# Log-Time Quantum Gravity (LTQG)

Area 05: Differential Geometry

Riemann Tensors, Curvature Invariants, and Geometric Analysis

Mathematical Physics Research Framework

October 17, 2025

**Abstract**

This document presents the differential geometry foundations of the Log-Time Quantum Gravity (LTQG) framework, focusing on rigorous computation of curvature tensors and geometric invariants. We provide complete implementations of Riemann tensor computation, curvature invariant analysis, and Weyl transformation effects within the LTQG coordinate system. The mathematical framework enables direct computation from metric tensors without geometric shortcuts, ensuring accuracy and consistency across all applications.

## Contents

## 1 Introduction

The differential geometry module of LTQG provides comprehensive tools for analyzing curved spacetime within the log-time coordinate framework. This area addresses the fundamental geometric objects that describe gravitational physics: the Riemann curvature tensor, various curvature invariants, and their transformation properties under coordinate changes.

### 1.1 Geometric Framework

In LTQG, we work with spacetime metrics that undergo Weyl transformations:

$$\tilde{g}_{\mu\nu} = \Omega^2(\tau) g_{\mu\nu} \tag{1}$$

where $\Omega(\tau) = 1/\tau$ is the Weyl factor and $\tau$ is the original time coordinate. The log-time coordinate $\sigma = \log(\tau/\tau_0)$ provides a natural framework for analyzing the geometric properties.

### 1.2 Mathematical Objectives

This module implements:

- Complete Riemann tensor computation from metric coefficients
- Ricci tensor and scalar curvature analysis
- Curvature invariants: Kretschmann scalar, Ricci squared
- Einstein tensor and constraint identification
- Weyl transformation analysis with proper derivative handling

## 2 Riemann Tensor Computation

### 2.1 Mathematical Foundation

The Riemann curvature tensor is defined in terms of Christoffel symbols:

$$R^{\rho}{}_{\sigma\mu\nu} = \partial_\mu \Gamma^{\rho}{}_{\sigma\nu} - \partial_\nu \Gamma^{\rho}{}_{\sigma\mu} + \Gamma^{\rho}{}_{\lambda\mu}\Gamma^{\lambda}{}_{\sigma\nu} - \Gamma^{\rho}{}_{\lambda\nu}\Gamma^{\lambda}{}_{\sigma\mu} \tag{2}$$

**Definition 2.1** (Christoffel Symbols)**.** The Christoffel symbols are computed from the metric tensor:

$$\Gamma^{\rho}{}_{\mu\nu} = \frac{1}{2} g^{\rho\lambda} \left(\partial_\mu g_{\lambda\nu} + \partial_\nu g_{\lambda\mu} - \partial_\lambda g_{\mu\nu}\right) \tag{3}$$

### 2.2 Implementation Details

The LTQG framework implements symbolic computation of the Riemann tensor:

```
class SymbolicCurvature:
    @staticmethod
    def riemann_tensor(g: sp.Matrix, coords: tuple,
                      Gamma: Optional[List] = None) -> List[List[List[List[sp.
                          Expr]]]]:
        """
        Compute Riemann curvature tensor from metric tensor.

        Returns:
            4D list structure R[a][b][c][d] representing R^a_bcd
        """
        n = g.shape[0]
        if Gamma is None:
            Gamma = SymbolicCurvature.christoffel_symbols(g, coords)

        R = [[[[sp.simplify(0) for _ in range(n)] for _ in range(n)]
              for _ in range(n)] for _ in range(n)]

        for a in range(n):
            for b in range(n):
                for c in range(n):
                    for d in range(n):
                        # R^a_bcd = partial_c Gamma^a_bd - partial_d Gamma^a_bc
                        # + Gamma^a_ec Gamma^e_bd - Gamma^a_ed Gamma^e_bc
                        term1 = (sp.diff(Gamma[a][b][d], coords[c]) -
                                sp.diff(Gamma[a][b][c], coords[d]))

                        term2 = sp.Integer(0)
                        for e in range(n):
                            term2 += (Gamma[a][e][c]*Gamma[e][b][d] -
                                     Gamma[a][e][d]*Gamma[e][b][c])

                        R[a][b][c][d] = sp.simplify(term1 + term2)

        return R
```

### 2.3 Computational Validation

The implementation validates against known analytical results:

**Theorem 2.2** (Riemann Tensor Properties)**.** *The computed Riemann tensor satisfies the fundamental symmetries:*

$$R_{\mu\nu\rho\sigma} = -R_{\nu\mu\rho\sigma} \quad \textit{(antisymmetry in first pair)} \tag{4}$$

$$R_{\mu\nu\rho\sigma} = -R_{\mu\nu\sigma\rho} \quad \textit{(antisymmetry in second pair)} \tag{5}$$

$$R_{\mu\nu\rho\sigma} = R_{\rho\sigma\mu\nu} \quad \textit{(exchange symmetry)} \tag{6}$$

## 3 Curvature Invariants

### 3.1 Ricci Tensor and Scalar Curvature

The Ricci tensor is obtained by contracting the Riemann tensor:

$$\mathrm{Ric}_{\mu\nu} = R^{\rho}{}_{\mu\rho\nu} = g^{\rho\sigma} R_{\sigma\mu\rho\nu} \tag{7}$$

The scalar curvature is:

$$R = g^{\mu\nu}\mathrm{Ric}_{\mu\nu} \tag{8}$$

```
@staticmethod
def ricci_tensor(g: sp.Matrix, coords: tuple,
                Riemann: Optional[List] = None) -> sp.Matrix:
    """Compute Ricci tensor Ric_mu_nu = R^rho_mu_rho_nu."""
    n = g.shape[0]
    if Riemann is None:
        Riemann = SymbolicCurvature.riemann_tensor(g, coords)

    Ric = sp.MutableDenseMatrix.zeros(n, n)

    for mu in range(n):
        for nu in range(n):
            contraction = sp.Integer(0)
            for rho in range(n):
                contraction += Riemann[rho][mu][rho][nu]
            Ric[mu, nu] = sp.simplify(contraction)

    return Ric

@staticmethod
def scalar_curvature(g: sp.Matrix, coords: tuple,
                     Ric: Optional[sp.Matrix] = None) -> sp.Expr:
    """Compute scalar curvature R = g^bd R_bd."""
    if Ric is None:
        Ric = SymbolicCurvature.ricci_tensor(g, coords)

    g_inv = g.inv()
    n = g.shape[0]

    scalar = sp.Integer(0)
    for b in range(n):
        for d in range(n):
            scalar += g_inv[b, d] * Ric[b, d]

    return sp.simplify(scalar)
```

## 3.2 Kretschmann Scalar

The Kretschmann scalar measures tidal forces and is defined as:

$$K = R_{\mu\nu\rho\sigma}R^{\mu\nu\rho\sigma} \tag{9}$$

This invariant is particularly important for detecting true singularities in spacetime.

```
@staticmethod
def kretschmann_scalar(g: sp.Matrix, coords: tuple,
                       Riemann: Optional[List] = None) -> sp.Expr:
    """Compute Kretschmann scalar K = R_abcd R^abcd."""
    n = g.shape[0]
    if Riemann is None:
        Riemann = SymbolicCurvature.riemann_tensor(g, coords)

    g_inv = g.inv()

    # Convert to covariant components
    R_down = [[[[sp.Integer(0) for _ in range(n)] for _ in range(n)]
               for _ in range(n)] for _ in range(n)]

    for a in range(n):
        for b in range(n):
            for c in range(n):
                for d in range(n):
                    # R_abcd = g_ae R^e_bcd
                    for e in range(n):
                        R_down[a][b][c][d] += g[a, e] * Riemann[e][b][c][d]

    # Contract with inverse metric
    K = sp.Integer(0)
    for mu in range(n):
        for nu in range(n):
            for rho in range(n):
                for sigma in range(n):
                    R_up = sp.Integer(0)
                    # R^munurho*sigma = g^mu*alpha g^nu*beta g^rho*gamma g^
                        sigma*delta R_alpha*beta*gamma*delta
```

```
                            for alpha in range(n):
                                for beta in range(n):
                                    for gamma in range(n):
                                        for delta in range(n):
                                            R_up += (g_inv[mu, alpha] * g_inv[nu, beta]
                                                *
                                                   g_inv[rho, gamma] * g_inv[sigma,
                                                       delta] *
                                                   R_down[alpha][beta][gamma][delta])

                            K += R_down[mu][nu][rho][sigma] * R_up

    return sp.simplify(K)
```

## 4 Einstein Tensor and Field Equations

### 4.1 Einstein Tensor Construction

The Einstein tensor is fundamental to general relativity:

$$G_{\mu\nu} = \mathrm{Ric}_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R \tag{10}$$

This tensor satisfies the Bianchi identity:

$$\nabla^\mu G_{\mu\nu} = 0 \tag{11}$$

```
def einstein_tensor(g, coords):
    """Compute Einstein tensor G_mu_nu = R_mu_nu - (1/2)*g_mu_nu*R."""
    Riem = riemann_tensor(g, coords)
    Ric = ricci_tensor(g, coords, Riem)
    Rsc = scalar_curvature(g, coords, Ric)
    n = g.shape[0]
    G = sp.MutableDenseMatrix.zeros(n, n)
    for a in range(n):
        for b in range(n):
            G[a,b] = sp.simplify(Ric[a,b] - sp.Rational(1,2)*g[a,b]*Rsc)
    return G
```

### 4.2 Field Equation Applications

In LTQG, the Einstein field equations take the form:

$$G_{\mu\nu} = \kappa T^{(\tau)}_{\mu\nu} \tag{12}$$

where $T^{(\tau)}_{\mu\nu}$ is the stress-energy tensor for matter with the scalar field $\tau$ serving as internal time.

## 5 Weyl Transformations in LTQG

### 5.1 Transformation Laws

Under Weyl transformations $\tilde{g}_{\mu\nu} = \Omega^2 g_{\mu\nu}$, the curvature tensors transform according to:

**Theorem 5.1** (Weyl Transformation of Scalar Curvature)**.** *The scalar curvature transforms as:*

$$\tilde{R} = \Omega^{-2} \left[ R - 6\Box \ln \Omega - 6(\nabla \ln \Omega)^2 \right] \tag{13}$$

*where* $\Box$ *is the d'Alembertian operator and* $(\nabla \ln \Omega)^2 = g^{\mu\nu} \partial_\mu \ln \Omega \partial_\nu \ln \Omega$.

### 5.2 LTQG Implementation

For the specific Weyl factor $\Omega = 1/\tau$ in LTQG:

```python
def analyze_flrw_curvature_invariants() -> None:
    """Analyze curvature invariants for transformed FLRW spacetime."""
    t, p = sp.symbols('t p', positive=True, real=True)

    # Original FLRW metric: ds^2 = -dt^2 + t^(2p)(dr^2 + r^2 dOmega^2)
    coords = (t, sp.Symbol('r'), sp.Symbol('theta'), sp.Symbol('phi'))

    # Weyl factor Omega(t) = 1/t
    Omega = 1/t

    # Transformed metric coefficients
    g_tilde = sp.diag(-Omega**2, Omega**2 * t**(2*p),
                      Omega**2 * t**(2*p) * sp.Symbol('r')**2,
                      Omega**2 * t**(2*p) * sp.Symbol('r')**2 *
                      sp.sin(sp.Symbol('theta'))**2)

    print("Scale factor: a(t) = t^p")
    print("Weyl factor: Omega(t) = 1/t")
    print("Transformed metric: g_tilde_mu_nu = Omega^2 g_mu_nu")

    # Compute curvature invariants
    curvature = SymbolicCurvature()
    Ric = curvature.ricci_tensor(g_tilde, coords)

    # Compute scalar curvature
    Rsc = curvature.scalar_curvature(g_tilde, coords, Ric)

    print(f"Regularized scalar curvature: R_tilde = {sp.simplify(Rsc)}")
    print("Expected result: R_tilde = 12(p-1)^2 = constant")
```

### 5.3 Curvature Regularization

**Theorem 5.2** (LTQG Curvature Regularization)**.** *For FLRW spacetimes with scale factor $a(t) = t^p$, the LTQG Weyl transformation yields:*

$$\widetilde{R} = 12(p-1)^2 \tag{14}$$

*This remarkable result shows that the transformed scalar curvature becomes a constant, independent of time, providing natural regularization of cosmological singularities.*

## 6 Validation and Applications

### 6.1 Computational Verification

The LTQG differential geometry module has been validated against multiple test cases:

1. **Schwarzschild Spacetime**: Verification of Kretschmann scalar $K = 48M^2/r^6$
2. **FLRW Cosmology**: Confirmation of curvature regularization for all values of $p$
3. **Weyl Identity**: Symbolic verification of transformation laws
4. **Symmetry Properties**: Validation of all Riemann tensor symmetries

### 6.2 Research Applications

The differential geometry framework enables:

- **Cosmological Singularities**: Analysis of curvature behavior near $t = 0$
- **Black Hole Physics**: Investigation of horizon and singularity structure
- **Quantum Gravity**: Geometric foundations for semiclassical approximations
- **Numerical Relativity**: Robust coordinate choices for computational simulations

### 6.3 Performance Metrics

The implementation achieves:

- **Symbolic Accuracy**: Exact symbolic computation with no approximations
- **Computational Efficiency**: Optimized tensor operations with caching
- **Memory Management**: Efficient handling of 4D tensor structures
- **Numerical Stability**: Robust simplification and factorization routines

## 7 Cross-References and Integration

### 7.1 Connection to Other LTQG Areas

This differential geometry module integrates with:

- **Area 01 (Core Mathematics)**: Provides geometric foundation for log-time transformations
- **Area 03 (Cosmology)**: Supplies curvature analysis for FLRW models
- **Area 04 (Quantum Field Theory)**: Enables curved spacetime QFT calculations
- **Area 06 (Variational Mechanics)**: Supports Einstein tensor computations for field equations

### 7.2 Code Dependencies

Key dependencies include:

```python
from ltqg_core import LogTimeTransform, banner, assert_close, LTQGConstants
import sympy as sp
import numpy as np
from typing import Callable, Tuple, Union, Optional, Dict, List
```

### 7.3 Future Extensions

Planned developments:

- Higher-dimensional spacetime analysis
- Non-commutative geometry extensions
- Numerical integration with finite element methods
- Advanced visualization of curvature invariants

## 8 Conclusion

The differential geometry module of LTQG provides a comprehensive and rigorous framework for analyzing curved spacetime geometry. The implementation combines symbolic computation accuracy with computational efficiency, enabling detailed analysis of curvature properties under Weyl transformations.

Key achievements include:

- Complete symbolic computation of all curvature tensors
- Rigorous validation against analytical results
- Natural regularization of cosmological singularities
- Integration with the broader LTQG mathematical framework

This foundation supports the geometric aspects of quantum gravity research and provides essential tools for analyzing spacetime structure in the LTQG coordinate system.

## References

[1] R.M. Wald, *General Relativity*, University of Chicago Press, 1984.

[2] S.M. Carroll, *Spacetime and Geometry: An Introduction to General Relativity*, Addison Wesley, 2004.

[3] LTQG Framework, *Core Mathematical Foundations*, Area 01 Documentation.

[4] LTQG Framework, *Cosmology and Spacetime*, Area 03 Documentation.

[5] SymPy Development Team, *SymPy: Python Library for Symbolic Mathematics*, 2023.