

## Snurpf et Diagrammes de Voronoi, Denise Vella-Chemla, mai 2026

On ne rappelle pas ce qu'est le Snurpf, un<sup>1</sup> système de numération par les restes dans les parties finies de  $\mathbb{N}$ . La note qui en montre l'application à la conjecture de Goldbach est là :

<https://denisevellachemla.eu/snurpf-exemple.pdf>.

On peut se reporter également aux livres suivants, pour voir que d'autres avaient pensé à ce Snurpf, i.e. au système de numération simultané multi-bases (Knuth, TAOCP, vol. 2, § 4.3.2, ou bien Knuth, Graham et Patashnik *Concrete mathematics, a foundation for computer science*, § 4.7, *Independent residues*).

### 1. Le prompt à gemini

Bonjour, Voici le programme que je souhaite que tu écrives, inspiré de ceux d'hier : pour tous les nombres pairs  $n$  variant de 6 à 102, tu vas pour chacun d'eux, trouver tous les nombres premiers inférieurs ou égaux à la racine carrée de  $n$ . Tu stockes les nombres premiers en question dans une liste  $P_n$ . Ensuite dans une seconde liste : tu stockes non seulement tous ces nombres mais également tous leurs multiples possibles plus petits que  $n$ . Tu m'affiches cette liste qu'on va appeler  $L_n$  et enfin, tu ajoutes dans cette liste, tous les éléments de la forme  $px + k$  avec  $k$  le reste modulaire de  $n$  modulo  $p$  (prenons l'exemple  $n = 98$ ) dans la liste  $L_n$ , tu aurais 2, 3, 5, 7 d'abord (les nombres premiers inférieurs ou égaux à  $\sqrt{98} = 9, \dots$  Tu aurais aussi les multiples de 2 que sont 4, 6, 8, ..., 96, tu aurais aussi les multiples de 3 que sont 3, 6, 9, ... 96, ainsi que les multiples de 5 ainsi que les multiples de 7, de 7 jusqu'à 91. Mais ensuite il te faut ajouter à la liste  $L_n$  les "congrus à  $n$  modulo  $p$ " ( $p=2,3,5,7$ ). Modulo 2, tu oublies parce que tu as déjà ajouté les nombres pairs. Modulo 3,  $98 = 2 \pmod{3}$ , donc tu dois ajouter à ta liste qui n'étaient pas déjà dedans les nombres 2, 5, 8, 11, 14, 17, etc, sauts de 3 en 3 jusqu'à 98, le dernier  $3x + 2$  que tu pourrais avoir à rajouter serait  $95=93+2$ . Modulo 5,  $98 = 3$ , tu dois ajouter à ta liste les nombres 3, 8, 13, 18, 23, 28 s'ils ne sont pas déjà dans la liste, et modulo 7 tu n'as personne à rajouter à ta liste car 7 est un diviseur de 98. Une fois que tu as cette liste définitive, je voudrais que tu me fasses un voronoi carré avec cette liste de germes et verticaux et horizontaux avec distance euclidienne et tu me l'affiches. Merci.

*Le programme*

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.spatial import Voronoi, voronoi_plot_2d
4 from scipy.spatial import cKDTree
5
6 def est_premier(num):
7     if num < 2:
8         return False
9     for i in range(2, int(np.sqrt(num)) + 1):
10         if num % i == 0:
11             return False
12     return True
```

---

1. "mon"

```

13
14 for n in range(6, 103, 2):
15     limite_sqrt = int(np.sqrt(n))
16     P_n = [p for p in range(2, limite_sqrt + 1) if est_premier(p)]
17     L_n_set = set()
18     for p in P_n:
19         multiple = p
20         while multiple < n:
21             L_n_set.add(multiple)
22             multiple += p
23     for p in P_n:
24         k = n % p # Le reste modulaire
25         valeur = k
26         while valeur <= n:
27             L_n_set.add(valeur)
28             valeur += p
29     L_n = sorted(list(L_n_set))
30     print(f"n = {n} | P_n = {P_n} | L_n = {L_n}")
31     germes = []
32     for x in L_n:
33         for y in L_n:
34             germes.append((x, y))
35     germes = np.array(germes)
36     resolution = 400
37     x_grille = np.linspace(0, n, resolution)
38     y_grille = np.linspace(0, n, resolution)
39     X, Y = np.meshgrid(x_grille, y_grille)
40     points_grille = np.vstack([X.ravel(), Y.ravel()]).T
41     arbre_kd = cKDTree(germes)
42     _, regions_pixels = arbre_kd.query(points_grille)
43     regions_pixels = regions_pixels.reshape(X.shape)
44     plt.figure(figsize=(8, 8))
45     plt.imshow(regions_pixels, extent=[0, n, 0, n], origin='lower', cmap='prism',
46               alpha=0.5)
47     if len(germes) > 3: # Securite pour scipy.spatial.Voronoi
48         vor = Voronoi(germes)
49         voronoi_plot_2d(vor, show_points=False, show_vertices=False,
50                         line_colors='black', line_width=1.0, ax=plt.gca())
51     plt.scatter(germes[:, 0], germes[:, 1], c='white', edgecolors='black', s
52               =15, zorder=3)
53     plt.xlim(0, n)
54     plt.ylim(0, n)
55     plt.xticks(range(0, n + 1, max(1, n // 10)))
56     plt.yticks(range(0, n + 1, max(1, n // 10)))
57     plt.grid(True, color='gray', linestyle=':', linewidth=0.5, alpha=0.5)
58     plt.title(f"Voronoi Euclidien pour n = {n} (Carre {n}x{n})", fontsize=12,
59             fontweight='bold')
60     #plt.show()
61     nomfic = 'snurpfvoronoi'+str(n)
62     plt.savefig(nomfic)
63     plt.close()
64     print('')

```

Listing 1 – Exemple de code Python

(J'ai rajouté les 3 lignes en fin de programme pour exporter les dessins).

*Remarque* : J'ai essayé le même prompt à Mistral, j'ai dû écrire moi-même la boucle sur  $n$  et le programme a planté parce que la liste n'était pas à deux dimensions dans l'appel à Voronoi. Je laisse cette IA de côté pour l'instant<sup>2</sup>.

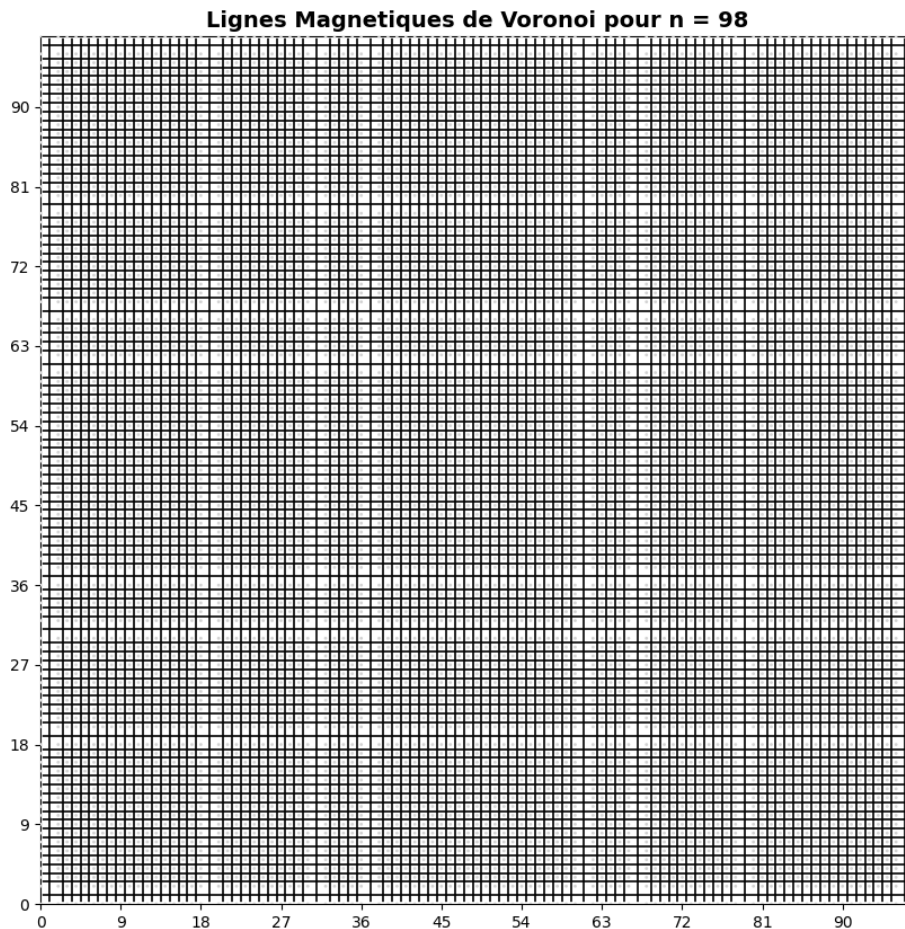
On peut regarder les dessins résultats du programme ici :

<https://denisevellachemla.eu/snurpfvoroigoemini.pdf>.

C'est joli, mais vite illisible ; alors on rajoute ce prompt.

Pourrais-tu dessiner les germes en beaucoup plus petit pour voir, parce que là, les dessins, à partir de 50, ça devient pas trop lisible. En fait, je ne sais pas si tu vois, mais il y a des lignes magnétiques, là où les germes "se repoussent", fais ressortir ces lignes dans des dessins noir et blanc pour voir...

Le résultat est par exemple celui-là, pour  $n = 98$ .



---

2. désir d'art de la programmation > désir de défense des intérêts propriétaires proches en termes kilométriques.

On devine bien les autoroutes pour les décomposants de Goldbach, 19, 31, 37, 61, 67 et 79. D'où ce dernier prompt qui engendre le dernier listing :

Non, ce résultat est moche, ne pourrais-tu pas ne laisser que les “autoroutes” : en calculant quel est la distance minimale entre deux germes et en la doublant pour être sûr que des autoroutes ne sont pas sélectionnées alors qu’elles ne devraient pas l’être ?

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.spatial import Voronoi
4
5 def est_premier(num):
6     if num < 2:
7         return False
8     for i in range(2, int(np.sqrt(num)) + 1):
9         if num % i == 0:
10            return False
11    return True
12
13 for n in range(6, 103, 2):
14     limite_sqrt = int(np.sqrt(n))
15     P_n = [p for p in range(2, limite_sqrt + 1) if est_premier(p)]
16     L_n_set = set()
17     for p in P_n:
18         multiple = p
19         while multiple < n:
20             L_n_set.add(multiple)
21             multiple += p
22     for p in P_n:
23         k = n % p
24         valeur = k
25         while valeur <= n:
26             L_n_set.add(valeur)
27             valeur += p
28     L_n = sorted(list(L_n_set))
29     germes = []
30     for x in L_n:
31         for y in L_n:
32             germes.append((x, y))
33     germes = np.array(germes)
34     if len(germes) < 4:
35         continue
36     vor = Voronoi(germes)
37     plt.figure(figsize=(10, 10), facecolor='white')
38     ax = plt.gca()
39     ax.set_facecolor('white')
40     # On filtre sur la LONGUEUR des segments de frontieres
41     # Les petites cellules ont des aretes de longueur 1.
42     # Les autoroutes creent des aretes de longueur bien plus grande.
43     seuil_longueur_arete = 1.05
44     for ridge_points, ridge_vertices in vor.ridge_dict.items():
45         # On ignore les sommets envoyes a l'infini par Scipy (indices negatifs)
46         if ridge_vertices[0] >= 0 and ridge_vertices[1] >= 0:
```

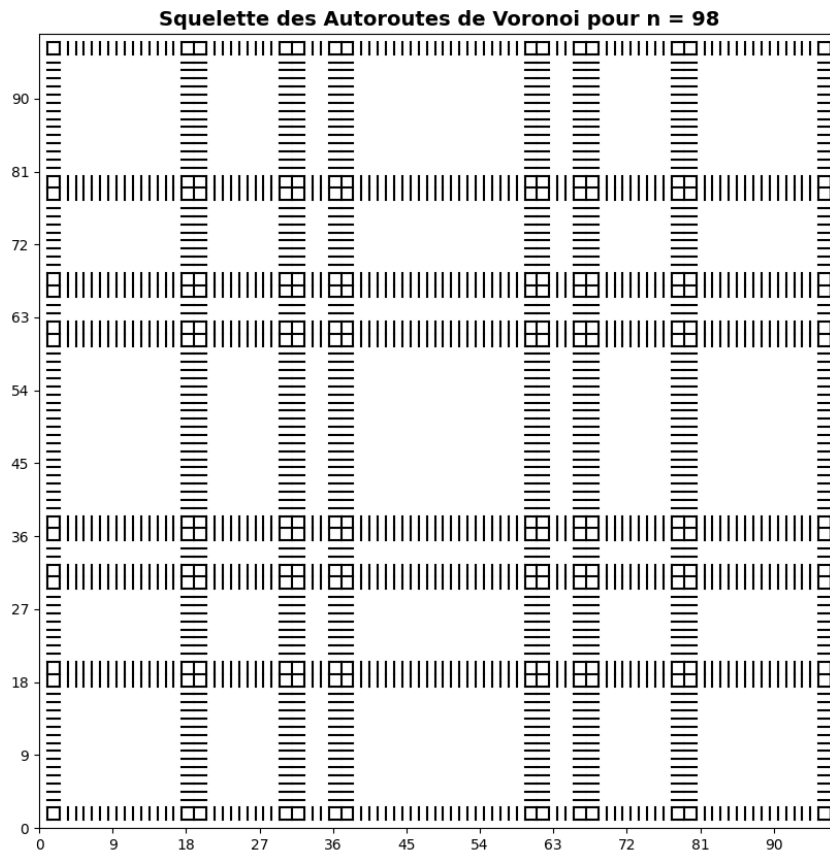
```

47     v1 = vor.vertices[ridge_vertices[0]]
48     v2 = vor.vertices[ridge_vertices[1]]
49     # Calcul de la longueur du segment de frontiere de Voronoi
50     longueur_segment = np.linalg.norm(v1 - v2)
51     # On ne garde que les segments longs (ceux des autoroutes !)
52     if longueur_segment > seuil_longueur_arete:
53         plt.plot([v1[0], v2[0]], [v1[1], v2[1]], color='black',
54                 linewidth=1.5)
55
56 plt.xlim(0, n)
57 plt.ylim(0, n)
58 plt.xticks(range(0, n + 1, max(2, n // 10)))
59 plt.yticks(range(0, n + 1, max(2, n // 10)))
60 plt.grid(False) # On enleve la grille pour ne pas confondre avec nos lignes
61 plt.title(f"Squelette des Autoroutes de Voronoi pour n = {n}", fontsize=14,
62         fontweight='bold')
63 #plt.show()
64 nomfic = 'snurpfvoronoider'+str(n)
65 plt.savefig(nomfic)
66 plt.close()
67 print('')

```

Listing 2 – Exemple de code Python

Et là, le résultat est tip-top, je trouve : par exemple, pour  $n = 98$ .



Le reste des résultats est stocké là : <https://snurpfvoronoider.pdf>.

Il fallait donc tester que la distance entre 2 germes soit supérieure à une distance unité, en quelque sorte, pour voir les décomposants de Goldbach apparaître, on ne sait pas prouver qu'il y a forcément une "autoroute" mais cette approche par les diagrammes de Voronoi est pédagogique, il me semble, ou du moins plus pédagogique que les phrases, congru à ci, incongru à ça, etc. que j'ai longtemps utilisées.

Pour sûr, le métier de programmeuse a définitivement changé, et celui de conceptrice de programmes aussi !