

Calculer les zéros de ζ par la construction de Connes et al., programme en langage python le plus simple possible
Denise Vella-Chemla, 5 octobre 2026

Je fournis à l'ia mistral les fichiers suivants :

<https://denisevella.mla.eu/expe-rust-dvc-20260721.pdf>
<https://denisevella.mla.eu/essayer-de-comprendre-approche-zeta-rust-dvc-20260721.pdf>
<https://denisevella.mla.eu/deepseek-der-20260722.pdf>
<https://denisevella.mla.eu/deepseek-avant-der-20260722.pdf>
<https://denisevella.mla.eu/deepseek-tout-dernier.pdf>
<https://denisevella.mla.eu/chat-gepetto-2.pdf>
<https://denisevella.mla.eu/mistral-analyse-conformite-aux-spec.pdf>
<https://denisevella.mla.eu/chatgpt-analyse-zeros-fantomes-pgm-python.pdf>
<https://denisevella.mla.eu/chatgpt-analyse-conformite-aux-spec.pdf>
<https://denisevella.mla.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 mistral.

1. La formule unique

Pour λ^2 entier (prendre 13), poser $L = \log \lambda^2$.

Construire la matrice symétrique τ de taille $2n + 1$ (indices $n, m = -n, \dots, n$) par la formule

$$\tau_{n,m} = \underbrace{W_{0,2}(n, m)}_{\text{pôle en } s=1} - \underbrace{W_{\mathbb{R}}(n, m)}_{\text{place réelle}} - \underbrace{\sum_{r=p^j \leq \lambda^2} \frac{\log p}{\sqrt{r}} q_{k,l}(r)}_{\text{places premières}} \quad (1)$$

avec les trois morceaux explicites :

$$W_{0,2}(n, m) = \frac{32 L \sinh^2\left(\frac{L}{4}\right) (L^2 - 16\pi^2 mn)}{(16\pi^2 m^2 + L^2) (16\pi^2 n^2 + L^2)}, \quad (2)$$

$$W_{\mathbb{R}}(n, m) = \begin{cases} \frac{\alpha(m) - \alpha(n)}{n - m}, & n \neq m, \\ 2(\gamma(n) - \beta(n)), & n = m, \end{cases} \quad (3)$$

$$q_{n,m}(k) = \begin{cases} \frac{\sin(\omega_m \log r) - \sin(\omega_n \log r)}{\pi(n-m)}, & n \neq m, \\ 2 \left(1 - \frac{\log r}{L}\right) \cos(\omega_n \log r), & n = m, \end{cases} \quad (4)$$

où $\omega_n = \frac{2\pi n}{L}$ et où les trois intégrales archimédiennes s'évaluent une fois pour toutes avec le noyau $\rho(x) = \frac{e^{x/2}}{2 \sinh x}$ (rappel : $\sinh x = \frac{1}{2}(e^x - e^{-x})$) :

$$\alpha(n) = \frac{1}{\pi} \int_0^L \sin(\omega_n x) \rho(x) dx, \quad (5)$$

$$\beta(n) = \frac{1}{L} \int_0^L x \cos(\omega_n x) \rho(x) dx, \quad (6)$$

$$\gamma(n) = \int_0^L (\cos(\omega_n x) - e^{-x/2}) \rho(x) dx + \frac{1}{2} \log \frac{4\pi(e^L - 1)}{e^L + 1} + \frac{\gamma_E}{2}. \quad (7)$$

Puis :

$$\begin{aligned} \varepsilon_n &= \text{plus petite valeur propre de } \tau, \\ \xi &= (\xi_{-n}, \dots, \xi_n) = \text{vecteur propre associé.} \end{aligned} \quad (8)$$

La formule qui donne les zéros

$$R(z) = \sum_{n=-n}^n \frac{\xi_n}{z - \frac{2\pi n}{L}} \quad (9)$$

Les zéros réels de R sont les parties imaginaires des zéros non triviaux de ζ :

$$R(t) = 0 \iff \zeta\left(\frac{1}{2} + it\right) = 0 \quad (\text{pour les premiers } t > 0). \quad (10)$$

Numériquement ($\lambda^2 = 13$, $n = 20$, double précision) :

k	zéro calculé	zéro de Riemann
1	14.134725141	14.134725142
2	21.022039639	21.022039639
3	25.010857580	25.010857580

2. Pourquoi ça marche ?

On pose

$$V_n(u) = L^{-1/2} \exp\left(\frac{2\pi i k}{L} \log u\right), \quad k = -n, \dots, n$$

La formule explicite de Weil. Le corps $\mathbb{Q}$ a trois sortes de places :

- le point $s = 1$ (pôle double de ζ),
- la place réelle $\mathbb{R}$,
- et les places premières p .

La formule explicite de Weil (1952) relie les zéros non triviaux $\rho = \frac{1}{2} + i\gamma$ à toutes ces places :

$$\sum_{\rho} \widehat{f}(\rho) = W_{0,2}(f) - W_{\mathbb{R}}(f) - \sum_{r=p^j \leq \lambda^2} \frac{\log p}{\sqrt{r}} q_{k,l}(r). \quad (11)$$

La matrice (1) n'est rien d'autre que cette identité évaluée sur la base de Fourier V_n (modes d'une fenêtre multiplicative $u \in [1/\lambda, \lambda]$, soit $x = \log u \in [-L/2, L/2]$, $L = 2 \log \lambda$) : c'est la matrice de Gram de la forme quadratique de Weil tronquée (construction de Galerkin, Connes-van Suijlekom).

Le critère spectral de l'hypothèse de Riemann. Connes a montré que l'hypothèse de Riemann équivaut au fait que la forme quadratique de Weil est positive et le noyau de la forme quadratique est engendré par la *réalisation spectrale* des zéros (les vecteurs e_{ρ}).

C'est pourquoi la plus petite valeur propre vérifie $\varepsilon_n \approx 0$: l'algorithme *teste* l'hypothèse de Riemann à chaque exécution, puis extrait du noyau le vecteur ξ .

Le décodage. La transformée de Fourier archimédienne de ξ est

$$\widehat{\xi}(z) = 2L^{-1/2} \sin\left(\frac{zL}{2}\right) R(z), \quad (12)$$

et l'équation $\widehat{\xi}(z) = 0$ est exactement l'équation $\Xi(z) = 0$ tronquée », où $\Xi(z) = \xi\left(\frac{1}{2} + iz\right)$ est la fonction majuscule de Riemann (mémoire de 1859) : $\xi(s) = \frac{1}{2}s(s-1)\pi^{-s/2}\Gamma\left(\frac{s}{2}\right)\zeta(s)$ est symétrique ($\xi(s) = \xi(1-s)$), et Ξ en est la version redressée, paire et réelle sur l'axe réel z - ses zéros sont les $\pm\gamma_k$. C'est Ξ (et non ξ) qu'approxime le programme, car il cherche des racines réelles de R sur l'axe $z = it$; c'est aussi la parité de Ξ qui explique les racines symétriques ($\pm 14,134\dots$) de ζ . Ne pas confondre avec $\Lambda(k)$, la fonction de von Mangoldt ($\Lambda(p^j) = \log p$, qui est le poids de la somme première dans la formule de la matrice τ).

Entre deux pôles consécutifs $\frac{2\pi n}{L}$, R est continue et monotone de $\pm\infty$ à $\mp\infty$: chaque intervalle contient exactement un zéro, qui coïncide avec un γ_k tant que la troncature est suffisante.

3. Rôle des paramètres, et zéros fantômes

- La coupure $p^j \leq \lambda^2 = e^L$ rend la fenêtre archimédienne ($u \leq \lambda$) et la somme première ($k \leq \lambda^2$) *auto-duales* en coordonnée logarithmique : c'est l'équilibre des trois contributions de (1) qui fait la qualité optimale autour de $\lambda^2 = 13$.

- Pour $\lambda^2 \leq 13$ à n fixé (expériences Rust, $n = 150, 100$ chiffres), l'erreur sur le premier zéro suit une loi empirique très régulière : *chaque unité de λ^2 perdue coûte environ cinq décimales* (erreur 10^{-55} à $\lambda^2 = 13$, 10^{-50} à 12 , $\dots$, 10^{-14} à 5 , 10^{-9} à 4). L'ordre de préférence est donc 13, puis 12, puis 11, etc.
- Les *zéros fantômes* qui s'intercalent pour λ^2 grand (apparus vers $\lambda^2 \geq 30$ dans les exécutions à $n = 150$) sont de vraies racines de R issues de la troncature : elles *dérivent quand n varie*, alors que les vrais zéros de ζ restent immobiles. Le critère pratique est donc la stabilité en n : un zéro authentique vérifie $t_{\text{zéro}}(n) \approx t_{\text{zéro}}(2n)$; un fantôme non. Le couple (λ^2, n) doit croître ensemble : plus la fenêtre est large, plus il faut de modes.
- Deuxième cause numérique de fantômes : si l'écart spectral $\lambda_2 - \lambda_1$ de τ tombe au niveau de la précision machine ($\sim 10^{-16}$ en double), le vecteur ξ est indéterminé et des racines apparaissent ou disparaissent au gré des arrondis (typique de $n = 10$). À $n = 20$, $\lambda^2 = 13$, les dix premiers zéros sortent à la précision machine en double précision.

4. Pourquoi moins de fantômes quand λ diminue, mais moins de précision

Expériences du 5 octobre 2026 (programme simplifié, $n = 20$, double précision, $\lambda^2 \in \{13, 7, 5, 3\}$) : plus λ^2 est petit, moins il y a de zéros parasites, mais plus les zéros calculés dérivent des vrais. Deux mécanismes indépendants, qui varient en sens opposés avec λ .

4.1. Le budget de cases : d'où viennent les fantômes

La fonction $R(z)$ possède $2n + 1$ pôles réels régulièrement espacés de $\frac{2\pi}{L}$ sur l'axe $z = it$.

Entre deux pôles consécutifs, R est continue et passe de $\pm\infty$ à $\mp\infty$: il y a donc *au plus une racine réelle par intervalle entre pôles*. Le nombre total de “cases” disponibles sur $[t_{\min}, t_{\max}]$ vaut

$$\#\text{cases} \approx (t_{\max} - t_{\min}) \cdot \frac{L}{2\pi}, \quad (13)$$

tandis que les vrais zéros de ζ en occupent seulement

$$\#\text{zéros} \approx (t_{\max} - t_{\min}) \cdot \frac{1}{2\pi} \log \frac{t}{2\pi} \quad (14)$$

(densité de Riemann-von Mangoldt ; soit $\approx 9,7$ zéros sur $[5, 55]$). Les cases excédentaires

$$G \approx \frac{t_{\max} - t_{\min}}{2\pi} \left(L - \log \frac{t}{2\pi} \right) \quad (15)$$

sont remplies par les *fantômes*.

Tableau des expériences :

λ^2	L	$2\pi/L$	cases sur $[5, 55]$	zéros vrais	fantômes
13	2,56	2,45	≈ 20	≈ 10	nombreux
7	1,95	3,23	≈ 15	≈ 10	3
5	1,61	3,90	≈ 13	≈ 10	2
3	1,10	5,72	≈ 9	≈ 10	<i>aucun</i>

Le cas $\lambda^2 = 3$ est décisif : il y a *moins de cases que de zéros* (la grille des pôles, d'espacement 5,72, est plus grossière que la distribution des zéros eux-mêmes, dont l'espacement vaut 4 à 7 dans cette zone). Aucun fantôme ne peut s'intercaler - mais deux zéros trop proches tombent dans la même case et deviennent *absents* (les zéros 5, 6, 7 et 9 manquent à l'appel). Réduire λ abaisse la résolution spectrale : moins de place pour les parasites, mais aussi moins de place pour les vrais zéros.

4.2. La précision, des contraintes qui s'opposent

Trois effets se conjuguent en sens inverse.

1. La troncature arithmétique. La somme des puissances premières est la moitié arithmétique de la formule de Weil : chaque $r = p^j$ y émet une “onde” de fréquence $\log r$ et d'amplitude $\Lambda(r)/\sqrt{r}$, et leur superposition réalise le membre de gauche (les zéros). Tronquer à λ^2 revient à retrancher de la matrice toutes les ondes d'ordre $r > \lambda^2$, soit une perturbation $\Delta\tau$ de norme comparable au poids des r omis - non négligeable, car les plus gros poids de la série ($\log 5/\sqrt{5} = 0,72$, $\log 7/\sqrt{7} = 0,93$) se trouvent dès les petits r . À $\lambda^2 = 3$, seules les ondes $r = 2$ et $r = 3$ subsistent. La plus petite valeur propre de la matrice perturbée est alors $\varepsilon_N \sim \|\Delta\tau\|$: c'est la mesure directe de la troncature ($\varepsilon_N = 6,6 \cdot 10^{-8}$ pour $\lambda^2 = 3$, contre $\sim 10^{-16}$ pour $\lambda^2 \geq 5$). Le vecteur ξ fait alors un angle avec le vrai noyau, sa transformée R est l'équation $\Xi(t) = 0$ plus un terme d'erreur $e(t) \sim \varepsilon_N$, et chaque racine dérive de $e(t^*)/\Xi'(t^*)$: quelques 10^{-5} sur γ_1 (pente raide), jusqu'à 0,49 sur γ_4 (pente plus faible) - ce sont les vrais zéros, déplacés par la perturbation.
2. ε_n en témoigne directement - c'est la jauge de qualité du calcul :

λ^2	ε_n mesuré	diagnostic
13	$-3,2 \cdot 10^{-16}$	forme dégénérée : RH testée idem idem
7	$-3,9 \cdot 10^{-16}$	
5	$+1,0 \cdot 10^{-16}$	
3	$+6,6 \cdot 10^{-8}$	le noyau est perdu : ξ n'est plus spectral

à $\lambda^2 \geq 5$, la plus petite valeur propre vaut $\sim 10^{-16}$: la forme est dégénérée au niveau de la précision machine, ξ appartient bien au noyau de Weil et les zéros sortent à 10^{-13} ou mieux. À $\lambda^2 = 3$, $\varepsilon_n = 6,6 \cdot 10^{-8}$: ξ est un vecteur propre avec un résidu substantiel, sa transformée $\hat{\xi}$ dévie de l'équation $\Xi(t) = 0$ d'une quantité $O(\varepsilon)$, et les racines partent en dérive (14,135 au lieu de 14,134 ; 30,92 au lieu de 30,42 ; 43,99 au lieu de 43,33). Les racines obtenues à $\lambda^2 = 3$ ne sont ni des fantômes ni des zéros : ce sont les *vrais zéros déplacés* par la perturbation de la forme.

3. La fenêtre archimédienne $u \in [1/\lambda, \lambda]$ est trop étroite pour que les intégrales α, β, γ captent la structure fine du facteur Γ .

4.3. Synthèse : deux courbes opposées en λ

La précision croît avec λ (plus de nombres premiers, fenêtre plus large, $\varepsilon_n \rightarrow 0$) ; le nombre de fantômes croît aussi avec λ (plus de cases que de zéros, cf. (13)-(14)). La valeur $\lambda^2 = 13$ est l'optimum : la plus grande fenêtre qui reste dans le budget de cases à $n = 20$ en double précision. Et ε_n sert de thermomètre : $\sim 10^{-16}$, tout va bien ; $\sim 10^{-8}$, le modèle ne ressemble déjà plus à ζ .

5. 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>.

6. Annexe : le programme en python de calcul des premiers zéros

```
# zeta_minimal.py
# -----
# Programme ULTRA-SIMPLIFIE, en pur Python (aucune bibliotheque :
# pas de numpy, pas de scipy, juste le module math), qui calcule
# les parties imaginaires des premiers zeros non triviaux de la
# fonction zeta de Riemann par la construction CCM
# (Connes-Consani-Moscovici, "Zeta Spectral Triples",
# arXiv:2511.22755).
#
# Chaîne de calcul (cf. le programme Rust de Ronnie Andrews et la
# traduction python de chatgpt, mais ici reduite a l'essentiel) :
#
#   lambda^2 -> L = log(lambda^2)
#   -> matrice de Weil tau (3 morceaux : pole + archimédien
#   + puissances de nombres premiers)
#   -> plus petit vecteur propre xi de tau
#   -> fonction rationnelle R(z) = somme xi_n / (z - 2*pi*n/L)
#   -> zeros reels de R = zeros de zeta sur la droite critique
#
# Precision obtenue : loin au-dela du millieme demande
# (erreur ~ 1e-10 ou mieux sur les premiers zeros), en une
# seconde environ.
# -----
import math

# ----- parametres -----
```

```

LAMBDA_SQ = 3          #  $\lambda^2$  : la "fenetre" archimediienne,  $L = \log 13$ 
N = 20                 # nombre de modes de Fourier de chaque cote -> matrice  $2N+1$ 
QUAD = 300             # ordre de la quadrature de Gauss-Legendre

L = math.log(LAMBDA_SQ)
DIM = 2 * N + 1
PI = math.pi
GAMMA_E = 0.5772156649015329    # constante d'Euler-Mascheroni

# ----- 1.  $\rho(x) = e^{\{x/2\}} / (2 \sinh x)$  (place archimediienne) -----
def rho(x):
    # au voisinage de 0 :  $\rho(x) \sim 1/(2x) + 1/4$ , mais rho n'apparait
    # jamais seule dans les integrandes (toujours multipliee par un
    # facteur qui annule la singularite), donc pas de cas special ici.
    return math.exp(x / 2) / (2 * math.sinh(x))

# ----- 2. quadrature de Gauss-Legendre sur  $[0, L]$  -----
def gauleg(n):
    # noeuds et poids sur  $[-1,1]$  par iteration de Newton sur les
    # polynomes de Legendre
    xs, ws = [], []
    for i in range(1, n + 1):
        z = math.cos(PI * (i - 0.25) / (n + 0.5))
        for _ in range(100):
            p0, p1 = 1.0, 0.0
            for k in range(1, n + 1):
                p2 = p1
                p1 = p0
                p0 = ((2 * k - 1) * z * p1 - (k - 1) * p2) / k
            dp = n * (z * p0 - p1) / (z * z - 1)
            dz = p0 / dp
            z -= dz
            if abs(dz) < 1e-15:
                break
        p0, p1 = 1.0, 0.0
        for k in range(1, n + 1):
            p2 = p1
            p1 = p0
            p0 = ((2 * k - 1) * z * p1 - (k - 1) * p2) / k
        dp = n * (z * p0 - p1) / (z * z - 1)
        xs.append(z)
        ws.append(2 / ((1 - z * z) * dp * dp))
    return xs, ws

XG, WG = gauleg(QUAD)

def gauss(f):
    # integrale de f sur  $[0, L]$  (les noeuds sont INTERIEURS a
    # l'intervalle, ce qui evite le point singulier  $x = 0$ )
    mid, half = L / 2, L / 2
    return half * sum(w * f(mid + half * x) for x, w in zip(XG, WG))

```

```

# ————— 3. les trois integrales archimediennes —————
# kappa_half corrige la diagonale (facteur Gamma de la fonction xi)
KAPPA = 0.5 * math.log(4 * PI * (math.exp(L) - 1) / (math.exp(L) + 1)) \
        + 0.5 * GAMMA_E

alpha = [0.0] * (N + 1)
beta = [0.0] * (N + 1)
gamma = [0.0] * (N + 1)
for n in range(1, N + 1):
    w = 2 * PI * n / L
    alpha[n] = gauss(lambda x: math.sin(w * x) * rho(x)) / PI
    beta[n] = gauss(lambda x: x * math.cos(w * x) * rho(x)) / L
    gamma[n] = gauss(lambda x: (math.cos(w * x) - math.exp(-x / 2))
                      * rho(x)) + KAPPA
beta[0] = gauss(lambda x: x * rho(x)) / L
gamma[0] = gauss(lambda x: (1 - math.exp(-x / 2)) * rho(x)) + KAPPA

# ————— 4. puissances de nombres premiers  $k = p^j \leq \text{lambda}^2$  —————
# avec poids  $\Lambda(k)/\sqrt{k} = \log(p)/\sqrt{k}$  (von Mangoldt)
def is_prime(p):
    for q in range(2, int(math.sqrt(p)) + 1):
        if p % q == 0:
            return False
    return p >= 2

POWERS = []
for p in range(2, LAMBDA_SQ + 1):
    if not is_prime(p):
        continue
    k = p
    while k <= LAMBDA_SQ:
        POWERS.append((k, math.log(p)))
        k *= p

# ————— 5. matrice de Weil  $\tau = \text{POLE} - \text{ARCHIMEEN} - \text{PREMIERS}$  —————
MODES = list(range(-N, N + 1))

def s_alpha(n):
    return -alpha[-n] if n < 0 else alpha[n]

tau = [[0.0] * DIM for _ in range(DIM)]
for i, n in enumerate(MODES):
    for j, m in enumerate(MODES):
        # (a) TERME DE POLE : le pole double de zeta en  $s = 1$ 
        pole = (32 * L * math.sinh(L / 4) ** 2
                 * (L * L - 16 * PI * PI * m * n)
                 / ((16 * PI * PI * m * m + L * L)
                    * (16 * PI * PI * n * n + L * L)))
        # (b) TERME ARCHIMEDIEN : la place reelle (facteur Gamma)
        if n == m:
            arch = 2 * (gamma[abs(n)] - beta[abs(n)])
        else:
            arch = (s_alpha(m) - s_alpha(n)) / (n - m)

```

```

# (c) TERME DES NOMBRES PREMIERS : les places p-adiques
w_n, w_m = 2 * PI * n / L, 2 * PI * m / L
prime = 0.0
for k, logp in POWERS:
    lk = math.log(k)
    if n == m:
        q = 2 * (1 - lk / L) * math.cos(w_n * lk)
    else:
        q = (math.sin(w_m * lk) - math.sin(w_n * lk)) \
            / (PI * (n - m))
    prime += q * logp / math.sqrt(k)
tau[i][j] = pole - arch - prime

# symétrisation (la matrice exacte est symétrique ; on nettoie les
# micro-erreurs d'arrondi)
for i in range(DIM):
    for j in range(i):
        v = (tau[i][j] + tau[j][i]) / 2
        tau[i][j] = tau[j][i] = v

# ————— 6. plus petit vecteur propre (algorithme de Jacobi) —————
def jacobi(a):
    n = len(a)
    A = [row[:] for row in a]
    V = [[1.0 if i == j else 0.0 for j in range(n)] for i in range(n)]
    for _ in range(300):
        off = sum(A[p][q] ** 2 for p in range(n) for q in range(p + 1, n))
        if off < 1e-30:
            break
    for p in range(n):
        for q in range(p + 1, n):
            if abs(A[p][q]) < 1e-300:
                continue
            th = (A[q][q] - A[p][p]) / (2 * A[p][q])
            t = math.copysign(1.0, th) / (abs(th) + math.sqrt(th * th + 1))
            c = 1 / math.sqrt(t * t + 1)
            s = t * c
            for k in range(n):
                akp, akq = A[k][p], A[k][q]
                A[k][p] = c * akp - s * akq
                A[k][q] = s * akp + c * akq
                vkp, vkq = V[k][p], V[k][q]
                V[k][p] = c * vkp - s * vkq
                V[k][q] = s * vkp + c * vkq
            for k in range(n):
                apk, aqk = A[p][k], A[q][k]
                A[p][k] = c * apk - s * aqk
                A[q][k] = s * apk + c * aqk
    vals = [A[i][i] for i in range(n)]
    return vals, V

vals, V = jacobi(tau)
i0 = min(range(DIM), key=lambda i: vals[i])
xi = [V[i][i0] for i in range(DIM)]

```

```

norm = math.sqrt(sum(v * v for v in xi))
xi = [v / norm for v in xi]

# ----- 7. fonction rationnelle  $R(z) = \text{somme } x_{i_n} / (z - 2 \pi i n / L)$  -----
POLES = [2 * PI * n / L for n in MODES]

def R(z):
    return sum(xi[i] / (z - POLES[i]) for i in range(DIM))

# ----- 8. zeros reels de R par dichotomie entre poles consecutifs -----
def bisect(a, b):
    fa, fb = R(a), R(b)
    if fa * fb > 0:
        return None
    for _ in range(100):
        m = (a + b) / 2
        fm = R(m)
        if fa * fm <= 0:
            b = m
        else:
            a = m
            fa = fm
    return (a + b) / 2

roots = []
sp = sorted(POLES)
for i in range(len(sp) - 1):
    a, b = sp[i] + 1e-9, sp[i + 1] - 1e-9
    if a <= 0 or b < 5 or a > 55:
        continue
    r = bisect(a, b)
    if r is not None:
        roots.append(r)

# ----- 9. comparaison avec les vrais zeros -----
TRUE_ZEROS = [14.134725141734693, 21.022039638771555,
               25.010857580145688, 30.424876125859513,
               32.935061587739189, 37.586178158825671,
               40.918719012147495, 43.327073280914999,
               48.005150881167159, 49.773832477672302]

print("lambda^2 =", LAMBDA_SQ, " N =", N, " dimension =", DIM)
print("plus petite valeur propre eps =", "%.3e" % vals[i0])
print()
print("racines reelles trouvees (toutes, y compris les fantomes) :")
print(" ", [round(r, 6) for r in roots])
print()
print(" k |          calcule |          Riemann |          erreur")
for k, tz in enumerate(TRUE_ZEROS, 1):
    near = [r for r in roots if abs(r - tz) < 0.8]
    if near:
        r = min(near, key=lambda r: abs(r - tz))

```

```
        print ("%2d | %12.9f | %13.9f | %.2e" % (k, r, tz, abs(r - tz)))
    else:
        print ("%2d |          ABSENT | %13.9f | " % (k, tz))
```