K-Theory of the Sequence Algebra, Spectral Dimension Three, and Path-Calibrated Spectral Metrics

Finite Certificates and Analytic Scope

Verification Code Companion

Charles D. Schaper, Ph.D.
Annals of the Chrysene Formalism, Volume 1, Number 4
Article 95, November 2026

1 Scope and reproducible result

The replacement program executes **12,472 explicit checks per mode**: 12,471 exact checks and one floating-point comparison. Ten negative controls are included in this total. Normal and optimized Python produce byte-identical outputs; no check relies on a Python `assert` statement. Only the Python standard library is required. The count measures calls to the validation function, not independent theorems.

The checks concern finite algebraic identities, integer lattice counts and graph witnesses. They do not prove the Bernoulli crossed-product K-theory theorem, infinite Weyl asymptotics, infinite graph duality, or molecular realization. The manuscript's analytic results have separate mathematical proofs and cited dependencies.

2 Corrections to inherited verification

The supplied source uses infinity on the diagonal of rejected states, yet spectral distance from a state to itself is zero. Retaining every structural state as a vertex, including the nine isolated rejected states, changes the ordered pair counts from 12/213 to 21 **finite and** 204 **infinite**. Adding one scalar summand per isolated vertex makes the edge representation faithful: its dimension is $2 \cdot 3 + 9 = 15$, and the graded lattice product fibre is $8 \cdot 15 = 120$.

The original rank recursion was arithmetic rather than an independent proof of iterated Pimsner–Voiculescu splitting. It also did not implement all of the matrix and filtration checks attributed to it. The replacement directly tests the claimed finite matrix blocks and withdraws the old filtration-distance claim. The original source is retained only as historical evidence.

3 Coverage and interpretation

R1: ranks and extension caution.
The split rank recursion through dimension ten agrees with exterior-binomial counts. A nonidentity shear fixes a subobject and quotient, showing why endpoint naturality alone does not prove identity on the middle group. The revised manuscript obtains the analytic groups from the cited Bernoulli-shift theorem; these checks neither establish nor weaken that theorem.

R2: Clifford squares and spectra.
Exact integer matrices verify all Clifford anticommutators and 375 graded squares. Two independent enumeration methods agree for small lattice balls and for six shifted internal

spectra. The radius-30 zero-mass ball has 113081 sites, hence $N(30) = 3618592$ at fibre dimension 32, and $N(30)/[(128\pi/3)30^3] \approx 0.9998555622$. This example assumes all internal masses are zero. Its decimal comparison is the sole floating-point check. Finite counts do not prove an asymptotic exponent.

R3: edge commutators.
For 196 rational edge data sets, the actual matrices $D = w^{-1}\sigma_x$ and $\pi(f) = \mathrm{diag}(u, v)$ are multiplied. Both the commutator and $C^T C = ((v-u)/w)^2 I$ are checked exactly, giving the stated singular values.

R4: source graph.
Every source vertex has zero self-distance. Distance potentials attain every finite pair distance; component indicators have zero edge seminorm and separate all disconnected pairs. The scalar isolated-vertex blocks restore faithfulness without changing the seminorm.

R5: weighted graph and cube witnesses.
The same potential and component checks run on all 64 simple four-vertex graph supports with rational weights, and on the $3 \times 3 \times 3$ cube. Its 729 distances agree with coordinate Manhattan distance. These witness checks complement the manuscript's triangle-inequality upper-bound proof, rather than replacing a supremum calculation by unsupported equality.

R6: cuts and bypasses.
Increasing the sole edge weight of a two-vertex graph increases its distance. Increasing one edge of a triangle leaves a two-unit bypass, so its endpoint distance is $\min(w, 2)$. Thus one diverging edge weight need not yield infinite structural distance.

4 Negative controls

The ten rejected claims give wrong ranks, identify a shear with identity, omit the lattice origin, put infinity on a state diagonal, retain the old pair counts, claim edge-only faithfulness or dimension, lower the cube diameter, or ignore a triangle bypass. These are deliberately false propositions tested against independently obtained data.

5 Reproduction and provenance

Run `python run_verification.py`. The runner verifies the pinned source hash, executes normal and optimized Python in temporary directories, compares every output byte, and writes `execution_audit.json`. The macro file is generated from the tested values. The source below is also embedded in the PDF and recreated by this standalone LaTeX file.

6 Complete executable source

Article 95 verification source

```
#!/usr/bin/env python3
"""Article 95 finite verification. Python standard library only.
No assertion statements: checks survive -O. No claim to prove C*-K-theory,
infinite Weyl asymptotics, countable-graph duality, or molecular realization.
"""
from fractions import Fraction as F
from itertools import product, combinations
```

```python
from collections import Counter
from math import comb, isqrt, pi
import json
checks=Counter(); numerical=0

def check(label, ok):
    checks[label]+=1
    if not ok: raise RuntimeError(label)

def matmul(a,b):
    return [[sum(x*y for x,y in zip(row,col)) for col in zip(*b)] for row in a]
def add(a,b): return [[x+y for x,y in zip(r,s)] for r,s in zip(a,b)]
def scale(k,a): return [[k*x for x in r] for r in a]
def trans(a): return list(map(list,zip(*a)))
def eye(n): return [[int(i==j) for j in range(n)] for i in range(n)]
def kron(a,b):
    return [[x*y for x in r for y in s] for r in a for s in b]
def comm(a,b): return add(matmul(a,b),scale(-1,matmul(b,a)))

# R1: conditional split-rank arithmetic, independently via exterior subsets.
ranks=[(1,0)]
for r in range(1,11):
    a,b=ranks[-1]; ranks.append((a+b,a+b))
    even=sum(comb(r,j) for j in range(0,r+1,2))
    odd=sum(comb(r,j) for j in range(1,r+1,2))
    check('R1 ranks',ranks[-1]==(even,odd))
    for mask in range(1<<r):
        check('R1 complement', (mask.bit_count()+(mask^((1<<r)-1)).bit_count())==r)
# An exact sequence end-action does not force the middle action to be identity.
shear=[[1,1],[0,1]]
check('R1 shear fixes subobject',matmul(shear,[[1],[0]])==[[1],[0]])
check('R1 shear fixes quotient',matmul([[0,1]],shear)==[[0,1]])
check('R1 shear not identity',shear!=eye(2))

# R2: exact Clifford square and finite lattice eigenvalue counts.
X=[[0,1],[1,0]]; Z=[[1,0],[0,-1]]; I=eye(2)
gamma=[kron(X,I),kron(Z,X),kron(Z,Z)]
for i in range(3):
    for j in range(3):
        check('R2 Clifford',add(matmul(gamma[i],gamma[j]),matmul(gamma[j],gamma[i]))==
            scale(2*int(i==j),eye(4)))
for n in product(range(-2,3),repeat=3):
    dl=[[sum(n[k]*gamma[k][i][j] for k in range(3)) for j in range(4)] for i in range
        (4)]
    for mu in [-2,0,1]:
        total=add(kron(scale(mu,eye(4)),X),kron(dl,Z))
        check('R2 graded square',matmul(total,total)==scale(mu*mu+sum(t*t for t in n),
            eye(8)))

def ball(R):
    return sum(2*isqrt(R*R-x*x-y*y)+1
        for x in range(-R,R+1)
        for y in range(-isqrt(R*R-x*x),isqrt(R*R-x*x)+1))
counts={R:ball(R) for R in range(31)}
for R in range(7):
    brute=sum(x*x+y*y+z*z<=R*R for x,y,z in product(range(-R,R+1),repeat=3))
    check('R2 independent count',counts[R]==brute)
for R in range(1,31):
    check('R2 nested balls',counts[R]>=counts[R-1])
    # Exact rational enclosure from covering unit cubes; sqrt(3)/2 < 1.
    check('R2 cube volume bounds',F(4)*F(223,71)*max(R-1,0)**3/3<=counts[R]<=F(4)*F
        (22,7)*(R+1)**3/3)
# Shifted internal energies: direct counting, not assumed zero-mass formula.
shifted={}
for L in range(1,7):
    masses=[0,1,2,3]
    exact=8*sum(sum(mu*mu+x*x+y*y+z*z<=L*L
        for x,y,z in product(range(-L,L+1),repeat=3)) for mu in masses)
```

```
    fast=8*sum(sum(2*isqrt(L*L-mu*mu-x*x-y*y)+1
        for x in range(-L,L+1) for y in range(-L,L+1)
        if L*L-mu*mu-x*x-y*y>=0) for mu in masses)
    check('R2 shifted spectrum',exact==fast); shifted[L]=exact
ratio=counts[30]/(4*pi*30**3/3)
check('R2 numerical ratio',abs(ratio-1)<.002); numerical+=1

# R3: form actual rational commutators and C^T C on edge blocks.
for u,v,w in product(range(-3,4),range(-3,4),[F(1,2),F(1),F(3,2),F(2)]):
    D=scale(1/w,X); P=[[F(u),0],[0,F(v)]]; C=comm(D,P)
    check('R3 edge block',C==[[0,F(v-u)/w],[F(u-v)/w,0]])
    check('R3 squared singular values',matmul(trans(C),C)==scale(F(v-u)**2/w**2,I))

# R4/R5 finite graph metric proof witnesses, including isolated vertices.
def distances(n,edges):
    d=[[F(0) if i==j else None for j in range(n)] for i in range(n)]
    for u,v,w in edges:d[u][v]=d[v][u]=w
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if d[i][k] is not None and d[k][j] is not None:
                    z=d[i][k]+d[k][j]
                    if d[i][j] is None or z<d[i][j]:d[i][j]=z
    return d

def audit_graph(n,edges,tag):
    d=distances(n,edges)
    for x in range(n):
        check(tag+' diagonal',d[x][x]==0)
        potential=[z if z is not None else F(0) for z in d[x]]
        for u,v,w in edges:check(tag+' Lipschitz witness',abs(potential[u]-potential[v
            ])<=w)
        component=[int(z is not None) for z in d[x]]
        for u,v,w in edges:check(tag+' zero seminorm',component[u]==component[v])
        for y in range(n):
            if d[x][y] is None:
                check(tag+' disconnected separation',component[x]-component[y]==1)
            else:
                check(tag+' attained distance',abs(potential[x]-potential[y])==d[x][y])
            check(tag+' symmetry',d[x][y]==d[y][x])
    return d
source_edges=[(3*n+1,3*n+2,F(1)) for n in [2,3,4]]
src=audit_graph(15,source_edges,'R4 source')
finite=sum(x is not None for row in src for x in row)
check('R4 pair totals',(finite,225-finite)==(21,204))
covered={v for u,v,w in source_edges}|{u for u,v,w in source_edges}
isolated=set(range(15))-covered
check('R4 faithful coverage',len(covered|isolated)==15 and len(isolated)==9)
check('R4 faithful dimension',2*len(source_edges)+len(isolated)==15)
# Every projection at a vertex acts nontrivially in edge/singleton representation.
for vertex in range(15):check('R4 faithful vertex',vertex in covered or vertex in
    isolated)
possible=list(combinations(range(4),2))
for mask in range(64):
    edges=[(u,v,F(1+(k%3),2)) for k,(u,v) in enumerate(possible) if (mask>>k)&1]
    audit_graph(4,edges,'R5 weighted graph')
vertices=list(product(range(3),repeat=3))
box_edges=[(i,j,F(1)) for i,u in enumerate(vertices) for j,v in enumerate(vertices) if
    i<j and sum(abs(a-b) for a,b in zip(u,v))==1]
box=audit_graph(27,box_edges,'R5 cube')
for i,u in enumerate(vertices):
    for j,v in enumerate(vertices):check('R5 Manhattan',box[i][j]==sum(abs(a-b) for a,b
        in zip(u,v)))
# R6: removal is a cut condition, not merely a large individual edge weight.
for W in [1,2,3,10,100]:
    bridge=distances(2,[(0,1,F(W))])
    cycle=distances(3,[(0,1,F(W)),(0,2,F(1)),(2,1,F(1))])
    check('R6 cut distance',bridge[0][1]==W)
```

```
    check('R6 bypass distance',cycle[0][1]==min(W,2))
# Negative controls are deliberately wrong claims, explicitly rejected.
wrong=[ranks[3]==(3,3),shear==eye(2),counts[1]==6,
       src[0][0] is None,finite==12,225-finite==213,
       len(covered)==15,2*len(source_edges)==15,
       box[0][26]==5,distances(3,[(0,1,F(100)),(0,2,F(1)),(2,1,F(1))])[0][1]==100]
for k,claim in enumerate(wrong):check('negative controls',not claim)
macros={'valLocalDim':4,'valTracelessBottRank':3,'valSpinorFibreDim':32,
 'valSpectralDim':3,'valWeylCoeffExact':r'128\pi/3','valWeylCoeffNum':f'{128*pi/3:.4f}',
 'valWeylRatioThirty':f'{ratio:.4f}','valUnifFibreDim':120,'valSourceStates':15,
 'valSourceAdmStates':6,'valSourceRejStates':9,'valSourcePairsTotal':225,
 'valSourceFinitePairs':21,'valSourceInfPairs':204,'valBoxThreePairs':729,'valMaxMetricError':0}
for r,name in enumerate(['Zero','One','Two','Three']):
    macros['valKZeroRank'+name]=ranks[r][0];macros['valKOneRank'+name]=ranks[r][1]
report={'checks':sum(checks.values()),'exact_checks':sum(checks.values())-numerical,
 'numerical_checks':numerical,'negative_controls':len(wrong),'blocks':dict(checks),
 'source_finite_pairs':finite,'source_infinite_pairs':225-finite,'calibrated_dimension':15,
 'lattice_ball_30':counts[30],'zero_mass_count_30':32*counts[30],
 'weyl_ratio_30_zero_mass':ratio,'shifted_counts':shifted,
 'scope':'Finite exact algebra and graph certificates; rank arithmetic checks the CEKN-derived formula, not the imported analytic theorem. No infinite C*-theory or chemical realization proof.'}
open('verification_values_a95.tex','w').write('% Generated finite checks; analytic K-theory depends on CEKN.\n'+''.join('\\newcommand{\\%s}{%s}\n'%(k,v) for k,v in macros.items()))
open('verification_report.json','w').write(json.dumps(report,indent=2,sort_keys=True)+'\n')
print(json.dumps(report,sort_keys=True))
```