

## Tables rondes (Denise Vella-Chemla, 27 août 2023).

Cette petite note pour garder trace de l'implémentation informatique du Snurpf (ou système de numération par les restes dans les parties finies de  $\mathbb{N}$ ) qu'on a en tête depuis longtemps, mais dont on n'avait pas jusque-là trouvé une manière agréable de le représenter.

Chaque nombre est positionné dans le plan complexe, suivant ses restes de division euclidienne par les nombres premiers inférieurs à sa racine. On note l'ensemble des nombres premiers en question  $\mathcal{B}$  (pour base). Dans les deux exemples qui seront présentés, on aura comme bases  $\mathcal{B} = [2, 3, 5, 7]$  à la recherche des décomposants de Goldbach de  $n = 98$  et  $\mathcal{B} = [2, 3, 5]$  à la recherche des décomposants de Goldbach de  $n = 40$ .

Leila Schneps a démontré que la caractérisation (en fait triviale) qu'on avait trouvée identifie bien les décomposants de Goldbach (supérieurs à sa racine) d'un nombre  $n$  : les nombres premiers compris entre  $\sqrt{n}$  et  $n/2$  qui ne partagent aucun reste de division euclidienne avec  $n$  sont des décomposants de Goldbach de  $n$ . Voir la note là.

Pour que les illustrations soient lisibles, on trace des cercles qui servent de supports au repérage des restes de division dans le plan complexe.

Les nombres pairs seront positionnés sur des cercles du demi-plan réel supérieur (droit), les nombres impairs seront positionnés sur des cercles du demi-plan réel inférieur (gauche).

S'il s'agissait de positionner uniquement des nombres pairs et des nombres impairs, le cercle des pairs et le cercle des impairs ont leur centre sur le cercle-unité, aux positions des deux racines secondes de l'unité qui sont 1 et -1. Ils ont tous les deux un rayon de  $1/2$ .

Pour positionner les nombres de l'intervalle  $[9, 24]$ , on va dessiner 3 points sur le cercle des pairs à droite de 0 et sur le cercle à gauche de 0 qui "codent" la divisibilité par 3. Ces 6 points en tout sont sur les 2 cercles présentés au paragraphe précédent, aux positions des trois racines tierces de l'unité qui sont 1 et  $j$  et  $j^2$ . Ils ont tous un rayon de  $1/4$ .

Et ainsi de suite : enfin, pour positionner les nombres de l'intervalle  $[25, 48]$ , on va dessiner 6 cercles qui "codent" la divisibilité par 5. Leur centre sont sur les 6 cercles présentés au paragraphe précédent, aux positions des cinq racines cinquièmes de l'unité qui sont les  $\exp\left(\frac{2\pi i}{5}\right)$  du cercle-unité à traduire et réduire adéquatement sur le cercle souhaité, en fonction de la position de son centre et de la taille de son rayon. Ils ont tous un rayon de  $1/8$ . Sur ces 6 cercles seront positionnés  $30 = 6 \times 5$  points aux positions des racines 5<sup>ièmes</sup> de l'unité.

On a choisi pour les mesures des rayons les inverses des puissances de 4 pour que les nombres pairs n'aillent pas "empiéter" sur les impairs et inversement, et de même à tous les niveaux successifs, pour que les alignements verticaux soient aussi séparés qu'il est possible.

La fonction qui permet de trouver les coordonnées de l'affixe d'un point  $n$  pour la base  $\mathcal{B} = [2, 3, 5, 7]$

est :

```
def trouvey(n, c):  
    x = r2* cos(2*pi*n/2) + r3* cos(2*pi*n/3) + r5* cos(2*pi*n/5)+r7* cos(2*pi*n/7)  
    y = r2* sin(2*pi*n/2) + r3* sin(2*pi*n/3) + r5* sin(2*pi*n/5)+r7* sin(2*pi*n/7)  
    plt.plot(x, y, c, marker='o', markersize=4)  
    plt.annotate(str(n),xy=(x, y))  
    return x,y
```

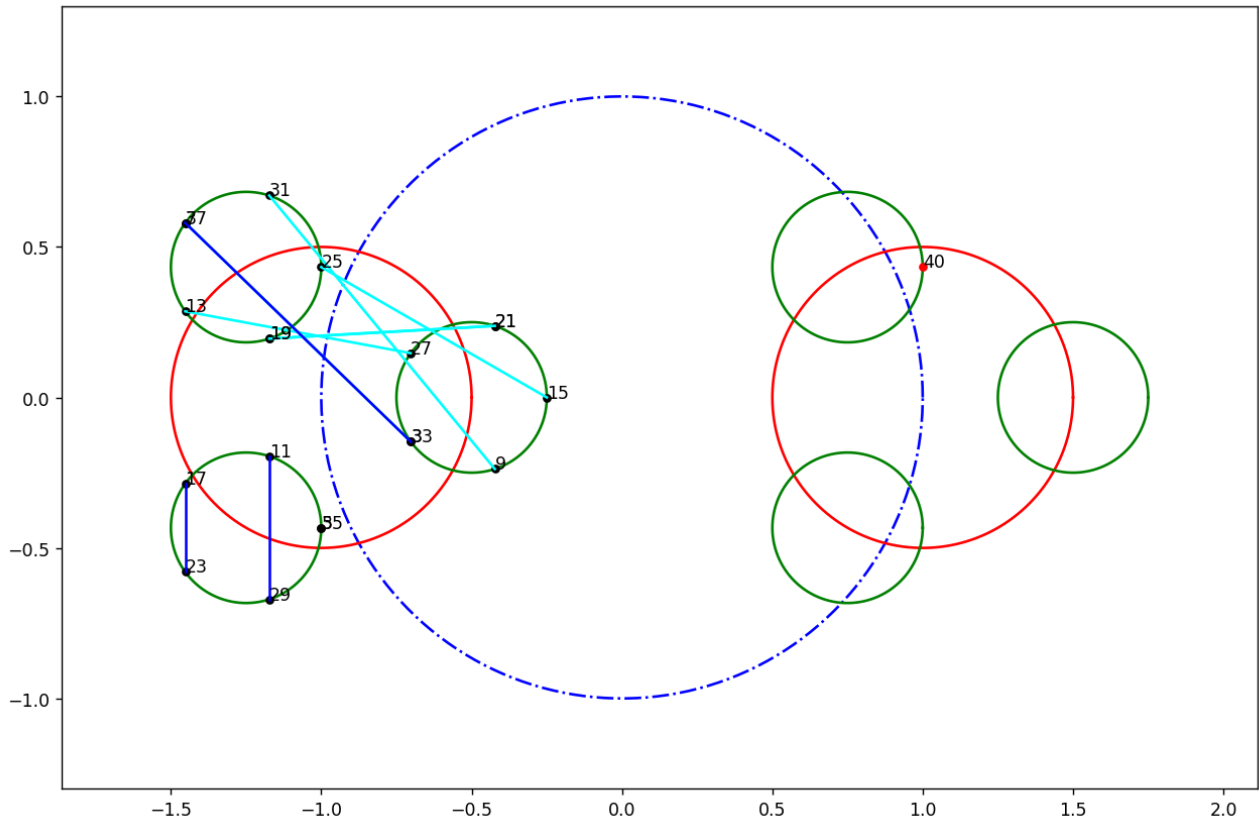


FIGURE 1 : décompositions de Goldbach de  $n = 40$  qui sont  $3 + 37, 11 + 29, 17 + 23$ .  
*Attention : 3 et 33 sont au même point.*

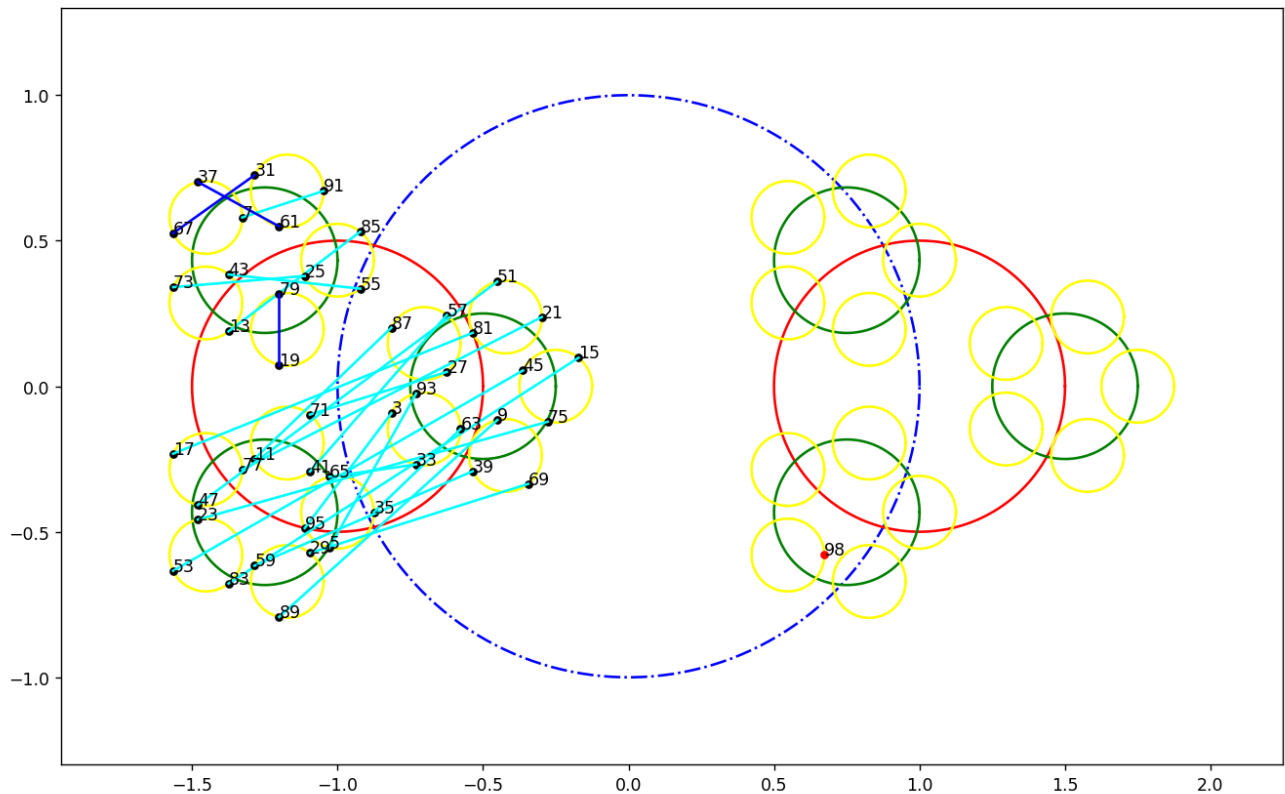


FIGURE 2 : décompositions de Goldbach de  $n = 98$  qui sont  $19 + 79, 31 + 67, 37 + 61$ .

On relie les deux sommants impairs dont la somme vaut  $n$  par un segment. On cherche à caractériser les décompositions pour distinguer les décompositions de Goldbach (en somme de deux nombres premiers) des autres décompositions (en somme contenant un nombre composé au moins). On calcule le périmètre des triangles  $[n, x, n - x]$ , leur aire, et la distance qui sépare l'abscisse de  $n$  de celle du milieu du segment  $[x, n - x]$ . On constate que les décompositions de Goldbach, du moins bon nombre d'entre elles, pour le très petit nombre d'exemples étudiés, maximisent cette dernière distance. Ci-dessous les visualisations, le programme et quelques résultats.

```
C:\Users\DENISE_2022\Desktop\conserve-banquet>python3 bodessin26à48.py
3 --> perim = 5.299449151387604 aire = 0.8361734721014444 distmilieu 2.088506694578861 DG.
5 --> perim = 4.358898943540672 aire = 7.177915983922293e-15 distmilieu 2.179449471770332
7 --> perim = 5.299449151387602 aire = 0.8361734721014387 distmilieu 2.0885066945788617
9 --> perim = 4.936769975775583 aire = 0.8978527070498491 distmilieu 1.8107359792910886
11 --> perim = 5.174325755308906 aire = 0.5166010012818898 distmilieu 2.3389792859822323 DG.
13 --> perim = 4.945570679534387 aire = 0.22568376580879224 distmilieu 2.0885066945788617
15 --> perim = 4.18890105931673 aire = 0.4330127018922135 distmilieu 1.6393596310754994
17 --> perim = 5.502637756781938 aire = 0.36034972054872655 distmilieu 2.600682775685266 DG.
19 --> perim = 4.373001009245786 aire = 0.04297380129143002 distmilieu 1.8107359792910895
```

FIGURE 3 : Chercher si les décomposants de Goldbach maximisent une distance ( $n=40$ ) (on voit que 7 et 13, non décomposants de 40, sont tout de même à grande distance du point 40).

```

C:\Users\DENISE_2022\Desktop\conserve-banquet>python3 bodessin50à120.py
3 --> perim = 3.847435987423748 aire = 0.3659990614777857 distmilieu 1.661986965969646
5 --> perim = 3.8170474067801807 aire = 0.45421096531501076 distmilieu 1.578551152375549
7 --> perim = 4.732977894233356 aire = 0.25292128870398106 distmilieu 2.2166026443472537
9 --> perim = 4.110920070470572 aire = 0.5541187174548746 distmilieu 1.5035691982829507
11 --> perim = 4.3175490668800185 aire = 0.5189700472266332 distmilieu 1.8103018041344512
13 --> perim = 4.6972136735633985 aire = 0.5241942714043881 distmilieu 2.0490163078592216
15 --> perim = 4.563054513291625 aire = 0.7349163945668143 distmilieu 1.474727402584816
17 --> perim = 4.8038231555828865 aire = 0.6439091005962185 distmilieu 1.8103018041344487
19 --> perim = 4.304722555442971 aire = 0.22829236393513905 distmilieu 2.0273688601322295 DG.
21 --> perim = 4.445655519213422 aire = 0.6751129958320433 distmilieu 1.585836481861092
23 --> perim = 4.459831057640576 aire = 0.4351742754005213 distmilieu 1.57855115237555
25 --> perim = 4.899858452924814 aire = 0.25114260371506086 distmilieu 2.2200823393022096
27 --> perim = 3.7666636961374698 aire = 0.24317470673992084 distmilieu 1.6299984322491068
29 --> perim = 3.6008008929396746 aire = 0.2125373176080315 distmilieu 1.398215937876957
31 --> perim = 5.192288882003902 aire = 0.37749019342592083 distmilieu 2.4194748168030134 DG.
33 --> perim = 3.459323320571371 aire = 0.0760149816063239 distmilieu 1.5785511523755484
35 --> perim = 3.2904815799185494 aire = 0.24325533915861183 distmilieu 1.4286124166959466
37 --> perim = 5.011263297809169 aire = 0.014536802621137362 distmilieu 2.3463220337758925 DG.
39 --> perim = 4.017618159207392 aire = 0.3040262828095534 distmilieu 1.5880941873437056
41 --> perim = 4.042155567464725 aire = 0.5426766638520251 distmilieu 1.6299984322491152
43 --> perim = 4.556207164120552 aire = 0.16827787117487994 distmilieu 2.0490163078592207
45 --> perim = 4.8378458566454885 aire = 0.7376202830851503 distmilieu 1.6619869659696487
47 --> perim = 4.908287631296902 aire = 0.914761514481756 distmilieu 1.7297877941811706
49 --> perim = 3.7741424538073134 aire = 0.0 distmilieu 1.8870712269036567

```

FIGURE 4 : Chercher si les décomposants de Goldbach maximisent une distance ( $n=98$ ) (on voit que 7, qui divise 98, mais aussi 43, qui ne le divisent, pas sont aussi à grandes distances du point associé à 98.

Résultats pour deux autres nombres :

```

C:\Users\DENISE_2022\Desktop\conserve-banquet>python3 bodessin26à48.py
3 --> perim = 4.9455706795344 aire = 0.22568376580881158 distmilieu 2.088506694578866 DG.
5 --> perim = 4.188901059316741 aire = 0.4330127018922235 distmilieu 1.6393596310755028
7 --> perim = 5.502637756781949 aire = 0.3603497205487277 distmilieu 2.6006827756852715 DG.
9 --> perim = 4.37300100924581 aire = 0.04297380129141827 distmilieu 1.8107359792910975
11 --> perim = 4.373001009245799 aire = 0.042973801291405885 distmilieu 1.810735979291095
13 --> perim = 5.502637756781961 aire = 0.36034972054874664 distmilieu 2.60068277568527 DG.
15 --> perim = 4.188901059316738 aire = 0.433012701892218 distmilieu 1.6393596310755032
17 --> perim = 4.945570679534397 aire = 0.22568376580881117 distmilieu 2.0885066945788657
19 --> perim = 5.174325755308915 aire = 0.5166010012818867 distmilieu 2.338979285982239 DG.
21 --> perim = 4.9367699757755865 aire = 0.8978527070498454 distmilieu 1.8107359792910944
23 --> perim = 5.299449151387618 aire = 0.8361734721014473 distmilieu 2.088506694578868
25 --> perim = 4.358898943540679 aire = 0.0 distmilieu 2.1794494717703397

```

FIGURE 5 : Chercher si les décomposants de Goldbach maximisent une distance ( $n=50$ )

```

C:\Users\DENISE_2022\Desktop\conserve-banquet>python3 bodessin50à120.py
3 --> perim = 3.3628009449017755 aire = 0.2646120017521994 distmilieu 1.4855798279065868
5 --> perim = 4.541123225738582 aire = 0.3763684992153665 distmilieu 2.0242308010267833 DG.
7 --> perim = 3.8480855571363684 aire = 0.48942070611572874 distmilieu 1.3187532776004725
9 --> perim = 4.0375709909459605 aire = 0.567737344108685 distmilieu 1.4374195364833826
11 --> perim = 3.9185700808569455 aire = 0.1216107825537629 distmilieu 1.8604063151079573 DG.
13 --> perim = 4.171816940287509 aire = 0.6215973127536124 distmilieu 1.5500450176145306
15 --> perim = 4.167385386113514 aire = 0.5342500353786689 distmilieu 1.282791817069301
17 --> perim = 4.587841100098727 aire = 0.49990657280154294 distmilieu 1.9166581196893075
19 --> perim = 3.7850710737352977 aire = 0.41179022488156836 distmilieu 1.5801654358412596
21 --> perim = 3.6573818931747666 aire = 0.3743063714507428 distmilieu 1.2361617634100268
23 --> perim = 4.555705534154152 aire = 0.32048769373295183 distmilieu 2.0710351189923495 DG.
25 --> perim = 3.368002110475985 aire = 0.3173979491105133 distmilieu 1.3696435770789637

```

FIGURE 6 : Chercher si les décomposants de Goldbach maximisent une distance ( $n=52$ )

Le programme python initial :

```
bodessin.py - GNU Emacs at DESKTOP-4PROGGL
File Edit Options Buffers Tools Python Isearch Help
← → × ↶ ↷ 📄 🏠
import matplotlib.pyplot as plt
import numpy as np
from numpy import pi, sin, cos
import math
from math import sqrt

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

def cercle(rayon, xcentre, ycentre, c, s):
    theta = np.linspace(0,2*pi,360)
    x = xcentre + rayon * cos(theta)
    y = ycentre + rayon * sin(theta)
    plt.plot(x,y, color=c, ls = s)

def trouveyx(n, c):
    x = r2* cos(2*pi*n/2) + r3* cos(2*pi*n/3) + r5* cos(2*pi*n/5)+r7* cos(2*pi*n/7)
    y = r2* sin(2*pi*n/2) + r3* sin(2*pi*n/3) + r5* sin(2*pi*n/5)+r7* sin(2*pi*n/7)
    plt.plot(x, y, c, marker='o', markersize=4)
    plt.annotate(str(n),xy=(x, y))
    return x,y

fig = plt.figure(figsize=(15,15))
ax = fig.gca()
ax.set_aspect('equal')
r2 = 1 ; r3 = 1/2 ; r5 = 1/4 ; r7 = 1/8
cercle(r2,0,0, 'blue', '-.-')
c3d = 1 ; c3g = c3d-2 ; c3y = 0
c5dx = 1 ; c5gx = c5dx-2 ; c5y = 0
c7dx = 1 ; c7gx = c7dx-2 ; c7y = 0
cercle(r3, c3d, c3y, 'red', '-')
cercle(r3, c3g, c3y, 'red', '-')
for m in range(3):
    cercle(r5, c5dx + r2* cos(m*2*pi/3), c5y + r2* sin(m*2*pi/3), 'green', '-')
-\\--- bodessin.py Top L28 (Python ElDoc Isearch)
```

FIGURE 7 : Début du programme

```

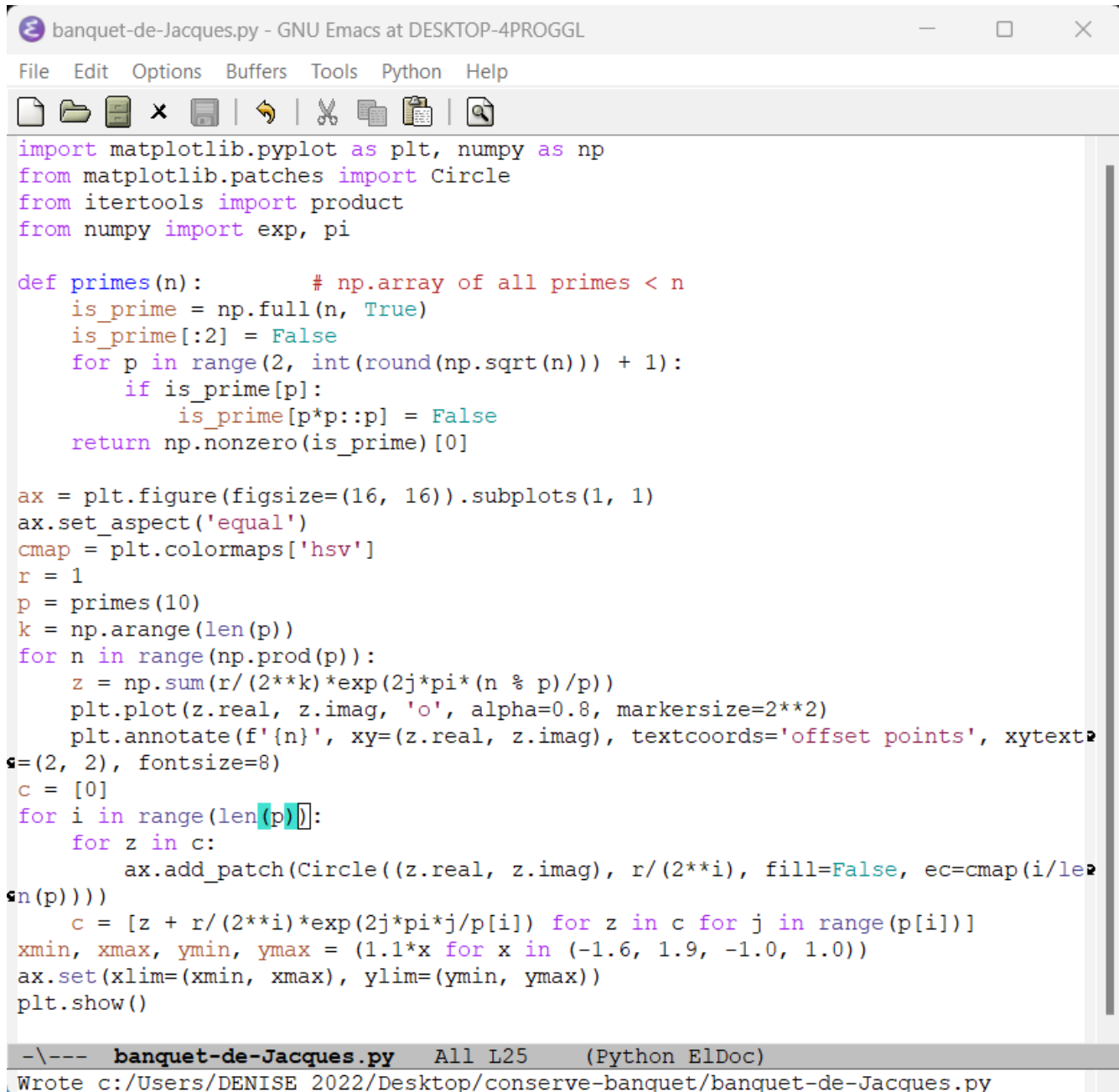
bodessin.py - GNU Emacs at DESKTOP-4PROGGL
File Edit Options Buffers Tools Python Help
[Icons]
for m in range(3):
    cercle(r5, c5dx + r3* cos(m*2*pi/3), c5y + r3* sin(m*2*pi/3), 'green', '-')
    cercle(r5, c5gx + r3* cos(m*2*pi/3), c5y + r3* sin(m*2*pi/3), 'green', '-')
    for n in range(5):
        cercle(r7, c7dx + r3* cos(m*2*pi/3) + r5* cos(n*2*pi/5), c7y + r3* sin(m*
s*2*pi/3) + r5* sin(n*2*pi/5), 'yellow', '-')
        cercle(r7, c7gx + r3* cos(m*2*pi/3) + r5* cos(n*2*pi/5), c7y + r3* sin(m*
s*2*pi/3) + r5* sin(n*2*pi/5), 'yellow', '-')
xprec, yprec = 0, 0
n = 98
for n in range(n, n+2, 2):
    xn, yn = trouvey(n, 'red')
    for d in range(3, n//2, 2):
        print(d, '--> ', end='')
        xd, yd = trouvey(d, 'black')
        xcompl, ycompl = trouvey(n-d, 'black')
        c1 = sqrt((xcompl-xd)**2+(ycompl-yd)**2)
        c2 = sqrt((xcompl-xn)**2+(ycompl-yn)**2)
        c3 = sqrt((xd-xn)**2+(yd-yn)**2)
        xmilieu=0.5*(xd+xcompl)
        ymilieu=0.5*(yd+ycompl)
        distmilieu = sqrt((xmilieu-xn)**2+(ymilieu-yn)**2)
        psur2 = 0.5*(c1+c2+c3)
        aireheron = sqrt(psur2*(psur2-c1)*(psur2-c2)*(psur2-c3))
        print('perim = ', c1+c2+c3, 'aire = ', aireheron, 'distmilieu ', distmilieu
su, end='')
        if prime(d) and prime(n-d):
            plt.plot([xd, xcompl], [yd, ycompl], 'blue')
            print(' DG. ')
        else:
            plt.plot([xd, xcompl], [yd, ycompl], 'cyan')
            print('')
#for n in range(1, 210+2):
#    if prime(n):
#        xn, yn = trouvey(n, 'cyan')
#    else:
#        xn, yn = trouvey(n, 'black')

xmin, xmax, ymin, ymax = ax.axis()
ax.set_xlim(xmin-0.2, xmax+0.2) ;
ax.set_ylim(ymin-0.2, ymax+0.2)
plt.show()
-\\*- bodessin.py Bot L46 (Python ElDoc)

```

FIGURE 8 : Fin du programme

Ci-après, un programme plus court et le placement obtenu des nombres jusqu'à  $2 \times 3 \times 5 \times 7$ .



```
import matplotlib.pyplot as plt, numpy as np
from matplotlib.patches import Circle
from itertools import product
from numpy import exp, pi

def primes(n):          # np.array of all primes < n
    is_prime = np.full(n, True)
    is_prime[:2] = False
    for p in range(2, int(round(np.sqrt(n))) + 1):
        if is_prime[p]:
            is_prime[p*p::p] = False
    return np.nonzero(is_prime)[0]

ax = plt.figure(figsize=(16, 16)).subplots(1, 1)
ax.set_aspect('equal')
cmap = plt.colormaps['hsv']
r = 1
p = primes(10)
k = np.arange(len(p))
for n in range(np.prod(p)):
    z = np.sum(r/(2**k)*exp(2j*pi*(n % p)/p))
    plt.plot(z.real, z.imag, 'o', alpha=0.8, markersize=2**2)
    plt.annotate(f'{n}', xy=(z.real, z.imag), textcoords='offset points', xytext=
(2, 2), fontsize=8)
    c = [0]
    for i in range(len(p)):
        for z in c:
            ax.add_patch(Circle((z.real, z.imag), r/(2**i), fill=False, ec=cmap(i/le
n(p))))
            c = [z + r/(2**i)*exp(2j*pi*j/p[i]) for z in c for j in range(p[i])]
xmin, xmax, ymin, ymax = (1.1*x for x in (-1.6, 1.9, -1.0, 1.0))
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
plt.show()

-\\--- banquet-de-Jacques.py  All L25  (Python ElDoc)
Wrote c:/Users/DENISE 2022/Desktop/conservé-banquet/banquet-de-Jacques.py
```

FIGURE 9 : Programme (numpy)

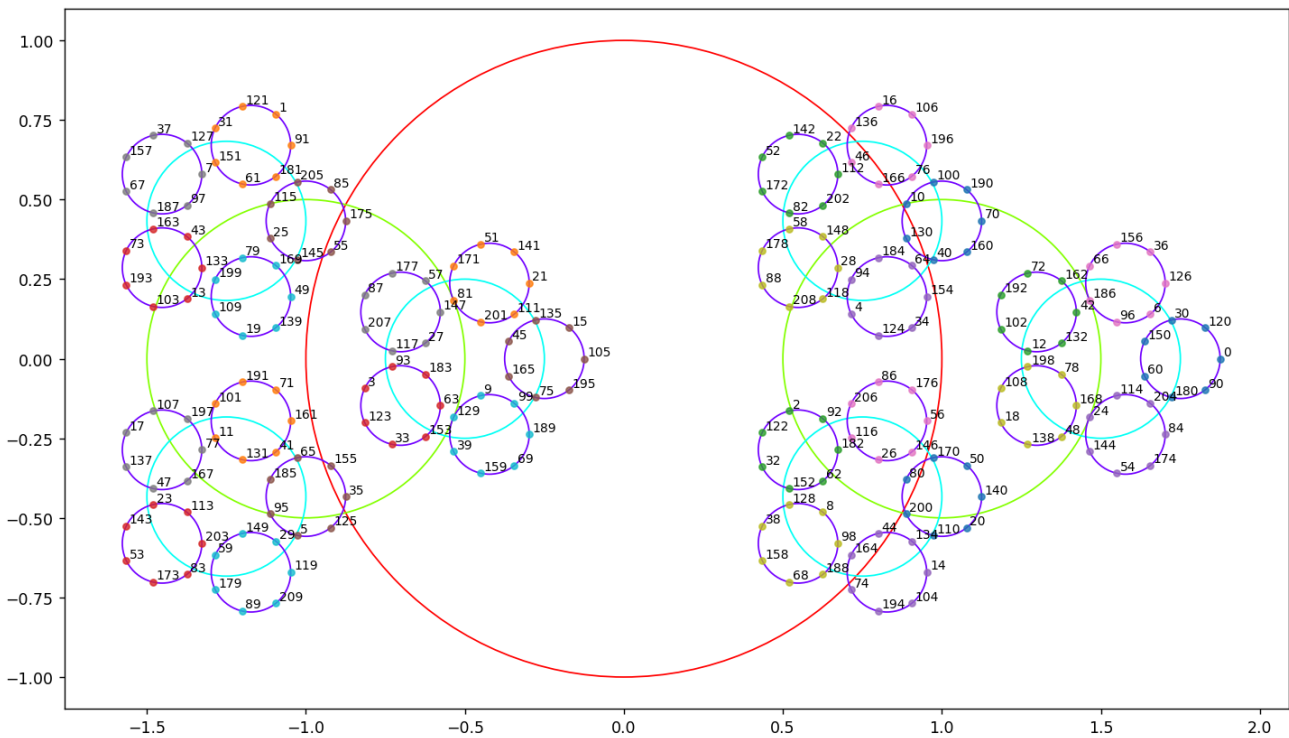


FIGURE 10 : Placement des 210 premiers nombres

Mais il y a quand même un moment où il faut réussir à se détacher des images, parce qu'elles deviennent un peu confuses, même si jolies (ci-dessous, le placement des  $2 \times 3 \times 5 \times 7 \times 11$ ).

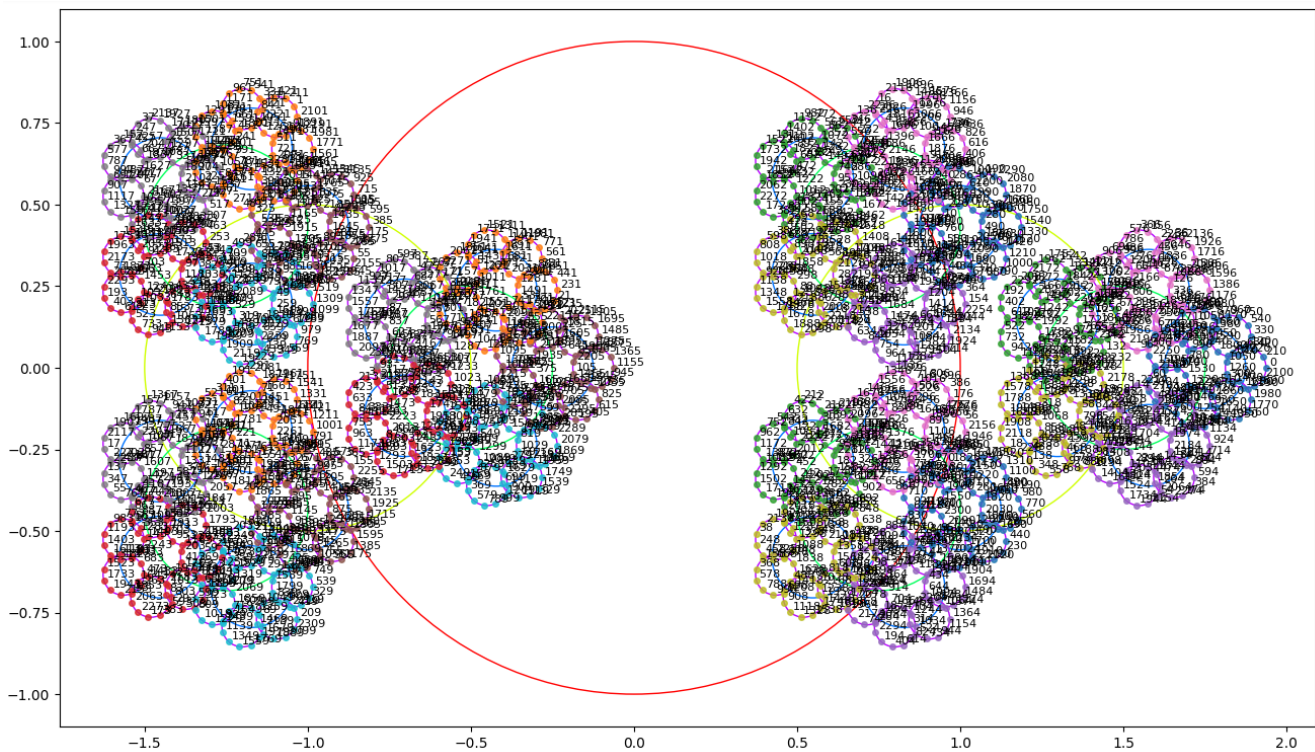


FIGURE 11 : Placement des 2310 premiers nombres