Les codes des cinq ia pour dessiner mon treillis Goldbach
Denise Vella-Chemla, juillet 2026

On fournit les 5 codes qu'on a demandés aux ia et qui ont pour résultat d'exécution le dessin de notre treillis pour la conjecture de Goldbach tel qu'on l'avait conçu en 2017 (voir par exemple https://denisevellachemla.eu/tauto.pdf). Mistral et deepseek ont été de loin les plus rapides pour l'obtention du code, gemini a rencontré beaucoup de difficultés et est même passé par une phase un rien délirante. Chatpgpt demande trop d'explication, il s'est avéré très lent sur cette tâche, ainsi, paradoxalement, que claude qui a la réputation d'être doué en codage. De toute façon, un seul test dans un contexte extrêmement spécifique ne peut être généralisé, mais il nous a semblé intéressant de mettre à disposition ces 5 codes, comme un état courant des capacités des ia à l'été 2026.

1. ia mistral

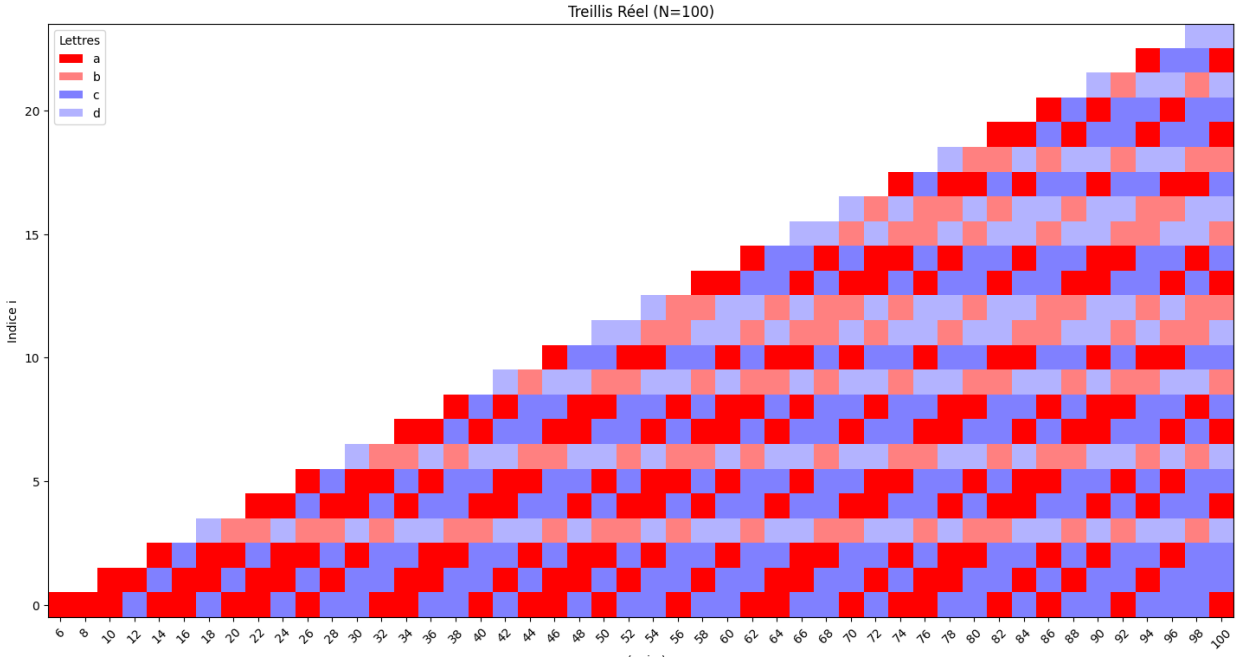
Programme

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap, BoundaryNorm
import matplotlib.patches as mpatches
import sympy as sp
import random
import math

letter_to_bits = {'a': (0, 0), 'b': (1, 0), 'c': (0, 1), 'd': (1, 1)}
bits_to_letter = {v: k for k, v in letter_to_bits.items()}

def apply_rule(lg, ld):
    u1, _ = letter_to_bits[lg]
    _, v2 = letter_to_bits[ld]
    return bits_to_letter[(u1, v2)]

def generer_sequences_reelles(N):
    impairs = list(range(3, N + 1, 2))
    abscisse = ['a' if sp.isprime(k) else 'd' for k in impairs]
    diagonale = ['a' if sp.isprime(k) else 'c' for k in impairs]
    return impairs, abscisse, diagonale

def generer_sequences_aleatoires(N, seed=42):
    random.seed(seed)
    impairs = list(range(3, N + 1, 2))
    abscisse, diagonale = [], []
    for k in impairs:
        proba = 1.0 / math.log(k) if k > 2 else 0.0
        bit = 1 if random.random() < proba else 0
        abscisse.append('a' if bit else 'd')
        diagonale.append('a' if bit else 'c')
    return impairs, abscisse, diagonale

def construire_matrice_treillis(N, abscisse, diagonale):
    n_list = list(range(6, N + 1, 2))
```

```
    max_rows = (N // 2 - 3) // 2 + 1
    matrice = np.full((max_rows, len(n_list)), np.nan)
    for col, n in enumerate(n_list):
        for i in range(len(abscisse)):
            p = 3 + 2 * i
            if p > n // 2:
                break
            q = n - p
            if q < 3 or q > N:
                continue
            j = (q - 3) // 2
            if j < 0 or j >= len(diagonale):
                continue
                matrice[i, col] =
                {'a': 0, 'b': 1, 'c': 2, 'd': 3}[apply_rule(abscisse[i],
                 diagonale[j])]
    return matrice, n_list

def afficher_treillis(matrice, n_list, titre):
    colors = [(1,0,0,1), (1,0,0,0.5), (0,0,1,0.5), (0,0,1,0.3)]
    cmap = ListedColormap(colors)
    norm = BoundaryNorm([0,1,2,3,4], cmap.N)
    ny = matrice.shape[0]
    plt.figure(figsize=(max(8, len(n_list)*0.5), max(6, ny*0.5)))
    plt.axis('equal')
    plt.imshow(matrice, aspect='equal', origin='lower', cmap=cmap,
    norm=norm, extent=[n_list[0]-0.5,
                       n_list[-1]+0.5,
                       -0.5, ny-0.5])
    plt.xlabel("n (pairs)")
    plt.ylabel("Indice i")
    plt.title(titre)
    plt.legend(handles=[mpatches.Patch(facecolor=c, label=l)
              for c,l in zip(colors, ['a','b','c','d'])],
              loc='upper left', title='Lettres')
    plt.tight_layout()
    plt.show()

def afficher_table():
    lettres = ['a', 'b', 'c', 'd']
    print("Table des 16 regles (Lignes = Seconde lettre |
          Colonnes = Premiere lettre):")
    print("R\\ | a | b | c | d")
    print("--------------------")
    for ld in lettres:
        print(f"{ld} |", end=" ")
        for lg in lettres:
            print(f" {apply_rule(lg, ld)} |", end="")
        print()

def statistiques_par_colonne(N=100):
    _, absc, diag = generer_sequences_reelles(N)
    matrice, n_list = construire_matrice_treillis(N, absc, diag)
    counts = [np.sum(matrice == i, axis=0) for i in range(4)]
    labels = ['a (premier+premier)', 'b (compose+premier)',
```

```
            'c (premier+compose)', 'd (compose+compose)']
    fig, axes = plt.subplots(4, 1, figsize=(12, 12), sharex=True)
    for i, ax in enumerate(axes):
        ax.plot(n_list, counts[i], linewidth=2)
        ax.axhline(0, color='k', linestyle=':', linewidth=1.5)
        ax.set_ylabel(f'Nombre de {labels[i][0]}')
        ax.set_title(labels[i])
        ax.grid(True, alpha=0.3)
    axes[-1].set_xlabel('n (pairs)')
    plt.suptitle('Repartition des lettres par colonne', fontsize=16,
                 fontweight='bold')
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    N = 100
    afficher_table()
    _, absc_reel, diag_reel = generer_sequences_reelles(N)
    matrice_reel, n_list = construire_matrice_treillis(N, absc_reel, diag_reel)
    afficher_treillis(matrice_reel, n_list, f"Treillis Reel (N={N})")
    _, absc_alea, diag_alea = generer_sequences_aleatoires(N, seed=42)
    matrice_alea, _ = construire_matrice_treillis(N, absc_alea, diag_alea)
    afficher_treillis(matrice_alea, n_list, f"Treillis Aleatoire (N={N})")
    statistiques_par_colonne(N)
```

Résultat d'exécution



2. ia deepseek

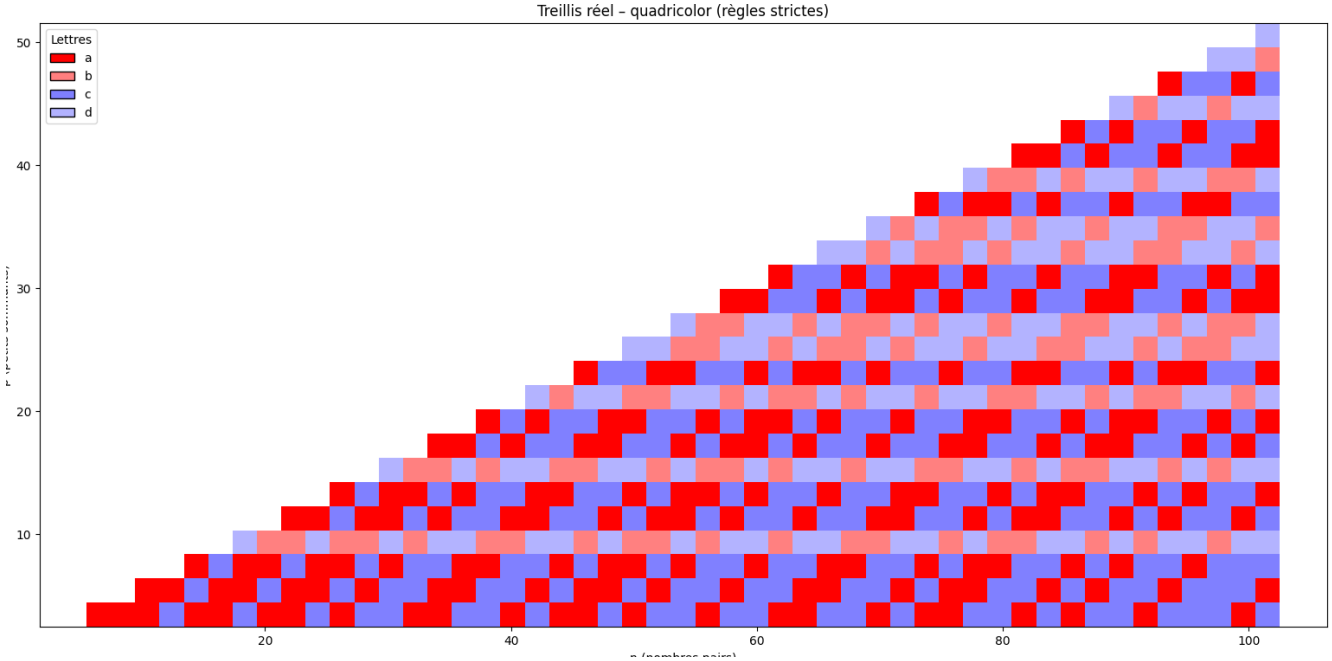
Programme

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap, BoundaryNorm
import matplotlib.patches as mpatches
import sympy as sp

# ============================================================
# DEFINITION STRICTE DES 16 REGLES (codage (u,v))
# ============================================================
letter_to_bits = {
    'a': (0, 0),
    'b': (1, 0),
    'c': (0, 1),
    'd': (1, 1)
}
bits_to_letter = {
    (0, 0): 'a',
    (1, 0): 'b',
    (0, 1): 'c',
    (1, 1): 'd'
}

def apply_rule(lettre_gauche, lettre_droite):
    """Applique R((u1,v1), (u2,v2)) = (u1, v2)."""
    u1, _ = letter_to_bits[lettre_gauche]
    _, v2 = letter_to_bits[lettre_droite]
    return bits_to_letter[(u1, v2)]

# ============================================================
# GENERATION DE LA SEQUENCE REELLE
# ============================================================
def generer_sequences_reelles(N):
"""Retourne les lettres sur l'axe abscisse (a/d) et
diagonale (a/c)."""
    impairs = list(range(3, N + 1, 2))
    abscisse = ['a' if sp.isprime(k) else 'd' for k in impairs]
    diagonale = ['a' if sp.isprime(k) else 'c' for k in impairs]
    return impairs, abscisse, diagonale

# ============================================================
# CONSTRUCTION DE LA MATRICE DU TREILLIS
# ============================================================
def construire_matrice_treillis(N, abscisse, diagonale):
    n_list = list(range(6, N + 1, 2))
    max_p = N // 2
    max_rows = (max_p - 3) // 2 + 1
    matrice = np.full((max_rows, len(n_list)), np.nan)

    for col, n in enumerate(n_list):
        for p in range(3, n // 2 + 1, 2):
```

```
            q = n - p
            if q < 3 or q > N:
                continue
            i = (p - 3) // 2
            j = (q - 3) // 2
            if i >= max_rows or j < 0 or j >= len(abscisse):
                continue

            # Lecture stricte sur les axes
            lettre_abscisse = abscisse[i]
            lettre_diagonale = diagonale[j]

            # Application de la regle
            lettre_case = apply_rule(lettre_abscisse, lettre_diagonale)
            code = {'a': 0, 'b': 1, 'c': 2, 'd': 3}[lettre_case]
            matrice[i, col] = code

    return matrice, n_list, max_p

# ============================================================
# AFFICHAGE AVEC CASES CARREES, LEGENDE EN CARRES
# ============================================================
def afficher_treillis(matrice, n_list, max_p, titre,
                      show_decomp=False, N_decomp=30):
    # Couleurs : a (rouge opaque), b (rouge alpha 0.5),
    # c (bleu alpha 0.5), d (bleu alpha 0.3)
    colors = [
        (1.0, 0.0, 0.0, 1.0),
        (1.0, 0.0, 0.0, 0.5),
        (0.0, 0.0, 1.0, 0.5),
        (0.0, 0.0, 1.0, 0.3)
    ]
    cmap = ListedColormap(colors)
    bounds = [0, 1, 2, 3, 4]
    norm = BoundaryNorm(bounds, cmap.N)

    # Ajustement pour que les cellules soient carrees
    nx = len(n_list)
    ny = matrice.shape[0]
    cell_size = 0.5
    fig_width = max(8, nx * cell_size)
    fig_height = max(6, ny * cell_size)
    plt.figure(figsize=(fig_width, fig_height))
    plt.axis('equal')
    plt.imshow(matrice, aspect='equal', origin='lower', cmap=cmap, norm=norm,
               extent=[n_list[0] - 0.5, n_list[-1] + 0.5, 2.5, max_p + 0.5])

    plt.xlabel("n (nombres pairs)", fontsize=10)
    plt.ylabel("p (petits sommants)", fontsize=10)
    plt.title(titre, fontsize=12)

    # Legende avec des patches carres (remplace la colorbar)
    legend_elements = [
        mpatches.Patch(facecolor=colors[0], edgecolor='black', label='a'),
        mpatches.Patch(facecolor=colors[1], edgecolor='black', label='b'),
```

```
        mpatches.Patch(facecolor=colors[2], edgecolor='black', label='c'),
        mpatches.Patch(facecolor=colors[3], edgecolor='black', label='d')
    ]
    plt.legend(handles=legend_elements, loc='upper left', title='Lettres')

    # OPTION : afficher les decompositions p+q en tres petit
    # Decommentez les lignes ci-dessous pour activer
    if show_decomp:
        for col, n in enumerate(n_list):
            if n > N_decomp:
                continue
            for i in range(matrice.shape[0]):
                code = matrice[i, col]
                if not np.isnan(code):
                    p = 3 + 2 * i
                    q = n - p
                    if q >= 3 and q <= N_decomp * 2:  # limite arbitraire
                        plt.text(n, p, f'{p}+{q}',
                                 fontsize=4, ha='center', va='center',
                                 color='white', weight='bold')

    plt.tight_layout()
    plt.show()

# ============================================================
# EXECUTION
# ============================================================
if __name__ == "__main__":
    N = 102    # Modifiable
    impairs, absc, diag = generer_sequences_reelles(N)
    matrice, n_list, max_p = construire_matrice_treillis(N, absc, diag)
    afficher_treillis(matrice, n_list, max_p,
                      "Treillis reel     quadricolor (regles strictes)",
                      show_decomp=False)   # Mettre True pour voir les sommes
```

Résultat d'exécution



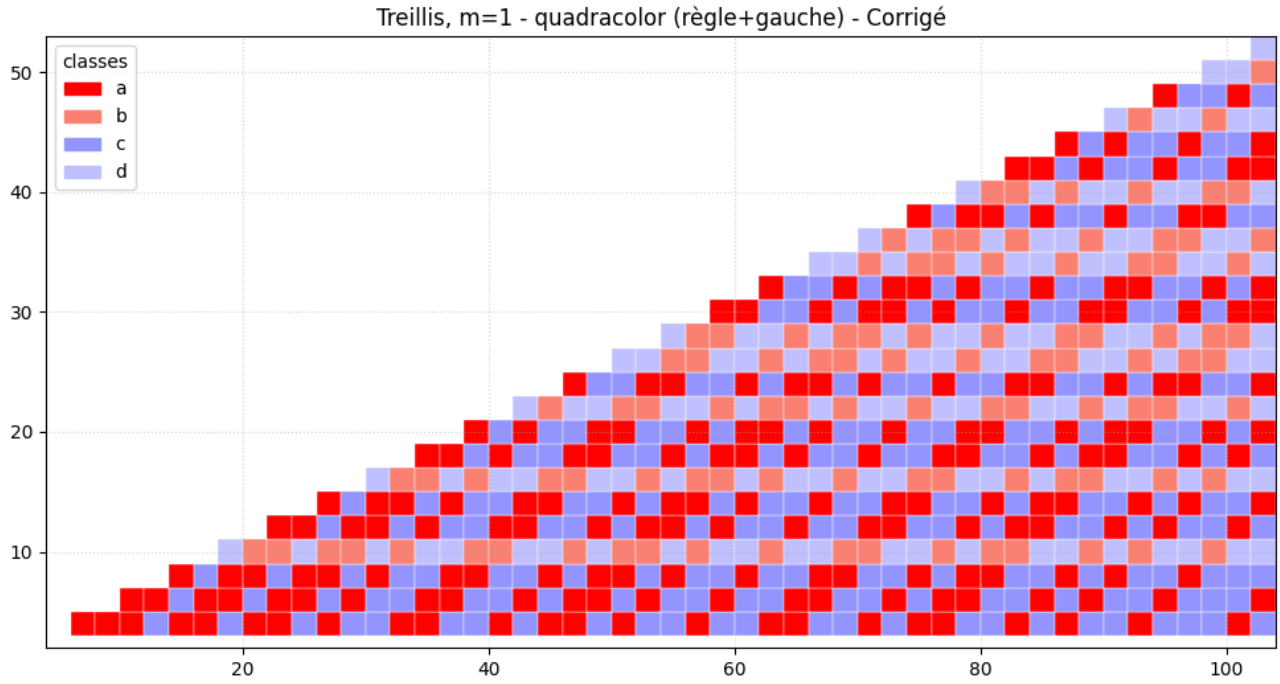
3. ia gemini

Programme

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

def is_prime(n):
    if n < 2: return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0: return False
    return True

# CORRECTION : Les couleurs sont indexees par (is_prime(j), is_prime(i))
# Ainsi, j (la diagonale) pilote la couleur de fond principale
COLORS = {
    (True, True): '#FF0000',      # a : j premier + i premier (Rouge)
    (True, False): '#FA8072',     # b : j premier + i compose (Rose)
    (False, True): '#9494FF',     # c : j compose + i premier (Bleu)
    (False, False): '#C0C0FF'     # d : j compose + i compose (Bleu clair)
}

def draw_goldbach_trellis_final(n_max=102):
    fig, ax = plt.subplots(figsize=(12, 6))

    # Parcours des nombres pairs n (Axe X)
    for n in range(6, n_max + 1, 2):
        # Parcours des termes impairs i sous l'escalier (Axe Y)
        for i in range(3, n // 2 + 1, 2):
            j = n - i
```

```
            # Application de la correction sur le couple (j, i)
            color = COLORS[(is_prime(j), is_prime(i))]

            # Dessin du pixel
            rect = plt.Rectangle((n, i), 2, 2, facecolor=color,
                                 edgecolor='white', linewidth=0.3)
            ax.add_patch(rect)

    # Ajustements graphiques identiques au modele
    ax.set_xlim(4, n_max + 2)
    ax.set_ylim(2, (n_max // 2) + 2)

    # Legende officielle
    labels = {
        'a': '#FF0000',
        'b': '#FA8072',
        'c': '#9494FF',
        'd': '#C0C0FF'
    }
    patches = [mpatches.Patch(color=color, label=label) for label,
                              color in labels.items()]
    ax.legend(handles=patches, title="classes", loc='upper left')

    ax.set_title("Treillis, m=1 - quadracolor (regle+gauche) - Corrige")
    plt.grid(True, which='both', linestyle=':', alpha=0.5)
    plt.show()

# Lancement du trace conforme
draw_goldbach_trellis_final(102)
```

Résultat d'exécution



4. ia claude

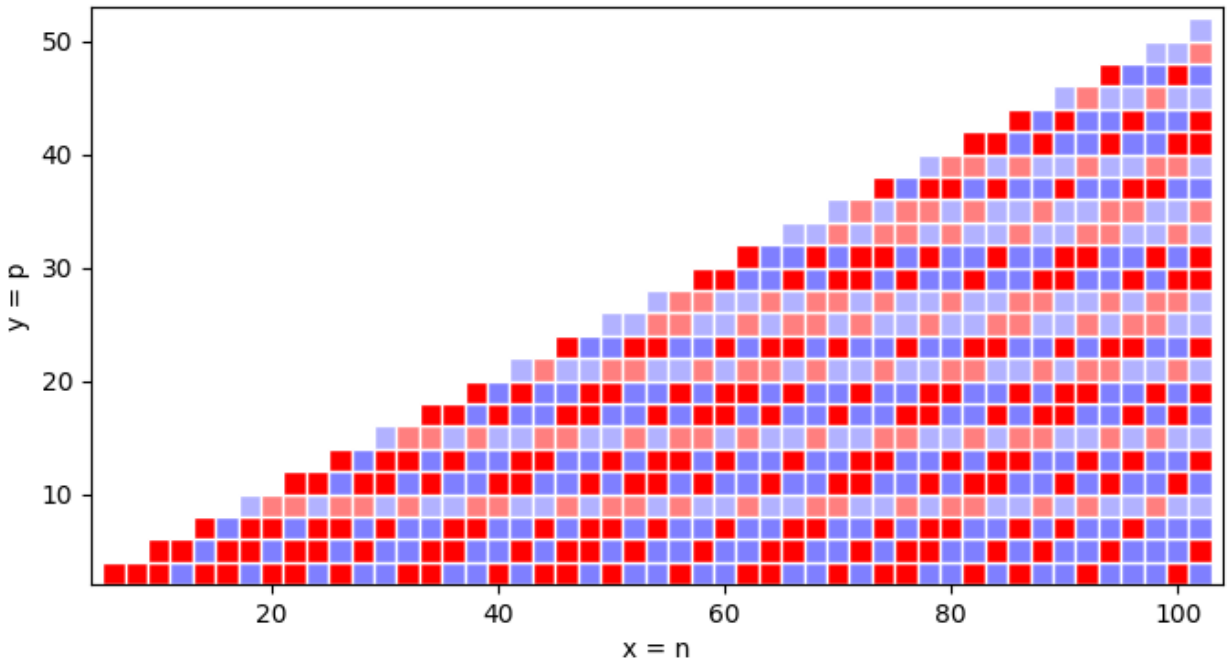
Programme

```
from sympy import isprime
import matplotlib.pyplot as plt

def is_coincidence(x, y):
    """
    x = n  (colonne : la somme totale)
    y = p  (ligne   : le premier sommant)

    Le point (x,y) est a l'intersection de 3 droites simples :
        (1) verticale   : X = x           -> fixe la colonne n
        (2) horizontale : Y = y
                          -> porte le bit u = "p non premier"
        (3) diagonale   : X - Y = x - y = q
                          -> porte le bit v = "q non premier"

    Coincidence a=(0,0)  ssi  les bits portes par (2) et (3) sont nuls,
    c'est-a-dire y premier ET (x - y) premier.
    Retourne None si (x,y) est hors du triangle valide (y > x - y, i.e. p > q).
    """
    q = x - y
    if y < 3 or q < y:
        return None
    return isprime(y) and isprime(q)
```

```
def build_triangle(n_max):
    """Reconstruit la grille des deux figures : x=n pair de 6 a n_max,
    y=p impair de 3 a n/2, en ne gardant que les cellules valides
    (donc uniquement les couples sous l'escalier vert, p <= n/2)."""
    grid = {}
    for n in range(6, n_max + 1, 2):
        for p in range(3, n // 2 + 1, 2):
            grid[(n, p)] = is_coincidence(n, p)
    return grid


def check_goldbach(n_max):
    """Verifie, colonne par colonne, l'existence d'au moins une coincidence."""
    grid = build_triangle(n_max)
    for n in range(6, n_max + 1, 2):
        ps = [p for (nn, p) in grid if nn == n]
        has_a = any(grid[(n, p)] for p in ps)
        print(f"n={n:3d}  nb de sommes={len(ps):2d}  "
              f"{'OK (au moins un a)' if has_a else '*** AUCUNE COINCIDENCE ***'}")


def plot_triangle(n_max, save_path=None):
    """Reproduit visuellement la figure du modele (escalier / dessin corrige) :
    x=n, y=p, uniquement les cellules sous l'escalier vert (p <= n/2).

    Correction : la famille de couleur (rouge/bleu) doit etre determinee par
    la primalite de q = n - p, PAS par celle de p. p est constant sur toute
    une ligne (horizontale) alors que q = n - p est constant sur une diagonale
    (n et p avancent ensemble par pas de 2) : coder la famille sur p donnait
    donc des bandes horizontales ; coder la famille sur q fait apparaitre les
    bandes en diagonale, comme dans le modele.
    """
    grid = build_triangle(n_max)
    fig, ax = plt.subplots(figsize=(8, 6))

    # cle = (q premier ?, p premier ?) -> la primalite de q pilote la famille
    # rouge/bleu, celle de p pilote la nuance claire/foncee a l'interieur
    # de la famille.
    colors = {
      (True, True):   (1, 0, 0, 1),    # a : p et q premiers
                                       -> rouge plein
      (True, False):  (1, 0, 0, 0.5),  # b : p compose, q premier
                                       -> rouge clair
      (False, True):  (0, 0, 1, 0.5),  # c : p premier, q compose
                                       -> bleu clair
      (False, False): (0, 0, 1, 0.3),  # d : p et q composes
                                       -> bleu tres clair
    }

    for (n, p), _ in grid.items():
        q = n - p
        key = (isprime(q), isprime(p))
        ax.add_patch(plt.Rectangle((n - 1, p - 1), 2, 2,
```

```
                                    facecolor=colors[key], edgecolor="white"))

    ax.set_xlim(4, n_max + 2)
    ax.set_ylim(2, n_max / 2 + 2)
    ax.set_aspect("equal")
    ax.set_xlabel("x = n"); ax.set_ylabel("y = p")
    plt.show()


if __name__ == "__main__":
    check_goldbach(40)
    plot_triangle(102, save_path="/home/goldbach_triangle_xy.png")
```

Résultat d'exécution



5. ia chatgpt

L'ia reconnaît ne pas avoir été à la hauteur pour cette tâche spécifique demandée.