Tautologies, matrices, rotations, décembre 2024

Cette manière de voir les nombres premiers me semble naturelle : plutôt que d'éliminer, selon l'instruction compliquée, qu'on doit appliquer lorsqu'on met en œuvre le crible d'Eratosthène, "tous les multiples du premier nombre qui n'a pas été barré", je préfère tout ramener sur le cercle unité, et éliminer les nombres n qui sont égaux à 1, quand on tourne n fois d'un angle plus petit $2\pi/d$, le retour sur le nombre 1 ayant lieu lorsque n est un multiple (sous-entendu entier) de d.

Ci-dessous, deux programmes python qui correspondent à cette façon de voir, l'un utilisant des points du cercle unité et l'autre utilisant les matrices de rotation correspondantes, selon l'idée de la Commission Lichnérowicz et qu'on trouve à cette page du site de Gérard Villemin :

http://villemin.gerard.free.fr/Wwwgvmm/Type/aaaCompl/Matrice.htm.

L'idée consiste à représenter les nombres complexes par des matrices 2×2.

Programme utilisant des exponentielles de norme 1

```
import mpmath
from mpmath import exp,pi
import math
from math import sqrt,floor

def premier(atester):
     k = 2
     if atester in [0, 1]:  return False
     if atester in [2, 3, 5, 7]:  return True
     while True:
          if k * k > atester:  return True
          else:
               if atester % k == 0:  return False
               else:  k = k + 1

epsilon = 1e-6
n = 98
for x in range(3,n//2+1,2):
     if x > sqrt(n):
          bool = True
          for p in range(3,floor(sqrt(n))+1):
               if premier(p):
                    if (abs(exp(2*pi*1j*n/p).real-exp(2*pi*1j*x/p).real) < epsilon)
                   and (abs(exp(2*pi*1j*n/p).imag-exp(2*pi*1j*x/p).imag) < epsilon):
                         bool = False
                    if abs(exp(2*pi*1j*x/p).real-1) < epsilon:
                         bool = False
          if bool:
               print(x, ' est un decomposant de Goldbach de ',n)
```

Programme utilisant des matrices de rotations

```
import numpy as np
from numpy import cos,sin,pi
import math
from math import floor, sqrt

def premier(atester):
     k = 2
     if atester in [0, 1]:  return False
     if atester in [2, 3, 5, 7]:  return True
     while True:
          if k * k > atester:  return True
          else:
               if atester % k == 0:  return False
               else:  k = k + 1

def matricesegales(M1,M2,epsilon):
     return((abs(M1[0,0]-M2[0,0])<epsilon) and
            (abs(M1[0,1]-M2[0,1])<epsilon) and
            (abs(M1[1,0]-M2[1,0])<epsilon) and
            (abs(M1[1,1]-M2[1,1])<epsilon))

epsilon = 1e-6
n = 98
for x in range(3,n//2+1,2):
     if x > sqrt(n):
          bool = True
          for p in range(3,floor(sqrt(n))+1):
               if premier(p):
                    Mnp = np.array([[cos(2*pi*n/p),-sin(2*pi*n/p)],
                                    [sin(2*pi*n/p),cos(2*pi*n/p)]])
                    Mxp = np.array([[cos(2*pi*x/p),-sin(2*pi*x/p)],
                                    [sin(2*pi*x/p),cos(2*pi*x/p)]])
                    if matricesegales(Mnp,Mxp,epsilon):
                         bool = False
                    if matricesegales(Mxp,np.array([[1,0],[0,1]]),epsilon):
                         bool = False
          if bool:
               print(x, ' est un decomposant de Goldbach de ',n)
```