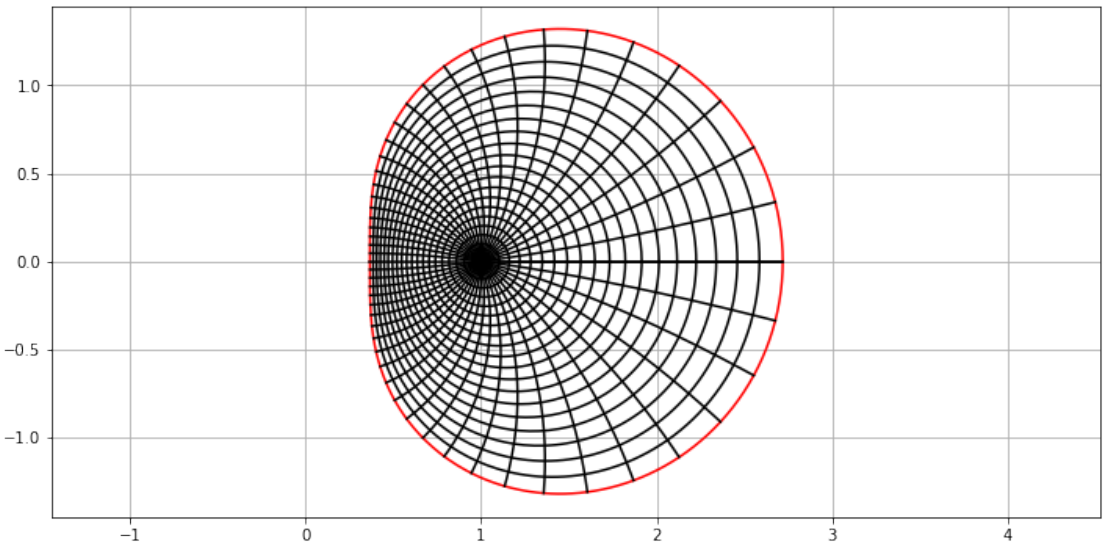
Remarque : La fonction de Lambert complexe possède en fait une **infinité** de branches. Nous nous cantonnons dans ce qui suit aux deux branches déjà évoquées pour les réels.

```
In [48]: def flambc(w, x):
             e = cmath.exp(w)
             wx = w * e - x
             return w - wx / (e * (w + 1) - (w + 2) * wx / (2 * (w + 1)))

In [49]: def lambertwc(x, branch, tol=1e-14):
             p = cmath.sqrt(2 * (math.e * x + 1))
             if p == 0: return -1
             if branch == 1: u = -1 - p
             else: u = -1 + p
             while abs(u * cmath.exp(u) - x) > tol:
                 u = flambc(u, x)
             return u

In [50]: def inf_expc(x, tol=1e-14):
             return cmath.exp(-lambertwc(-cmath.log(x), 2, tol))

In [51]: inf_expc(1j)

Out[51]: (0.43828293672703206+0.3605924718713855j)

In [52]: inf_expc(math.sqrt(2))

Out[52]: (2.0000000000000004-0j)
```

Et maintenant, traçons le graphe de $z \mapsto z^{[\infty]}$! Pour être honnête, l'ensemble de départ de la fonction étant très compliqué, traçons le graphe de $\Phi(z)^{[\infty]}$. L'ensemble de départ de cette fonction étant un disque, le choix de coordonnées polaires s'impose.

```
In [53]: for r in subdivision(0, 1, 20):
             xs = []
             ys = []
             for theta in subdivision (0, 2 * math.pi, 300):
```

```
            t = r * cmath.exp(1j * theta)
            w = inf_expc(cmath.exp(t * cmath.exp(-t)))
            xs.append(w.real)
            ys.append(w.imag)
        if r < 1:
            plt.plot(xs, ys, 'k')
        else:
            plt.plot(xs, ys, 'r')
for theta in subdivision(0, 2 * math.pi, 50):
    xs = []
    ys = []
    for r in subdivision (0, 1, 200):
        t = r * cmath.exp(1j * theta)
        w = inf_expc(cmath.exp(t * cmath.exp(-t)))
        xs.append(w.real)
        ys.append(w.imag)
    plt.plot(xs, ys, 'k')
plt.grid()
plt.axis('equal')
plt.show()
```



Exercice : Comprenez le dessin ci-dessus.

1.3.5 3.5 Que se passe-t-il en dehors de l'ensemble de définition ?

Que se passe-t-il **en dehors** de l'ensemble de définition de $z \mapsto z^{[\infty]}$? Eh bien, la suite $(z^{[n]})$ diverge, me direz-vous. Oui, mais il y a plusieurs façons de diverger. La suite peut tendre (en module) vers l'infini, ou alors les termes d'indices pairs converger vers une certaine limite, et ceux d'indice impair vers une autre. Ou alors il faut peut-être regarder modulo 3 ou 4. Ou alors la suite peut faire n'importe quoi.

Bref, tentons une expérience. Pour $z \in \mathbb{C}$, calculons $z^{[0]}, z^{[1]}, z^{[2]}$, etc, jusqu'à ce que $|z^{[n]}|$ devienne "très grand" (disons plus grand que 200), ou que n dépasse une certaine valeur `niter`. Et renvoyons le nombre d'itérations effectuées. La fonction `iterer` fait le travail. Elle renvoie également 0 si la suite à l'air de converger.

```
In [54]: def iterer(z, niter):
             k = 0
             w = 1
             w1 = z ** w
             while k < niter and abs(w1) < 200:
                 w, w1 = w1, z ** w1
                 k = k + 1
             if abs(w - w1) < 1e-5: k = 0
             return k
```

Par exemple, pour $z = 2$ la suite tend très vite vers l'infini.

```
In [55]: iterer(2, 256)
```

```
Out[55]: 3
```

Pour $z = i$, la suite converge.

```
In [56]: iterer(1j, 256)
```

```
Out[56]: 0
```

Pour $z = 3 + 2i$, on a divergence, mais la suite diverge "moins vite" que pour "z=2".

```
In [57]: iterer(3+2j, 256)
```

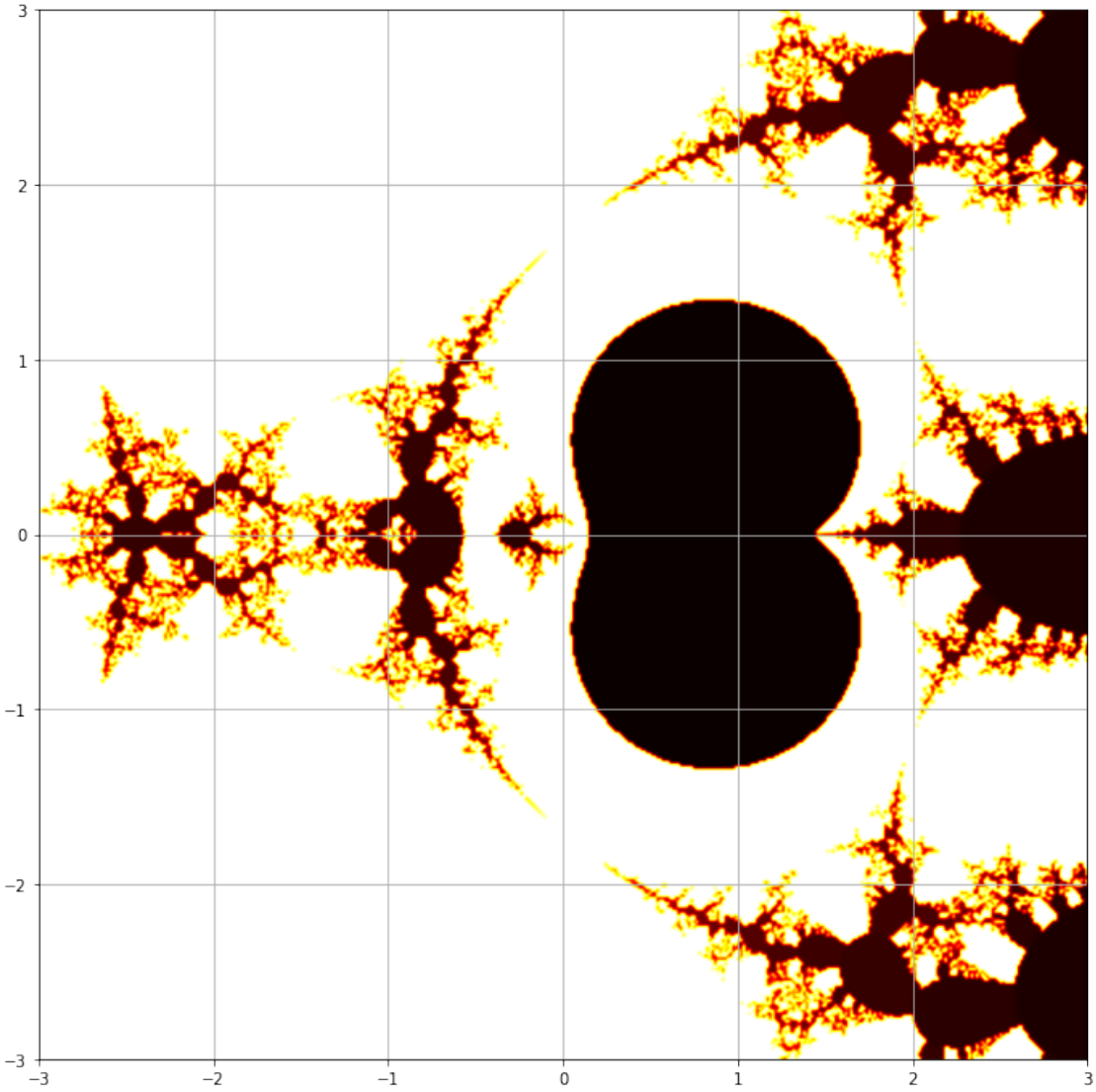
```
Out[57]: 14
```

Maintenant, itérons pour tous les z appartenant à un certain rectangle et mettons les valeurs retournées par `iterer` dans une matrice. Puis affichons. Nous obtenons une carte qui nous donne la "vitesse de divergence" en fonction de z. Je ne commenterai pas. Contentez-vous de regarder.

Soyez patients, l'affichage prend un certain temps (environ 7 secondes sur ma machine).

```
In [58]: plt.rcParams['figure.figsize'] = (12, 12)
         a = 3
         xmin, xmax = -a, a
         ymin, ymax = -a, a
         nx, ny = 400, 400
         niter = 64
         M = (ny + 1) * [None]
         for i in range(ny + 1):
             M[i] = (nx + 1) * [0]
         for i in range(nx + 1):
             for j in range(ny + 1):
                 x = xmin + i * (xmax - xmin) / nx
```

```
            y = ymin + j * (ymax - ymin) / ny
            z = x + 1j * y
            M[j][i] = 4 * iterer(z, niter)
    plt.imshow(M, origin='lower', cmap='hot', interpolation='bicubic', extent=(xmin, xmax,
    plt.grid()
    plt.show()
```



Nous voyons au centre l'ensemble déjà vu plus haut (une espèce de néphroïde diront certains), là où la suite converge. Et tout autour, des tas de choses intéressantes. Une structure très compliquée, et certainement digne d'être étudiée. Ce que je ne ferai pas ici.

Quelques suggestions :

- Augmentez les valeurs de `nx` et `ny`, à 600 par exemple. Attention, multiplier ces valeurs par 2 multiplie le temps de calcul par 4.

- Faites un zoom autour des points -1, e^{-e} et $e^{\frac{1}{e}}$ en ajustant les valeurs de `xmin`, `xmax`, `ymin` et `ymax`.
- Cherchez d'autres points intéressants.

1.4 4. Petite bibliographie

Outre les sources standard d'information, les deux articles ci-dessous m'ont été d'une aide précieuse. Ils se trouvent facilement sur Internet.

- Corless, Gonnet, Hare, Jeffrey, Knuth - *On the Lambert W Function*, Advances in Computational Mathematics, **5**, 329-359.
- Galidakis - *On an Application of Lambert's W Function to Infinite Exponentials*, Complex Variables, Vol 49, N^o 11, 15 September 2004, 759-780.
- Documentation de scipy : fonction lambertw
- Documentation de mpmath : fonction lambertw