Fêter le π-day avec 13 jours de retard (Denise Vella-Chemla, 27 mars 2024)

On continue d'essayer de simplifier ce qu'on a trouvé jusque là, toujours dans le but de comprendre.

On utilise pour l'exponentielle une fonction simplifiée *monexp*. Elle utilise le fait que [1]

$$\exp x = \lim_{n\to\infty} \left(1 + x \times \frac{1}{n}\right)^n .$$

On décide de remplacer l'infini par 10000.

On utilise pour l'intégrale, plutôt que la fonction *quad* du package `scipy.integrate` de `python` [2] une fonction qu'on appelle *integrale*2 et qui approxime l'intégrale par la méthode dite Simpson 1/3 (voir par exemple ici https://en.wikipedia.org/wiki/Simpson%27s_rule.).

Le programme qui semble montrer les zéros de ζ devient plus simple, le voici.

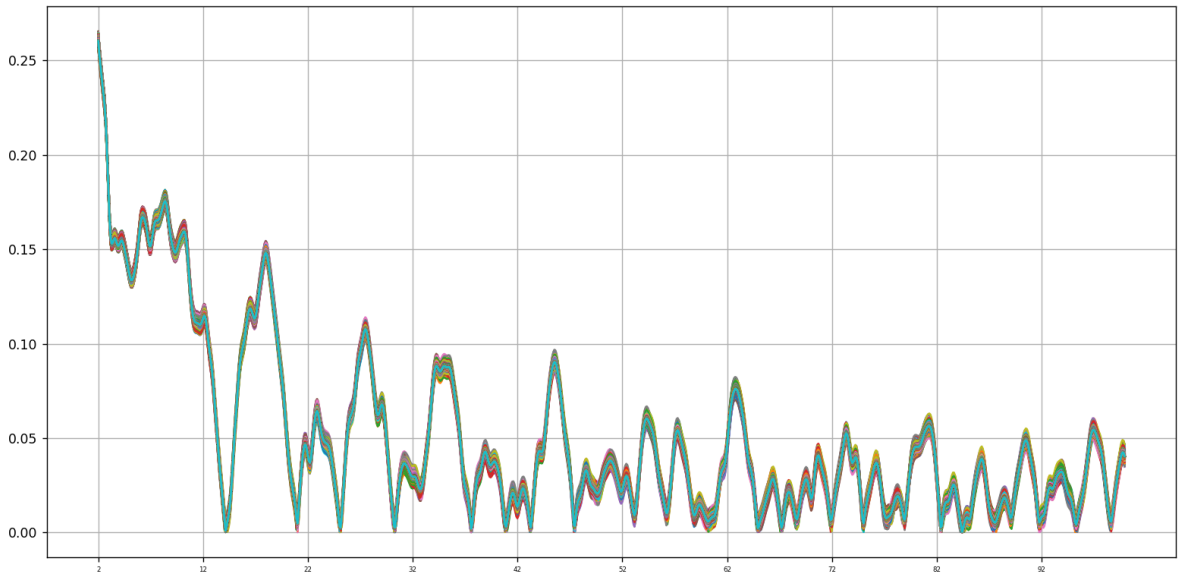
```
import matplotlib.pyplot as plt
import numpy as np
from numpy import linspace, modf, e, sqrt, log, cos, sin, exp, floor, pi
def integrale2(fct, a, b, nbpts):    integration par methode Simpson 1/3
     assert(nbpts%2 == 0)
     x = np.linspace(a,b,num=nbpts+1,endpoint=True)
     h = (b-a)/nbpts
     y = fct(x)
     return (y[0]+4*np.sum(y[1:nbpts:2])+2*np.sum(y[2:nbpts-1:2])+y[1:])  * h/3

def monexp(x):    n represente une sorte d'infini, quand on est petit
     n=10000 ; return(pow(1+x/n,n))

def monlog(x):    meme remarque que pour fonction monexp
     n=10000 ; return((-1+pow(x,1/n))*n)

def y(t):
     a = integrale2(lambda x :  (modf(monexp(x))[0])*cos(x*t)/monexp(x/2),-10,10,1000)
     b = integrale2(lambda x :  (modf(monexp(x))[0])*sin(x*t)/monexp(x/2),-10,10,1000)
     return(sqrt(a*a+b*b))

minx = 2
maxx = 100
nbpoints = 1000
a, b, c = minx, maxx, nbpoints
plt.grid(True)
plt.xticks(range(minx,maxx,10), fontsize=5)
t = [a+k*(b-a)/c for k in range(c)]  puisque a = 0 remplacer par k*b/c
plt.plot(t, [y(tt) for tt in t])
plt.show()
```

[1] On a, symétriquement, trouvé que $\ln(x) = \lim_{n\to\infty} \left(-1 + x^{\frac{1}{n}}\right) \times n$.

[2] angoissante car elle n'arrête pas d'écrire en sortie des messages d'appel à la prudence pour des problèmes de convergence...

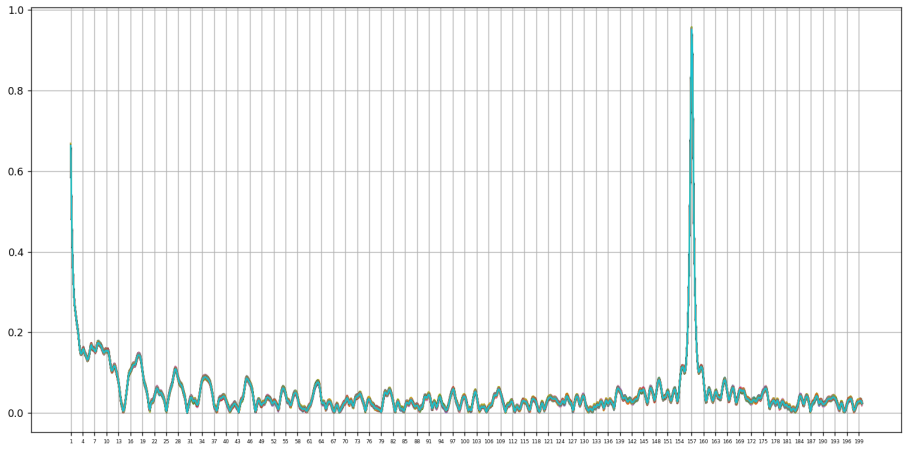
Il faut souligner que l'idée initiale ayant mené à ce programme est celle de Balthasar van der Pol, qui a montré les zéros de ζ en inventant un dispositif constitué d'une scie électro-mécanique circulaire à dents de tailles décroissantes et tournant à une vitesse précise, et l'envoi à certaines fréquences de faisceaux de lumière qui passaient à travers les dents de la scie tournante (voir article de van der Pol de 1947).

Voici son résultat :

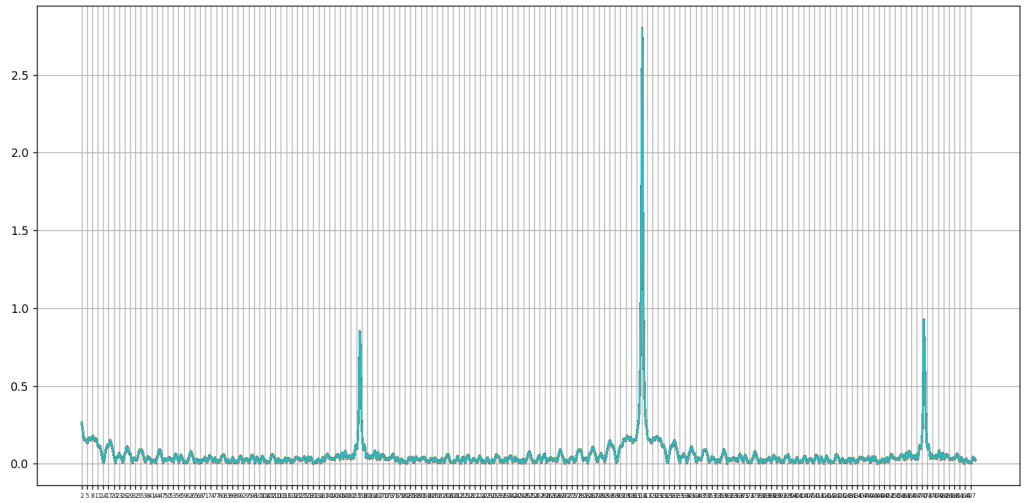


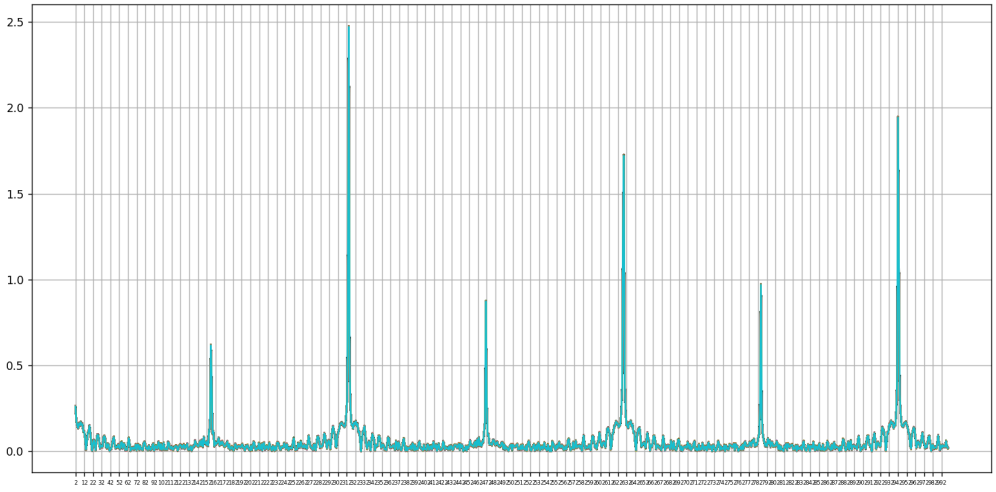
L'intégrale complexe est séparée en intégrale sur le cosinus, intégrale sur le sinus, puis agrégation des deux intégrales en renvoyant $\sqrt{a^2 + b^2}$.

On décide d'aller au-delà de 100 et là, surprise, il y a une pointe pour $n = 157$. On ne comprend pas, on cherche le cosinus de 157, il est proche de 1 (0.95). Ce qui est intéressant, c'est de voir l'aspect symétrique des zéros autour de ce pôle : il y a un zéro en 179, symétrique de 135 par rapport à 157, puis un autre, en 183 (sym. de 131), puis en 187, 195, 198 (sym resp. de 127, 119, 116).



Du coup, on programme jusqu'à 500 et là, on voit apparaître enfin des sortes de Dirac[3] (c'est le nom donné aux pics de valeurs que prend une fonction autour desquels les valeurs sont nulles). Ici les valeurs autour des simili-Dirac sont beaucoup plus petites que les valeurs prises aux pics. Les pics ont lieu pour les valeurs 157, 314, ..., c'est à dire pour les valeurs multiples de $(100\times)\ \pi/2$.





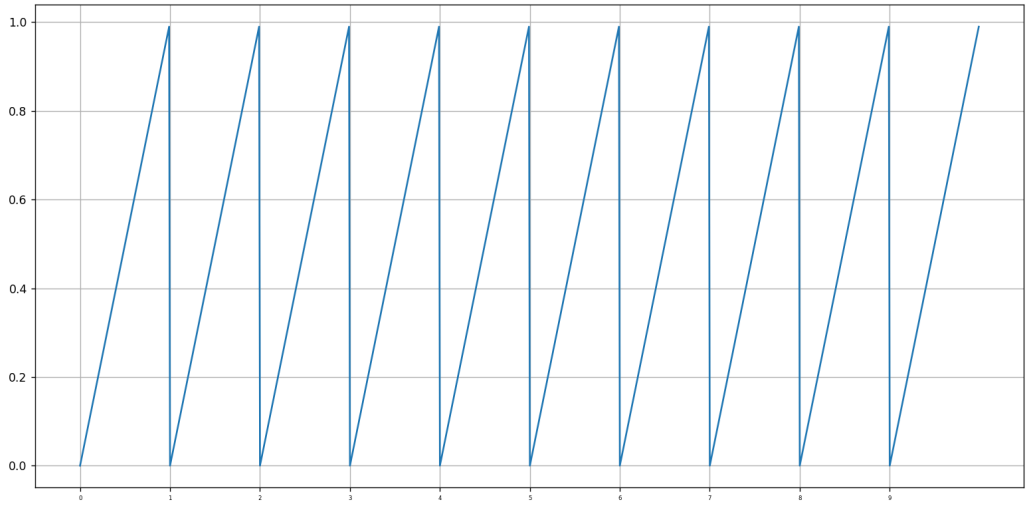
Si l'on souhaite (jusqu'à 200)"ramener le pic de 157 vers le bas, pour rater moins de zéros autour de 157, on peut changer le paramètre *nbpoints*.

Par rapport aux zéros de la fonction ζ, on en rate un certain nombre, notamment autour des pics, et ce d'autant plus qu'on va loin.

De toute façon, on reste ennuyée par la fonction partie fractionnaire, $f(x) = x \bmod 1$, (programmée en python soit en utilisant le symbole % lorsqu'on l'utilise sur des réels ou entiers, soit en utilisant

[3]Cela corrobore peut-être le fait que certains utilise un opérateur de Dirac, dans leur quête de la démonstration de l'hypothèse de Riemann.

la fonction `modf` du package `numpy` lorsqu'on souhaite l'utiliser sur des vecteurs) qui calcule le reste modulaire : cette fonction est une fonction en dents de scie[4], elle est donc problématique pour pouvoir raisonner convenablement. On rappelle son allure ci-dessous :



[4]C'est un peu le cas de le dire.