

Challenge 02: Gravitational Positivity Causality Bounds

Pure Thought AI Challenge 02

Pure Thought AI Challenges Project

January 18, 2026

Abstract

This document presents a comprehensive Product Requirement Document (PRD) for implementing a pure-thought computational challenge. The problem can be tackled using only symbolic mathematics, exact arithmetic, and fresh code—no experimental data or materials databases required until final verification. All results must be accompanied by machine-checkable certificates.

Contents

Domain: Quantum Gravity Particle Physics

Difficulty: High

Timeline: 3-9 months

Prerequisites: Scattering amplitudes, dispersion relations, convex optimization

0.1 Problem Statement

0.1.1 Scientific Context

Effective field theories (EFTs) with gravity cannot be arbitrary. Quantum consistency, unitarity, causality, and analyticity impose stringent constraints on the coefficients of higher-derivative terms like R^2 , R^3 in the gravitational action. These constraints distinguish theories that can be UV-completed into consistent quantum gravity from those in the "swampland."

0.1.2 The Core Question

What are the sharp bounds on Wilson coefficients of higher-derivative gravitational operators imposed purely by consistency conditions?

For example, for corrections to Einstein gravity:

1

$$S = \int d^4x \sqrt{-g} \left[\frac{M_{\text{Pl}}^2}{2} R + a R^2 + a_1 R_{\mu\nu} R^{\mu\nu} + a_2 R^{\mu\nu\rho\sigma} R_{\mu\nu\rho\sigma} + \dots \right]$$

What ranges of a , a_1 , a_2 , ... are consistent with:

- Unitarity (positive-norm states)
- Causality (no superluminal propagation)
- Analyticity (dispersion relations)
- Crossing symmetry

0.1.3 Why This Matters

- **Swampland program:** Determines which low-energy theories can couple to gravity
- **Phenomenology:** Constrains quantum gravity corrections to GR
- **Rigorous:** Produces mathematical

no-go theorems, not heuristics

0.2 Mathematical Formulation

0.2.1 Problem Definition

Consider graviton-graviton scattering amplitude $M(s,t)$ where s,t are Mandelstam variables.

Constraints from physics:

- **Unitarity:** $\text{Im } M(s,t) \geq 0$ in physical region
- **Analyticity:** $M(s,t)$ is analytic except on physical cuts
- **Crossing:** $M(s,t) = M(t,s) = M(u,t)$ where $u = -s-t$
- **Causality (shockwave):** Time delay $t \geq 0$ for shockwave scattering
- Translates to: certain combinations of Wilson coefficients must be positive
- **Dispersion relation:** For fixed $t < 0$,

$$M(s,t) = M(0,t) + s \int_0^\infty \frac{\text{Im } M(s',t)}{(s' - s)^2} ds'$$

Optimization formulation:

Given a set of Wilson coefficients a, a, \dots, a , determine feasibility:

```

1 Minimize/Maximize:  a
2 Subject to:
3   - Dispersion relation holds
4   - Im M(s,t) ≥ 0 (unitarity)
5   - Crossing symmetry satisfied
6   - Causality bounds satisfied
7   - Regge bound: M(s,t) ~ s^α at large s

```

This is a **Sum-of-Squares (SoS) or Semidefinite Program (SDP)** that can be solved with certificates.

0.2.2 Certificate of Correctness

Allowed region certificate:

- Explicit Wilson coefficients a, a, \dots
- Explicit scattering amplitude $M(s,t)$ satisfying all constraints
- Verification: compute amplitude, check unitarity numerically

Forbidden region certificate:

- Dual SoS certificate: polynomial $p(s,t)$ such that:
- $p(s,t)$ is positive on physical region
- $p(s,t)$ [violated constraint] $ds dt < 0$
- This proves the region is impossible

0.3 Implementation Approach

0.3.1 Phase 1: $2 \rightarrow 2$ Graviton Scattering at Tree Level (Month 1-2)

Build amplitude calculator:

- Einstein gravity amplitude

```

1  def einstein_amplitude(s, t, M_pl):
2      """
3      Pure GR amplitude for graviton-graviton
4      graviton-graviton
5      """
6      u = -s - t
7      return (s*t*u) / M_pl**2 # Schematic

```

- Higher-derivative corrections

```

1  def corrected_amplitude(s, t, M_pl, wilson_coeffs):
2      """
3      Amplitude including R , R , ... corrections
4      """
5      M_0 = einstein_amplitude(s, t, M_pl)
6      M_R2 = wilson_coeffs['a1'] * (s**2 + t**2 + u**2)
7      M_R3 = wilson_coeffs['a2'] * (s**3 + t**3 + u**3)
8      return M_0 + M_R2 / M_pl**4 + M_R3 / M_pl**6 + ...

```

- Verify crossing symmetry

```

1  assert abs(M(s,t) - M(t,s)) < 1e-10
2  assert abs(M(s,t) - M(u,t)) < 1e-10

```

0.3.2 Phase 2: Dispersion Relations (Month 2-3)

Implement forward dispersion relation:

```

1  def dispersion_relation(M, s, t, s_min, s_max):
2      """
3      Check if M(s,t) satisfies dispersion relation
4
5      M(s,t) = poly(s,t) + s^N / (2*pi*i) * \int_{s_min}^{s_max} ds' Im M(s',t) / (s'-s)
6      """
7      # Subtracted dispersion relation
8      lhs = M(s, t)
9
10     # Polynomial subtraction
11     poly_part = sum(c_n * s**n for n, c_n in subtractions)
12
13     # Dispersive integral
14     def integrand(s_prime):
15         return imag(M(s_prime + 1j*epsilon, t)) / (s_prime**N * (s_prime - s))
16
17     dispersive_part = s**N / np.pi * quad(integrand, s_min, s_max)[0]
18
19     rhs = poly_part + dispersive_part

```

```

20
21     return abs(lhs - rhs) # Should be ~ 0

```

0.3.3 Phase 3: Causality from Shockwave Scattering (Month 3-4)

Eikonal phase shift:

The shockwave time delay is related to the eikonal phase:

$$1 \quad (b) = (1/2s) \quad d q / (2 \quad) \quad e^{\{i q \cdot b\}} M(s, t = -q^2)$$

Causality: $(b) \geq 0$ for all impact parameters b

This translates to positivity constraints on $M(s, t)$ at fixed t .

0.3.4 Phase 4: SDP Formulation Solver (Month 4-6)

Set up SDP:

```

1  import cvxpy as cp
2
3  # Variables: Wilson coefficients
4  a = cp.Variable(n_coeffs)
5
6  # Constraints
7  constraints = []
8
9  # 1. Unitarity: Im M >= 0
10 for s_i, t_i in grid_points:
11     Im_M = imaginary_part_amplitude(s_i, t_i, a)
12     constraints.append(Im_M >= 0)
13
14 # 2. Causality: shockwave positivity
15 for b_i in impact_parameters:
16     delta = eikonal_phase(b_i, a)
17     constraints.append(delta >= 0)
18
19 # 3. Dispersion relation (approximate as polynomial constraints)
20 # ... implement as sum-of-squares
21
22 # Objective: maximize/minimize specific coefficient
23 objective = cp.Maximize(a[0]) # e.g., bound on a
24
25 problem = cp.Problem(objective, constraints)
26 problem.solve(solver=cp.MOSEK, verbose=True)
27
28 # Extract dual certificate
29 dual_cert = constraints[i].dual_value

```

0.3.5 Phase 5: Exact Certificates (Month 6-9)

Generate SoS certificates:

Use Sum-of-Squares decomposition to prove bounds:

```

1  from sympy import symbols, expand
2  from sympy.polys.polytools import Poly
3

```

```

4 s, t = symbols('s t', real=True)
5
6 # Claim: a > a_min
7 # Proof: Show that (a - a_min) can be written as SoS
8
9 def find_sos_certificate(constraint_poly, variables):
10     """
11     Find polynomials {p_i} such that:
12     constraint_poly = p_i + (physical constraints)
13     """
14     # Use SDP to find {p_i}
15     # Return explicit polynomials
16     pass

```

0.4 Example Starting Prompt

```

1 I need you to derive and implement bounds on gravitational Wilson
  coefficients
2 from causality and unitarity.
3
4 GOAL: Bound the coefficient a of the R term in the gravitational
  action.
5
6 PHASE 1 - Set up the amplitude:
7 1. Write down the tree-level graviton-graviton scattering amplitude
8    in Einstein gravity (just GR).
9
10 2. Add the R correction term and write the corrected amplitude M(s,t;
11    a ).
12
13 3. Verify crossing symmetry: M(s,t) = M(t,s) = M(u,t).
14
15 PHASE 2 - Implement dispersion relation:
16 4. Write the forward dispersion relation (t=0) for graviton scattering.
17
18 5. Express the dispersive integral in terms of Im M(s, 0).
19
20 6. Use unitarity: Im M(s,0) >= 0 for s > threshold.
21
22 PHASE 3 - Causality constraint:
23 7. Implement the eikonal phase shift (b) from the amplitude.
24
25 8. Impose (b) <= 0 for all impact parameters b.
26
27 9. Translate this into a constraint on a .
28
29 PHASE 4 - Solve the optimization:
30 10. Formulate as SDP: maximize a subject to all constraints.
31
32 11. Also minimize a to get the allowed range.
33
34 12. Extract dual certificate from the SDP solver.

```

```

35 PHASE 5 - Verify:
36 13. Check that the certificate is a valid SoS decomposition.
37
38 14. Export certificate in machine-readable format.
39
40 Please implement this step-by-step, verifying each constraint
41     independently
    before combining them.

```

0.5 Success Criteria

0.5.1 Minimum Viable Result (3 months)

Single coefficient bound:

- Rigorous bounds on a (R^2 coefficient)
- Both upper and lower bounds
- Dual certificate extracted and verified

Validation:

- Reproduce known bounds from literature (if any)
- Independent verification of certificate

0.5.2 Strong Result (6 months)

Multi-parameter bounds:

- Simultaneous bounds on a , a , a
- Allowed region in 3D parameter space
- Boundary of region certified with SoS

Novel bounds:

- Tighter than previous literature, or
- New multi-coupling constraints
- Machine-checkable certificates

0.5.3 Publication-Quality Result (9 months)

Complete EFT space:

- All Wilson coefficients up to dimension 8 operators
- Full allowed region characterized
- Phase diagram of allowed vs. forbidden regions

Formal proofs:

- Certificates formalized in Lean/Isabelle
- Automated theorem proving for no-go regions
- Published with proof repository

0.6 Verification Protocol**0.6.1 Automated Checks**

```

1 def verify_bound_certificate(coeff_bounds, dual_certificate):
2     """
3     Verify that the dual certificate proves the bound.
4     """
5     # 1. Check SoS decomposition
6     sos_polynomials =
7         extract_sos_from_certificate(dual_certificate)
8     reconstructed = sum(p**2 for p in sos_polynomials)
9     assert is_equivalent(reconstructed, constraint_polynomial)
10
11     # 2. Verify positivity on physical region
12     test_points = generate_physical_region_samples(n=1000)
13     for s, t in test_points:
14         assert eval_certificate(dual_certificate, s, t) >= -1e-10
15
16     # 3. Check bound is saturated correctly
17     critical_amplitude =
18         construct_amplitude_from_certificate(dual_certificate)
19     verify_unitarity(critical_amplitude)
20     verify_causality(critical_amplitude)
21     verify_crossing(critical_amplitude)
22
23     return "VERIFIED"

```

0.6.2 Exported Artifacts

- **Bound certificate:** `bounda1.sos`
- Sum-of-Squares decomposition
- Exact rational coefficients
- Verifiable by independent SoS checkers
- **Allowed region:** `allowedregion.json`

```

1 {
2     "coefficients": ["a1", "a2", "a3"],
3     "bounds": {
4         "a1": {"min": -0.5, "max": 0.5},
5         "a2": {"min": 0.0, "max": 1.0}
6     }
7 }

```

```
6     },  
7     "constraints": ["unitarity", "causality", "crossing"]  
8 }
```

- **Proof script:** `gravitational_bounds.lean`

0.7 Milestone Checklist

Tree-level amplitude calculator implemented

Crossing symmetry verified numerically

Forward dispersion relation implemented

Unitarity bounds imposed

Eikonal/shockwave causality constraints derived

SDP solver setup with certificate extraction

First single-coefficient bound obtained

Dual certificate verified independently

Multi-parameter bounds computed

Formal verification initiated

Publication draft ready

Next Steps: Start with implementing the tree-level amplitude and verify crossing symmetry before adding any constraints.