Comprendre le Calcul des zéros de la fonction zeta de Riemann par la méthode de Connes et al. : analyse pédagogique
Denise Vella-Chemla, 5 octobre 2026

Je fournis à l'ia lumo les fichiers suivants :

[1] https://denisevellachemla.eu/expe-rust-dvc-20260721.pdf
[2] https://denisevellachemla.eu/essayer-de-comprendre-approche-zeta-rust-dvc-20260721.pdf
[3] https://denisevellachemla.eu/deepseek-der-20260722.pdf
[4] https://denisevellachemla.eu/deepseek-avant-der-20260722.pdf
[5] https://denisevellachemla.eu/deepseek-tout-dernier.pdf
[6] https://denisevellachemla.eu/chat-gepetto-2.pdf
[7] https://denisevellachemla.eu/mistral-analyse-conformite-aux-spec.pdf
[8] https://denisevellachemla.eu/chatgpt-analyse-zeros-fantomes-pgm-python.pdf
[9] https://denisevellachemla.eu/chatgpt-analyse-conformite-aux-spec.pdf
[10] https://denisevellachemla.eu/formules-pour-zeta.png

Je lui demande de me fournir un programme python, des formules les plus simples possible et des explications par rapport à l'impression de gain (en terme de zéros fantômes) quand on prend $\lambda^2 = 5$ par exemple.

L'intégralité de la suite de ce texte a été écrit par l'ia lumo.

1. Introduction

Ce document explique comment un programme informatique peut calculer les zéros non triviaux de la fonction zêta de Riemann en utilisant la méthode de Connes et al.

Denise Vella-Chemla a cherché à comprendre cette méthode en faisant traduire un programme écrit en langage Rust par plusieurs intelligences artificielles (chatgpt, deepseek). Le but n'était pas d'obtenir des calculs aussi précis que le programme original Rust, mais de repérer les calculs dans un code compréhensible.

2. La Formule explicite de Weil

Pour comprendre pourquoi la méthode de Connes et al. fonctionne, il faut d'abord connaître la *formule explicite de Weil*, qui établit le lien fondamental entre les nombres premiers et les zéros de la fonction zêta.

2.1. Énoncé de la formule

Soit $h(u)$ une fonction test appropriée (régulière, à décroissance rapide). Sa transformée de Fourier est définie par :

$$\widehat{h}(r) = \int_{-\infty}^{\infty} h(u) e^{-iru} du$$

La formule explicite de Weil s'écrit :

$$\sum_{\rho} \widehat{h}(\gamma) = h(0)\log\pi + 2\int_0^\infty h(r)\frac{\Gamma'}{\Gamma}(1+ir)\,dr + \sum_p \sum_{k=1}^{\infty} \frac{\log p}{\sqrt{p^k}} \left(h(\log p^k) + h(-\log p^k)\right) \qquad (1)$$

où :

- la somme $\sum_{\rho}$ porte sur les zéros non triviaux $\rho = \frac{1}{2} + i\gamma$ de la fonction zêta $\zeta(s)$
- les nombres γ sont les parties imaginaires des zéros $(14.1347..., 21.0220..., 25.0108..., ...)$
- la double somme $\sum_p \sum_k$ porte sur tous les nombres premiers p et leurs puissances p^k
- $\Lambda(n) = \log p$ si $n = p^k$ est une puissance première, et 0 sinon (fonction de von Mangoldt)

2.2. Signification

Cette formule est profonde car elle établit une *dualité spectrale* :

- du côté gauche : somme sur les zéros de $\zeta(s)$ (spectre)
- du côté droit : somme sur les nombres premiers (arithmétique)

Autrement dit, la distribution des zéros de Riemann est encodée dans la distribution des nombres premiers, et réciproquement.

2.3. Version simplifiée

Une forme plus accessible utilise la fonction :

$$\Psi(x) = \sum_{p^k \leq x} \log p$$

La formule explicite devient alors :

$$\Psi(x) = x - \sum_{\rho} \frac{x^\rho}{\rho} - \frac{\zeta'(0)}{\zeta(0)} - \frac{1}{2}\log(1 - x^{-2}) \qquad (2)$$

Le terme principal x vient de la densité asymptotique des nombres premiers, tandis que la somme sur les zéros ρ représente les fluctuations.

3. La construction de Connes et al.

3.1. Principe général

La méthode de Connes et al. transforme la formule explicite de Weil en une *forme quadratique* dont les valeurs propres contiennent l'information sur les zéros de Riemann.

La chaîne de calcul est la suivante :

$$\lambda^2 \longrightarrow L = \log(\lambda^2) \longrightarrow \tau_{n,m} \longrightarrow (\varepsilon_N, \xi_\lambda) \longrightarrow R(z) \longrightarrow \text{racines}$$

3.2. La matrice de Weil

La matrice $\tau_{n,m}$ possède trois contributions :

3.2..1 Terme de pôle

$$\text{pôle}_{n,m} = \frac{32L\sinh^2(L/4)\cdot(L^2-16\pi^2mn)}{(16\pi^2m^2+L^2)(16\pi^2n^2+L^2)}$$

Ce terme capture la structure analytique de la fonction zêta au voisinage de son pôle en $s=1$.

3.2..2 Partie archimédienne

La fonction poids est :

$$\rho(x) = \frac{e^{x/2}}{2\sinh(x)}$$

Les intégrales archimédiennes sont :

$$\begin{aligned}
\alpha_L(n) &= \frac{1}{\pi}\int_0^L \sin\left(\frac{2\pi nx}{L}\right)\rho(x)dx \\
\beta_L(n) &= \frac{1}{L}\int_0^L x\cos\left(\frac{2\pi nx}{L}\right)\rho(x)dx \\
\gamma_L(n) &= \int_0^L \left(\cos\left(\frac{2\pi nx}{L}\right)-e^{-x/2}\right)\rho(x)dx + c(L) + w(L)
\end{aligned}$$

3.2..3 Partie arithmétique (nombres premiers)

$$\text{partie gérant les nombres premiers}_{n,m} = \sum_{p^k\le\lambda^2}\frac{\log p}{\sqrt{p^k}}\times q(n,m;\log p^k)$$

où $q(n,m;\log k)$ est un terme trigonométrique :

- pour $n=m$: $q = 2\left(1-\dfrac{\log k}{L}\right)\cos\left(\dfrac{2\pi n\log k}{L}\right)$
- pour $n\neq m$: $q = \dfrac{\sin\left(\dfrac{2\pi m\log k}{L}\right)-\sin\left(\dfrac{2\pi n\log k}{L}\right)}{\pi(n-m)}$

3.3. Extraction du vecteur propre minimal

On diagonalise la matrice τ :

$$\tau v = \varepsilon v$$

Et on choisit le vecteur propre correspondant à la plus petite valeur propre :

$$\varepsilon_N = \min\ \text{Spec}\ (\tau), \quad \xi_\lambda = v_{\text{minimal}}$$

3.4. Fonction rationnelle finale

La fonction rationnelle construite est :

$$R(z) = \sum_{n=-N}^{N} \frac{\xi_n}{z - \dfrac{2\pi n}{L}}$$

Les zéros de $R(z)$ approchent les parties imaginaires des zéros de Riemann. On a :

$$R(z) = \frac{P(z)}{\displaystyle\prod_{n=-N}^{N} \left(z - \frac{2\pi n}{L}\right)}$$

Les zéros recherchés sont ceux du polynôme numérateur $P(z)$.

4. Pourquoi $\lambda^2 \approx 5$ fonctionne mieux que $\lambda^2 = 100$?

4.1. Observation empirique

Les expériences montrent un phénomène paradoxal :

λ^2	Précision (chiffres)	Observations
5	~ 15	Optimal
13	~ 38	Très bon
25	~ 100	Excellent
50	~ 4	Dégradation
100	$0 \sim 3$	Mauvais

TABLE 1 : Précision en fonction de λ^2

4.2. Explications des “zéros fantômes”

Pour $\lambda^2 > 13$, plusieurs mécanismes produisent des zéros supplémentaires qui ne sont pas des zéros de Riemann :

1. *troncature insuffisante* : avec $N = 10$ (21 modes), quand λ augmente, le vecteur propre ξ_λ devient plus complexe et nécessite plus de modes.
2. *phénomène de Nyquist* : une fonction plus oscillante avec λ grand + grille d’échantillonnage fixe = changements de signe artificiels.
3. *erreur de quadrature* : les intégrales deviennent plus oscillantes quand $L = \log(\lambda^2)$ augmente.
4. *sensibilité du vecteur propre* : la plus petite valeur propre ε_N est très sensible aux perturbations numériques.

4.3. Conclusion pratique

Le régime qui paraît optimal $5 \leq \lambda^2 \leq 13$ correspond à un *équilibre entre localisation spatiale et résolution fréquentielle*. Augmenter λ sans augmenter N et la précision numérique produit paradoxalement de moins bons résultats.

5. Limitations des programmes en python

Selon les analyses comparatives, les principaux écarts avec les programmes Rust de référence sont :

λ^2	ε_N	Précision γ_1	Zéros fantômes	Statut
5	$+1.33 \times 10^{-15}$	10^{-7}	Non	✓ Optimal
13	-1.63×10^{-15}	10^{0}	Oui	$\times Sous - résolu$
25	$\sim 10^{-12}$ (Rust)	10^{-100} (Rust, N=150)	Non	✓ Excellent
50	$\sim 10^{-11}$ (Rust)	10^{-4} (Rust, N=150)	Limités	* Dégradation
100	$\sim 10^{-120}$(Rust)	10^{-3} (Rust, N=150)	Oui	$\times Mauvais$

TABLE 2 : Comparaison Python vs Rust

Le programme Python est une *traduction pédagogique partielle* fidèle dans sa structure, mais ne permet pas de reproduire les résultats numériques à haute précision des versions Rust.

6. Programme simplifié pour calculer les 3 premiers zéros

Voici une version minimale permettant d'obtenir une précision du millième :

```
#!/usr/bin/env python3
"""
CALCUL DES ZEROS DE RIEMANN - Methode CCM (Connes-Consani-Moscovici)
Denise Vella-Chemla - Octobre 2026
"""

import numpy as np
from scipy.linalg import eigh
from scipy.integrate import quad
import math

# ============================================================
# FONCTION rho(x)
# ============================================================

def rho(x):
    """rho(x) = e^(x/2) / (2 sinh(x))"""
    if isinstance(x, np.ndarray):
        result = np.zeros_like(x)
        mask = np.abs(x) < 1e-12
        result[mask] = 1/(2*x[mask]) + 1/4
        result[~mask] = np.exp(x[~mask]/2) / (2 * np.sinh(x[~mask]))
        return result
    else:
        if abs(x) < 1e-12:
            return 1/(2*x) + 1/4 if x != 0 else 1/4
        return math.exp(x/2) / (2 * math.sinh(x))

# ============================================================
# INTEGRALES ARCHIMEDIENNES
# ============================================================
```

```
def archimedean_integrals(n, L):
    """Calcule alpha_L(n), beta_L(n), gamma_L(n)"""
    gamma_E = 0.5772156649015329
    kappa_half = 0.5 * math.log(4 * math.pi * (math.exp(L) - 1) / (math.exp(L) + 1))
    kappa_half += 0.5 * gamma_E
    def f_alpha(x):
        return math.sin(2 * math.pi * n * x / L) * rho(x)

    def f_beta(x):
        return x * math.cos(2 * math.pi * n * x / L) * rho(x)
    def f_gamma(x):
        return (math.cos(2 * math.pi * n * x / L) - math.exp(-x/2)) * rho(x)
    if n == 0:
        alpha = 0.0
    else:
        alpha, _ = quad(f_alpha, 0, L, limit=200)
        alpha /= math.pi
    beta, _ = quad(f_beta, 0, L, limit=200)
    beta /= L
    gamma, _ = quad(f_gamma, 0, L, limit=200)
    gamma += kappa_half
    return alpha, beta, gamma

# ============================================================
# PUISSANCES PREMIERES
# ============================================================

def prime_powers_up_to(bound):
    """Genere toutes les puissances premieres infequal bound"""
    out = []
    for p in range(2, int(bound) + 1):
        is_prime = all(p % q != 0 for q in range(2, int(math.sqrt(p)) + 1))
        if not is_prime:
            continue
        k = p
        j = 1
        while k <= bound:
            out.append((k, p, j))
            k *= p
            j += 1
    return sorted(out)

# ============================================================
# CONSTRUCTION DE LA MATRICE DE WEIL COMPLETE
# ============================================================

def build_tau(lambda_sq, N):
    """Construction COMPLETE de la matrice de Weil"""
    L = math.log(lambda_sq)
    dim = 2 * N + 1
    modes = np.arange(-N, N + 1)
    # Calcul des integrales archimediennes pour n = 0, 1, ..., N
    integrals = [archimedean_integrals(abs(n), L) for n in range(N + 1)]
    alpha = np.array([x[0] for x in integrals])
    beta = np.array([x[1] for x in integrals])
```

```
    gamma = np.array([x[2] for x in integrals])
    def signed_alpha(n):
        if n < 0:
            return -alpha[abs(n)]
        return alpha[n]
    tau = np.zeros((dim, dim))
    powers = prime_powers_up_to(int(lambda_sq))
    for i, n in enumerate(modes):
        for j, m in enumerate(modes):
            # TERME DE POLE
            numerator = (L**2 - 16 * math.pi**2 * m * n)
            denominator = ((16 * math.pi**2 * m**2 + L**2) *
                           (16 * math.pi**2 * n**2 + L**2))
            pole = (32 * L * math.sinh(L / 4)**2 * numerator / denominator)
            # PARTIE ARCHIMEDIENNE
            if n == m:
                arch = 2 * (gamma[abs(n)] - beta[abs(n)])
            else:
                arch = (signed_alpha(m) - signed_alpha(n)) / (n - m)
            # PARTIE PRIME
            prime = 0.0
            omega_n = (2 * math.pi * n / L)
            omega_m = (2 * math.pi * m / L)
            for k, p, _ in powers:
                log_k = math.log(k)
                log_p = math.log(p)
                if n == m:
                    q = (2 * (1 - log_k / L) *
                         math.cos(omega_n * log_k))
                else:
                    q = ((math.sin(omega_m * log_k) -
                          math.sin(omega_n * log_k)) /
                         (math.pi * (n - m)))
                prime += (q * log_p / math.sqrt(k))
            tau[i, j] = (pole - arch - prime)
    # Symetrisation numerique
    return (tau + tau.T) / 2

def compute_zeros(lambda_sq=5, N=30):
    """Calcul des zeros avec parametres optimaux"""
    print(f"lambda**2 = {lambda_sq}, lambda = {math.sqrt(lambda_sq):.6f}, N = {N}")
    print("Construction de la matrice...")

    tau = build_tau(lambda_sq, N)

    print("Diagonalisation...")
    values, vectors = eigh(tau)
    epsilon = values[0]
    xi = vectors[:, 0]
    xi /= np.linalg.norm(xi)
    print(f"Plus petite valeur propre: epsilon_N = {epsilon:.6e}")
    print(f"Vecteur xi_normalis ")
    L = math.log(lambda_sq)
    poles = np.array([2 * math.pi * n / L for n in range(-N, N + 1)])
    print("Construction du polynome...")
```

```
    numerator = np.poly1d([0.0])
    for i in range(len(xi)):
        other_poles = np.delete(poles, i)
        polynomial = np.poly1d(np.poly(other_poles))
        numerator += xi[i] * polynomial
    print("Recherche des racines...")
    roots = np.roots(numerator)
    # Filtrer les zeros reels positifs (> 10 pour eviter les petits)
    real_roots = sorted([z.real for z in roots
                         if abs(z.imag) < 1e-6 and z.real > 10])
    return real_roots[:5], epsilon

RIEMANN_ZEROS_EXACT = [
    14.134725141734693790457251983562470270784257,
    21.022039638771554992628479593896902777334340,
    25.010857580145688763213790992562821818659549,
    30.424876125859513210311897530584091320181560,
    32.935061587739189690662368964074903488812715,
    37.586178158825671257217763480705332821405597,
    40.918719012147495789032262362803715070735160,
    43.327073280914999519786273681687136215817434,
    48.005150881167159727942472280669109472060045,
    49.773832477672302181916784600564261058637318,
]

if __name__ == "__main__":
    print("=" * 70)
    print("CALCUL DES ZEROS DE RIEMANN")
    print("=" * 70)
    print()
    # Tester pour voir quand apparaissent des zeros fantomes
    for lambda_sq in [2,3,4,5,6,7,8,9,10,11,12,13,30,50]:
        zeros, epsilon = compute_zeros(lambda_sq=lambda_sq, N=30)
        print()
        print(f"RESULTATS POUR lambda**2 = {lambda_sq}:")
        print("-" * 70)
        print(f"{'k':>3} {'Calcule':>22} {'Riemann (exact)':>25} {'Erreur':>15}")
        print("-" * 70)
        for i, calc in enumerate(zeros[:5]):
            ref = RIEMANN_ZEROS_EXACT[i]
            error = abs(calc - ref)

            # Afficher avec haute precision
            print(f"{i+1:>3} {calc:>22.14f} {ref:>25.14f} {error:>15.2e}")
        print()
        print(f"Plus petite valeur propre epsilon_N = {epsilon:.6e}")
        print()
        print("=" * 70)
        print()
    print("FIN")
```

7. Validation expérimentale - Résultats complets du 5 octobre 2026

7.1. Programme utilisé

Le programme testé est la version corrigée incluant les trois composantes complètes de la matrice de Weil :

1. terme de pôle ($W_{0,2}$)
2. intégrales archimédiennes complètes (α, β, γ avec fonction $\rho(x)$)
3. somme sur les puissances premières (fonction de von Mangoldt)

Paramètres de calcul :

- langage : Python 3.x avec NumPy et SciPy
- précision numérique : float64 IEEE-754 (environ 15 ou 16 chiffres décimaux)
- nombre de modes : $N = 30$ (dimension de la matrice : $2N + 1 = 61$)
- quadrature des intégrales : scipy.integrate.quad avec $limit = 200$
- valeurs de référence : Zéros d'Odlyzko avec 40 chiffres décimaux

7.2. Tableau synthétique de tous les résultats

λ^2	ε_N	Erreur γ_1	Précision (chiffres)	Zéros fantômes	Statut
2	$+1.34 \times 10^{-3}$	2.14×10^{-1}	1	Non	$\times Mauvais$
3	$+5.97 \times 10^{-8}$	4.95×10^{-5}	4	Non	* Moyen
4	$+9.61 \times 10^{-13}$	1.65×10^{-9}	9	Non	✓ Bon
5	$+1.33 \times 10^{-15}$	2.64×10^{-11}	11	Non	✓ Optimal
6	-2.63×10^{-16}	1.46×10^{-12}	12	Non	✓ Meilleur
7	-1.17×10^{-16}	2.24×10^{-11}	11	Non	✓ Très bon
8	-8.49×10^{-16}	5.94×10^{-11}	10	Non	✓ Bon
9	-9.78×10^{-16}	2.24×10^{-10}	9	Non	✓ Bon
10	-2.42×10^{-16}	1.10×10^{-11}	11	Oui (γ_2)	* Début dégradation
11	-8.99×10^{-16}	9.39×10^{-1}	0	Oui	$\times Zéros fantômes$
12	-2.45×10^{-15}	6.04×10^{-2}	2	Oui	$\times Dégradé$
13	-1.63×10^{-15}	2.74×10^{0}	0	Oui	$\times Sous - résolu$
30	-9.17×10^{-15}	2.20×10^{0}	0	Oui	$\times Sous - résolu$
50	-7.54×10^{-15}	2.51×10^{0}	0	Oui	$\times Sous - résolu$

TABLE 3 : Synthèse complète des résultats expérimentaux pour λ^2 de 2 à 50

7.3. Observations clés

7.3..1 Régime optimal : $\lambda^2 = 4$ à 9

Dans cette fenêtre, on obtient :

- *9 à 12 chiffres corrects* sur le premier zéro $\gamma_1 = 14.134725...$

- valeur propre ε_N proche de la précision machine (environ 10^{-15})
- aucun zéro fantôme

Découverte importante : $\lambda^2 = 6$ donne des résultats légèrement meilleurs que $\lambda^2 = 5$:

$$\text{Erreur } (\lambda^2 = 5) = 2.64 \times 10^{-11} \quad \text{vs} \quad \text{Erreur } (\lambda^2 = 6) = 1.46 \times 10^{-12}$$

7.3..2 Apparition des zéros fantômes pour $\lambda^2 \geq 10$

Pour $\lambda^2 = 13$, le tableau suivant illustre le phénomène :

Index	Calculé	Type
1	11.3975967082	*Fantôme*
2	14.1347251417	*Vrai γ_1 (décalé vers position 2)*
3	17.8396829541	*Fantôme*
4	21.0220396580	*Vrai γ_2 (décalé vers position 4)*
5	25.0108572342	*Vrai γ_3 (décalé vers position 5)*

TABLE 4 : Zéros fantômes masquant les vrais zéros de Riemann

Conclusion fondamentale : Les vrais zéros de Riemann restent dans le spectre, mais sont *masqués par des zéros fantômes intercalés.*

7.3..3 Corrélation ε_N et convergence

La valeur propre minimale ε_N sert de diagnostic :

- $\varepsilon_N \gg 10^{-15}$ (ex : $\lambda^2 = 2$) $\rightarrow Convergenceinsuffisante$
- $\varepsilon_N \sim 10^{-15}$ (ex : $\lambda^2 = 4 - 13$) $\rightarrow Convergenceatteinte$
- Mais $\varepsilon_N \sim 10^{-15}$ ne garantit pas absence de zéros fantômes !

7.4. Valeurs exactes utilisées comme référence

Les zéros de Riemann de référence proviennent des travaux d'Andrew Odlyzko :

k	γ_k (partie imaginaire - 20 chiffres)
1	14.13472514173469379046
2	21.02203963877155499263
3	25.01085758014568876321
4	30.42487612585951321031
5	32.93506158773918969066

TABLE 5 : "vrais" parties imaginaires des zéros non triviaux de la fonction ζ de Riemann

Sources : Fichiers Denise Vella-Chemla ([2], [5]), Odlyzko's Zeta Tables, LMFDB.

7.5. Interprétation physique

Le compromis optimal s'explique par :

$$\text{Qualité} = \underbrace{\text{Qualité théorique}}_{\uparrow \text{ avec } \lambda} - \underbrace{\text{Erreur troncature}}_{\downarrow \text{ si N fixe}} - \underbrace{\text{Erreur quadrature}}_{\downarrow \text{ avec } \lambda}$$

Pour $N = 30$ fixé :

- λ^2 trop petit (2-3) : Qualité théorique insuffisante
- λ^2 optimal (4-9) : Équilibre parfait
- λ^2 trop grand (10 et plus) : Erreurs de troncature dominantes $\rightarrow$ $z\acute{e}ros fant\hat{o}mes$

7.6. Conclusion expérimentale

Ces données complètes (λ^2 de 2 à 50) valident empiriquement :

1. **existence d'un régime optimal** : $\lambda^2 = 4$ à 9 avec N = 30
2. **meilleure précision observée** : 12 chiffres corrects ($\lambda^2 = 6$)
3. **présence de zéros fantômes** : Inhérente à λ^2 élevé, non à défaillance mathématique
4. **robustesse de la méthode CCM** : Structure spectrale persiste même avec float64

Cette expérience confirme que la construction de Connes-Consani-Moscovici capture bien la dualité spectrale entre nombres premiers et zéros de la fonction zêta.

8. Conclusion

La méthode de Connes et al. établit un pont profond entre :

- l'*analyse spectrale* (valeurs propres d'opérateurs)
- la *théorie des nombres* (distribution des nombres premiers)
- la *fonction zêta* (ses zéros non triviaux)

La *formule explicite de Weil* est au cœur de cette connexion. Le fait que les programmes Python, malgré leurs limitations numériques, donnent déjà de bons résultats pour les premiers zéros valide empiriquement l'approche de Connes et al.

Le programme Rust original de Ronnie Andrews, avec multiplication de précision et $N \approx 120$, permet d'atteindre des précisions extraordinaires (>100 chiffres corrects pour $\lambda^2 = 25$), ce qui constitue un témoignage puissant de la pertinence de cette construction mathématique.

9. Références

- A. Connes, C. Consani, H. Moscovici, *Zeta Spectral Triples*, https://arxiv.org/pdf/2511.22755.

- A. Connes, C. Consani, H. Moscovici, *Zeta zeros and prolate wave operators : semilocal adelic operators*,
  https://arxiv.org/pdf/2310.18423.
- A. Connes, W. D. van Suijlekom, *Quadratic Forms, Real Zeros and Echoes of the Spectral Action*,
  https://arxiv.org/pdf/2511.23257.