Analyse de conformité du programme `python` proposé par l'ia chatgpt aux spécifications théoriques des articles de Connes et al.
Denise Vella-Chemla pilotant l'ia mistral, 22 juillet 2026

1. Introduction

Ce document présente une analyse détaillée du programme `python` proposé par l'ia chapgpt au regard des spécifications théoriques établies dans les articles de référence :

- [1] *Quadratic Forms, Real Zeros and Echoes of the Spectral Action* (Connes et Van Suijlekom, 2025) https://arxiv.org/pdf/2511.23257
- [2] *Zeta Spectral Triples* (Connes, Consani, Moscovici, 2024) https://arxiv.org/pdf/2511.22755

L'objectif est d'identifier les écarts entre l'implémentation numérique et les prescriptions théoriques, en soulignant les choix techniques qui s'écartent des spécifications, que ce soit par nécessité pratique, par limite de calcul, ou par interprétation différente.

2. Structure du programme et correspondances théoriques

2.1. Architecture globale

Le programme suit la chaîne de traitement décrite dans les articles :

1. calcul des puissances premières $\leq \lambda^2$
2. construction de la matrice de Weil $\tau_{n,m}$
3. extraction du vecteur propre minimal $\xi_{\lambda,N}$
4. construction de la fonction rationnelle $R(z)$
5. calcul des zéros de $R(z)$

Cette structure est **conforme** à la méthodologie décrite dans *Zeta Spectral Triples* ([2] § 5).

2.2. Implémentation des composantes

2.2..1 1. Génération des puissances premières (fonction `prime_powers_up_to`)

- **théorie** : la somme sur les places non-archimédiennes porte sur tous les entiers $k \in [2, \exp(L)]$ avec la fonction de von Mangoldt $\Lambda(k)$ (cf. *Zeta Spectral Triples* ([2] § 3.3)).
- **code** : génère uniquement les puissances premières $p^k \leq$ une certaine borne.
- **analyse** : **conforme**. En effet, $\Lambda(k) = \log p$ si $k = p^m$, et $\Lambda(k) = 0$ sinon. La somme sur les puissances premières est donc équivalente à la somme sur tous les entiers.
- **écart technique** : la borne utilisée est $\text{int}(\lambda^2)$ au lieu de $\lfloor \exp(L) \rfloor$. Or $L = 2 \log \lambda$, donc $\exp(L) = \lambda^2$. **Conforme** si λ^2 est entier, sinon **écart mineur** (exclusion des entiers dans $[\text{int}(\lambda^2), \lambda^2]$).

2.2..2 2. Fonction $\rho(x)$

- **théorie** : $\rho(x) = \dfrac{\exp\left(\dfrac{x}{2}\right)}{\exp(x) - \exp(-x)} = \dfrac{\exp\left(\dfrac{x}{2}\right)}{2\sinh(x)}$ (*Zeta Spectral Triples* ([2] § 4.5-§ 4.7)).
- **code** :

```
def rho(x):
    x = np.asarray(x)
    result = (np.exp(x / 2)/(2 * np.sinh(x)))
    small = np.abs(x) < 1e-12
    if np.any(small):
        result = np.where(small, 1 / (2 * x) + 1 / 4, result)
    return result
```

- **analyse** : **conforme**.

  Le traitement pour x proche de 0 utilise le développement $\rho(x) \approx \dfrac{1}{2x} + \dfrac{1}{4} + \mathrm{O}(x^2)$, qui est correct.

2.2..3 3. Intégrales archimédiennes (`archimedean_integrals`)

- **théorie** : les intégrales sont définies analytiquement via des fonctions spéciales (hypergéométriques ${}_2F_1$, digamma ψ, Hurwitz-Lerch Φ) (*Zeta Spectral Triples* ([2] Proposition 4.2).
- **code** : utilise une intégration numérique avec `fixed_quad` (quadrature de Gauss-Lobatto).
- **analyse** :
  - **écart majeur** : **précision insuffisante**. Les articles utilisent explicitement une précision de **200 chiffres** pour garantir la stabilité numérique, notamment pour les petites valeurs propres $\epsilon_N \sim 10^{-50}$ (*Zeta Spectral Triples* ([2] Tableau § 6)). Le code utilise la précision double standard de `python` ($\sim$ 15 à 17 chiffres), ce qui est **insuffisant** pour reproduire les résultats numériques de l'article.
  - **écart sur la constante** : la contribution archimédienne diagonale utilise :

    ```
    kappa_half = 0.5*log(4*pi*(exp(L)-1)/(exp(L)+1))+0.5*gamma_E
    ```

    qui correspond à $w(L)$ (*Zeta Spectral Triples* ([2] § 4.11)). Cependant, la formule théorique pour $\gamma_L(n)$ inclut également $c(L)$ (*Zeta Spectral Triples* ([2] § 4.11)) :

    $$c(L) = \log(e^{L/2} + 1) + \frac{1}{4}\left(-2\log(e^L + 1) - \pi - \log(4)\right) + \tan^{-1}(e^{L/2})$$

    → **il manque $c(L)$ dans les termes diagonaux de la matrice.**

2.2..4 4. Construction de la matrice $\tau_{n,m}$ (`build_tau`)

- **théorie** : la matrice $\tau_{n,m} = QW_\lambda(V_n, V_m)$ a trois contributions (*Zeta Spectral Triples* ([2] § 4)) :

- pôle :

$$W_{0,2}(V_n, V_m) \tag{4.2}$$

- puissances premières :

$$\sum_{1<k\leq \exp(L)} \Lambda(k) k^{-1/2} q(U_n, U_m)(\log k) \tag{4.3}$$

- archimédienne :

$$W_{\mathbb{R}}(V_n, V_m) \tag{4.4}$$

- **code** : implémente les trois contributions.
- **analyse** :
  - **contribution du pôle** : **conforme**. La formule implémentée :

```
numerator = (L2-16*math.pi2mn)
denominator = ((16math.pi**2m2 + L2)(16math.pi2*n2 + L**2))
pole = (32Lmath.sinh(L / 4)**2*numerator/denominator)
```

  correspond exactement à (*Zeta Spectral Triples* ([2] § 4.2)) :

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

  - **contribution des puissances premières** : **conforme**. Le code calcule :

```
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))
```

  Or $\Lambda(k) = \log p$ pour $k = p^m$, donc $\dfrac{\Lambda(k)}{\sqrt{k}} = \dfrac{\log p}{\sqrt{k}}$. La formule est donc correcte.

  - **contribution archimédienne** : **partiellement conforme**. Les termes non-diagonaux ($m \neq n$) utilisent :

```
arch = (signed_alpha(m) - signed_alpha(n))/(n - m)
```

  qui correspond à (*Zeta Spectral Triples* ([2] Proposition 4.3) :

$$W_{\mathbb{R}}(V_n, V_m) = \frac{\alpha_L(m) - \alpha_L(n)}{n - m} \quad (m \neq n)$$

  → **conforme pour** $m \neq n$. Pour $m = n$, le terme diagonal utilise :

```
arch = 2 * (gamma[abs(n)] - beta[abs(n)])
```

  qui correspond à $2(\gamma_L(n) - \beta_L(n))$, mais **manque** $c(L)$ (cf. § 3).

2.2..5 5. Vecteur propre minimal (`smallest_weil_state`)

- **théorie** : le vecteur propre ξ est normalisé par $\delta_N(\xi) = 1$, où δ_N est le noyau de Dirichlet (*Zeta Spectral Triples* ([2] § 5.3).
- **code** : normalise par $\|\xi\|_2 = 1$:

```
xi /= np.linalg.norm(xi)
```

- **analyse** : **écart de normalisation**. Cependant, comme les zéros de $\widehat{\xi}(z)$ sont invariants par multiplication de ξ par une constante non nulle, cet écart **n'affecte pas la position des zéros calculés**. Il affecte uniquement l'échelle de la fonction $\widehat{\xi}(z)$, mais pas ses racines.

2.2..6 6. Calcul des zéros (`compute_zeros`)

- **théorie** : les zéros de $\widehat{\xi}(z)$ sont donnés par (*Zeta Spectral Triples* Théorème 5.10) :

$$\widehat{\xi}(z) = 2L^{-1/2} \sin\left(\frac{zL}{2}\right) \left(\sum_{j=-N}^{N} \frac{\xi_j}{z - \dfrac{2\pi j}{L}} \right)$$

  Les zéros sont :
  - les zéros de $\sin\left(\dfrac{zL}{2}\right)$: $z = \dfrac{2\pi n}{L}$ pour $n \in \mathbb{Z}$.
  - les zéros du polynôme $P(z) = \displaystyle\sum_{j=-N}^{N} \xi_j \prod_{k \neq j} \left(z - \frac{2\pi k}{L}\right)$.
- **code** : calcule uniquement les zéros de $P(z)$:

```
def numerator_polynomial(xi, L):
    poles = np.array([2*math.pi/L for n in range(-N, N + 1)], dtype=float)
    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)
    return numerator
```

- **analyse** : **écart partiel**. Le code ne calcule que les zéros de $P(z)$, mais ignore les zéros de $\sin\left(\dfrac{zL}{2}\right)$. Cependant :
  - les zéros de $\sin\left(\dfrac{zL}{2}\right)$ correspondent aux pôles de la fonction rationnelle

$$R(z) = \sum \frac{\xi_j}{z - \dfrac{2\pi j}{L}}.$$

  - ces pôles sont annulés par les zéros de $P(z)$ lorsque $\xi_j \neq 0$ (*Zeta Spectral Triples* [2 théorème 5.10 (iii)).

— les zéros non triviaux (ceux qui approchent les zéros de Riemann) sont exactement les zéros de $P(z)$.

→ Pour la comparaison avec les zéros de Riemann, calculer uniquement les zéros de $P(z)$ est suffisant.

3. Écarts majeurs et impacts

Composante	**Écart**	**Impact**
intégrales archimédiennes	Précision numérique insuffisante (double vs 200 chiffres)	Erreurs significatives sur les petites valeurs propres ϵ_N (ex. $\epsilon_N \sim 10^{-50}$ dans l'article vs erreurs numériques dans le code)
Constante $c(L)$ manquante dans les termes diagonaux	Décalage systématique des valeurs propres, affectant la précision des zéros calculés	
Taille de N	$N = 10$ dans l'exemple vs $N = 120$ dans l'article	Moins de modes → moins de précision pour les zéros supérieurs
Normalisation du vecteur propre	$\|\xi\|_2 = 1$ vs $\delta_N(\xi) = 1$	Aucun impact sur les zéros (invariance par scaling)
Calcul des zéros	Seuls les zéros de $P(z)$ sont calculés	Aucun impact pour la comparaison avec Riemann (les zéros manquants sont connus et annulés)
Symétrisation de la matrice	$(\tau + \tau^T)/2$ appliqué	Pratique numérique saine, mais théoriquement inutile (la matrice devrait être symétrique)

4. Recommandations pour une implémentation conforme

4.1. Corrections immédiates

1. **ajouter** $c(L)$: modifier `archimedean_integrals` pour inclure $c(L)$ dans `kappa_half` :

```
def c_L(L):
return (math.log(math.exp(L/2) + 1) +
0.25 * (-2 * math.log(math.exp(L) + 1) - math.pi - math.log(4)) +
math.atan(math.exp(L/2)))
kappa_half = 0.5*log(4*pi*(exp(L)-1)/(exp(L)+1))+0.5*gamma_E+c_L(L)
```

2. **Augmenter** N : utiliser $N \geq 120$ pour reproduire les résultats de l'article (ex. pour $\lambda = \sqrt{13}$).
3. **Corriger la borne des puissances premières** : remplacer $\text{int}(\lambda^2)$ par $\lfloor \exp(L) \rfloor$:

```
bound = math.floor(math.exp(L))
```

4.2. Améliorations structurelles

1. **précision arbitraire** : utiliser une bibliothèque comme `mpmath` pour les intégrales et les calculs matriciels, avec une précision de 200 chiffres :

import mpmath mpmath.mp.dps = 200 200 chiffres de precision

2. **formules analytiques** : remplacer l'intégration numérique des intégrales archimédiennes par les formules analytiques de l'article (Proposition 4.2), utilisant les fonctions spéciales de `mpmath` :

from mpmath import hyper, psi, phi ${}_2F_1$, digamma, Hurwitz-Lerch

3. **normalisation conforme** : implémenter la normalisation $\delta_N(\xi) = 1$:

$delta_N = np.ones(2 * N + 1)/np.sqrt(L)$ Noyau de Dirichlet $xi/ = np.dot(delta_N, xi)$ Normalisation par delta$_N(xi) = 1$

4.3. Validation

- comparer les valeurs propres minimales ϵ_N avec celles de l'article (*Zeta Spectral Triples* [2] Tableau § 6).
  Pour $\lambda = \sqrt{13}$ et $N = 120$, on devrait obtenir $\epsilon_N \sim 10^{-55}$ pour le premier zéro.
- vérifier que les zéros calculés pour $P(z)$ correspondent aux zéros de Riemann avec une erreur $< 10^{-50}$ pour les premiers zéros (cf. *Zeta Spectral Triples* ([2] § 6)).

5. Conclusion

Le programme en `python` de l'ia chapgpt implémente globalement la méthodologie théorique des articles de Connes et al., mais présente des **écarts critiques** qui limitent sa précision :

- **précision numérique** : l'utilisation de la double précision au lieu de 200 chiffres est le principal obstacle à la reproduction des résultats de l'article.
- **contribution archimédienne** : l'omission de $c(L)$ introduit un biais systématique.
- **paramètres** : la valeur de N par défaut ($N = 10$) est trop faible pour capturer les zéros supérieurs.

Ces écarts sont **techniquement surmontables** :

- la précision arbitraire est disponible via `mpmath`.
- les formules analytiques pour les intégrales archimédiennes sont fournies dans l'article.
- les ressources computationnelles pour $N = 120$ sont accessibles.

Une implémentation corrigée, combinant ces améliorations, devrait reproduire les résultats numériques spectaculaires de l'article (ex. erreur $< 10^{-50}$ pour les premiers zéros de Riemann avec $\lambda = \sqrt{13}$ et $N = 120$).