Felix Pressler

Dark Matter from Quantum Gravity:

Curvature Correlators and Geons in 4d Causal Dynamical Triangulations

Master's Thesis

in partial fulfillment of the requirements for the degree of

Master of Science - MSc

in Physics

submitted to

University of Graz
Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Phys. Dr.rer.nat. Axel Maas

Co-Supervisor

Dipl.-Phys. Dr.rer.nat. Simon Plätzer

Institute of Physics

Graz, March 2025

Abstract

The problem of dark matter remains elusive to this day with many possible theoretical solutions already exhausted. At the same time, quantum gravity, representing another frontier of fundamental physics, is similarly known for high-level solutions and the notorious difficulty to discern between them experimentally. This exploratory thesis aims at bridging the gap between these topics by exploring a possible dark matter candidate arising solely from a quantum gravitational treatment. Working in 4d Causal Dynamical Triangulations (CDT) and building on a recently introduced measure for curvature in this theoretical framework, curvature correlation functions were built and analyzed for possible particle-like properties. It was found that the general behavior of these correlators shows signs of universality across different volume regimes as well as for different curvature operators, hinting at a massive particle possibly representing a Geon. A value for the screening mass was extracted from the data and, using known scaling relations, estimated to be on the order of $10^{-9}\,\mathrm{kg}$, placing it at a tenth below the Planck mass. Finally some tentative remarks were made regarding further properties of these results, indicating possible directions for further research. While speculative, this thesis does hint at a justifiable interpretation along the lines of a massive Geon emerging in CDT, possibly describing a candidate particle for dark matter, strongly motivating further investigations.

Acknowledgements

I want to thank my main advisor Axel Maas for hours of patient explanations, restless motivation and his appreciative guidance, without which I would have never been able to work on such a project. I also thank my co-supervisor Simon Plätzer for his patience with me and his caring explanations, especially in the first few phases of this work, when technical details seemed overwhelming.

Furthermore I am indebted to Dániel Németh for providing us with the simulation code that was used in this work, his kind help at the start of the project and thoughtful discussions of the same, for which I would also like to express my gratitude to Renate Loll. For providing me with a starting point on Dijkstra's algorithm I want to thank David Kneidinger.

Danke Ute, dass du immer Verständnis hattest, auch ohne alles zu verstehen, sogar während wir zu Hause unser eigenes kleines Abenteuer begonnen haben.

To my daughter

Contents

1. Introduction and Motivation

The origin and nature of dark matter is a nearly century-old problem by now and still one of the most pressing questions theoretical and fundamental physics strive to answer. This proposed type of matter is hypothesized to have peculiar properties: It can't be seen but only felt (gravitationally) while at the same time it should exist in such abundance that it outnumbers all of the visible matter roughly by a factor of five [1]. Technically speaking, these observational properties demand that dark matter does not interact electromagnetically but has a mass that significantly contributes to the overall mass of galaxies, that it can clump, that it is stable and generally behaves in ways that have been elaborated in great detail.[1] As of today, a multitude of theories provide possible descriptions for such an unseen gravitational component, be it in terms of modern particle physics or through a modification of gravitation, but due to lack of empirical observations discerning between those theories, a generally accepted explanation remains elusive [3].

While the problem of dark matter emerged from probing gravity at very large scales, another frontier of modern physics deals with the peculiarities of the same fundamental force acting at the smallest scales known to us, where neither general relativity (GR), describing gravity, nor the second fundamental theory of physics, quantum field theory (QFT), can be neglected [1, 4]. While there is currently no generally accepted theory to describe these conditions, it is strongly believed that one has to exist, since our current understanding of the universe provides us with examples of situations where both conditions are met (e.g. in the interior of a black hole or at the big bang) [5]. Thus, it is of great importance for a candidate theory of quantum gravity to make testable predictions that are empirically falsifiable. One such theory that has recently drawn attention to it is called causal dynamical triangulations (CDT).

The main motivation for this present work has been to fuse those two research programs such that a possible candidate particle for dark matter might be found to arise from quantum gravity itself. Our approach provides exploratory investigations into how a particle interpretation might be possible in CDT and how this might link to particles arising from the curvature of spacetime, so-called *Geons*. While many technical details are still open, the following sections provide a comprehensive theoretical overview of CDT itself, how curvature can be defined in triangulated spacetime, how observables and correlation functions are then obtained and how those could be interpreted in terms of particle-like behavior. After reviewing the specific methodology that was used to obtain them, tentative results are discussed and interpreted. Finally, concluding remarks are made and a possible picture for future research in this area is drawn.

[1] see e.g. [2]

2. Theoretical Framework

In this chapter, the theoretical framework underlying the current work is established. It is in the nature of this topic that these explanations can by no means be complete but will consist only of the building blocks necessary for our investigations. Regarding quantum gravity, this will entail a brief overview of the subject with a short introduction into CDT and the possibilities and challenges it poses to our aims. Subsequently a review of possible definitions for physical observables in CDT and QFT is given and general demands for a theory describing Geons and dark matter are addressed.

2.1. Quantum Gravity and CDT

Modern physics rests heavily on two grand theories that have succeeded in explaining practically all phenomena currently testable. On the one hand, there is quantum field theory or QFT, which underpins quantum electrodynamics and the weak interactions as well as quantum chromodynamics, thus leading to theories for electromagnetism, the strong and the weak nuclear interaction, which are also known as three of the four fundamental forces. General relativity or GR on the other hand follows a different mathematical structure to disclose the workings of the fourth fundamental force, gravitation. It has been a long-sought goal for physics to find another theory fusing both quantum mechanics as well as gravity [1, 6]. This striving has produced a multitude of different approaches on how to tackle this problem and while they are all worthwhile studying, for the purpose of this work we will focus on a single path that leads to the theory of causal dynamical triangulations.

A common starting point for most quantum field theories has been the path integral formalism, in which physical statements are obtained by evaluating the integral over field configurations for a specific action [7]. The classical theory of general relativity provides such an object in form of the Einstein-Hilbert action [1]

$$S_{EH} = \frac{1}{16\pi G} \int d^4x \sqrt{g}(R - 2\Lambda) \equiv \int d^4x \sqrt{g} M_{pl}^2 (R - 2\Lambda) \tag{1}$$

where G is the gravitational constant, g is the determinant of the metric tensor $g_{\mu\nu}$, R is the Ricci scalar, which will be addressed later on, and Λ denotes the cosmological constant. The constant prefactors can be absorbed in the square of the Planck mass M_{pl}^2, as is shown on the right hand side. A naive way to proceed to a quantum theory of general relativity (and thus gravity) would be to plug this formula into the standard path integral prescription. With a bit of care, one then obtains the partition function Z as [8]:

$$Z = \int_M \mathcal{D}[g_{\mu\nu}] e^{iS_{EH}[g_{\mu\nu}]} \tag{2}$$

Here it is evident that the metric $g_{\mu\nu}$ itself is interpreted as the field and integration is

done over the space of all possible metrics for a given manifold M.[2] The problem at the heart of this equation is that this path integral is – within perturbation theory – non-renormalizable, because a coupling with negative mass dimension -2 appears through G [9]. Moreover, within such a framework the concept of locality is becoming increasingly intricate, since simple computations show how coordinates in local observables can be easily exchanged [1]. Generally speaking, the notion of distance itself must become an expectation value of some sort [10].

CDT provides a way to approach this prescription and make it amenable to computer simulations. It does so, roughly speaking, by discretizing the integral such that it becomes a sum over possible spacetime geometries. To this end (and in the spirit of similar lattice methods in QFT), a spacetime lattice is introduced where the lattice spacing a acts as a regulator, providing a UV cutoff.[3] The chosen lattice type is a triangulation, for which it was already shown by Regge that a natural implementation of the Einstein-Hilbert action exists [8, 11]. Unlike previous attempts to explore quantum gravity using lattice methods, this approach brings several promising features with it, two of which are exceptionally different to other methods that might come to mind:

- Through the usage of Regge calculus, distinct continuous geometries are produced without the need to introduce a coordinate system at any point. This guarantees inherent diffeomorphism invariance.
- In contrast to perturbative approaches to quantum gravity discretization, which mostly rely on a fixed background, the lattice itself is the dynamical quantity in CDT. Thus, the theory is manifestly background independent and non-perturbative.[4]

Using this methodology, the discretized partition function for a specific lattice constant a then becomes [8]

$$Z_a = \sum_T \frac{1}{C_T} e^{-S_E^{\text{Regge}}[T]} \tag{3}$$

where summation is done over all possible triangulations T. The combinatorial factor C_T is the number of elements in the automorphism group of T and accounts for over-counting of differently labeled but otherwise identical triangulations. The weight factor corresponds to the usual Boltzmann weight after Wick rotation of the action, which is given by an adapted Regge-version of the Einstein-Hilbert action [8, 12]. This quantity and its dependencies will be discussed in the following, where we will give a concise overview of how calculations are done using this framework.

[2] The square brackets indicate that integration is actually performed over all diffeomorphism equivalence classes of metrics

[3] CDT is not a discrete theory of gravity, since physical quantities are obtained in the continuum limit $a \to 0$. In abuse of language, the limit $ds^2 \to 0$ is henceforth denoted as "UV", analogous to conventional lattice theory.

[4] Note that there exist background-independent perturbative approaches, from which CDT differs greatly, however, through its reliance on the *whole* quantum geometries to describe the path integral.

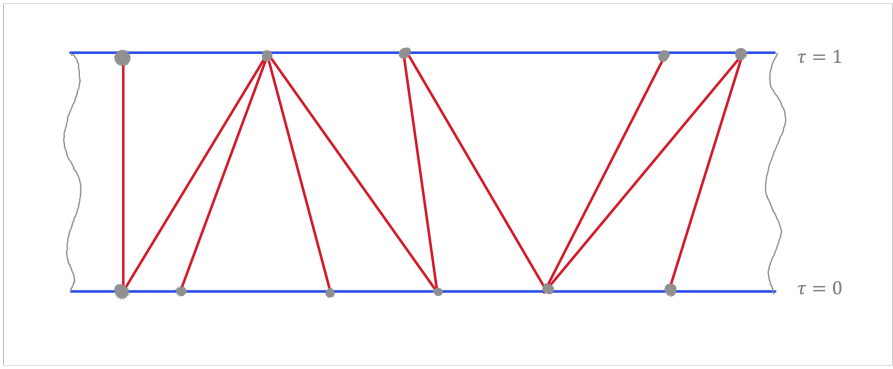

Figure 1: Illustration of CDT geometries in 1+1 dimensions

Time flows upwards along the vertical direction, the horizontal lines are slices of constant proper time, also called timeslices. On these slices, all links between vertices are spatial. The temporal links are shown in red and the vertices are denoted as grey dots. A 2-simplex is then formed by combination of two temporal and one spatial link. This band structure extends vertically to further timeslices and horizontally to further vertices. For a spherical topology, vertical and horizontal ends are glued together (identified with each other). Note that all red and all blue lines have, in fact, the same physical length (either timelike or spacelike link length) and the distortion shown in this figure hints at the nontrivial curvature of the geometry.

Doing CDT

This section follows closely [8, 12–15] as well as the lecture notes [16–18].

In CDT, spacetime is built as an undirected graph using standard building blocks consisting of vertices and links. Together, they form *simplices*, which are the d-dimensional generalizations of triangles. Two different edge lengths occur in the graph, each one being associated with either spatial or temporal edges:

$$l^2_{\text{spacelike}} = a^2 \tag{4}$$

$$l^2_{\text{timelike}} = -\alpha a^2, \quad \alpha > 0 \tag{5}$$

A notion of proper time is induced through a foliation of spacetime, assuring that the vertices all lie on slices of constant proper time.[5] In this way, the resulting simplices are glued together on a fixed topological manifold, which is often chosen to be either a sphere or a torus, providing periodic boundary conditions. The simplices are always situated between two timeslices and can be oriented in different ways, depending on the number of vertices they share on each timeslice. An illustration of this concept is found in fig. 1 for the 2d case. If one denotes the number of vertices a simplex shares on slice τ and $\tau+1$ as (n, m), the 4-simplex (which is used in 4d CDT) can be oriented between timeslices in the following ways: $(4, 1), (1, 4), (3, 2)$ and $(2, 3)$. The aforementioned action then depends

[5] Such a slice can be identified as a spatial slice because of this feature. Since it is a slice of constant time, it will be also called *timeslice* in the following.

on the triangulation through the number of simplices sharing an orientation as well as three other quantities: the bare gravitational coupling, the bare cosmological constant and the ratio between edge lengths (spatial and temporal) $-\alpha$. After deciding on those constants as input parameters, the last step is to perform a random walk through the space of possible geometries using Markov Chain Monte Carlo techniques with a fixed set of possible moves to update the local geometry of the triangulation. The aim behind this is to generate a sample of an ensemble of triangulations such that the probability to draw a specific triangulation T from it approaches the distribution [8]

$$P(T) = \frac{1}{Z} e^{-S_E[T]} \tag{6}$$

The action for the 4d case can be expressed as [8]

$$S_E = -(\kappa_0 + 6\Delta)N_0 + \kappa_4(N_4^{(4,1)} + N_4^{(3,2)}) + \Delta(N_4^{(4,1)} + N_4^{(3,2)}) \tag{7}$$

where the κ_i depend on the bare inverse Newton constant κ and bare cosmological constant λ, Δ is the asymmetry parameter capturing the geometric impact of different link lengths, N_0 denotes the number of vertices in a configuration and the N_4 denote the number of simplices of different type. Through its dependence on these input parameters, CDT operates in an intricate three dimensional parameter space. Through fine-tuning of κ_4 to its critical value in actual computations, this can be reduced to two dimensions, where the theory is well described solely by the two parameters κ_0 and Δ.

This shows, in essence, how CDT serves as a computational tool to explore quantum gravity at the Planck scale [8, 12–14].

Here it is worth to note that depending on the point in phase space under investigation, CDT configurations[6] exhibit different behavior. One of the findings so far has been that there exists a certain phase, where typical configurations seem to emerge in a de Sitter like shape, with recent research suggesting the validity of a physical interpretation in terms of de Sitter geometry [8, 12, 19]. This discovery is a major pillar of this current work, where the first steps have been to recreate some features of those geometries as consistency checks, as will be shown in section 3.1. Recall that a de Sitter universe is characterized by a continuous expansion and a constant, positive scalar curvature [6]. It can also be described as the coset manifold of two Lorentz groups, analogous to the group theoretic description of a general sphere [20]. Thus, the validity of describing a certain geometry as de Sitter like depends heavily on its curvature properties. The contents of section 2.2 will therefore deal with ways to define notions of curvature on discrete spaces and how this is eventually done in CDT.

CDT in the broader picture of quantum gravity

While the first glimpse may raise questions regarding the validity of assumptions underlying the CDT approach to quantum gravity, predictions made using this framework

[6] Is used synonym to *triangulations* or *universes*.

seem to hold up well against other approaches so far. Not only has there been increasing evidence for the CDT prediction of a "fractal" universe with a dynamical reduction in dimensionality from four to two for small geodesic distances, but recent research also made progress in identifying limits of the theory with those evaluated through a different approach to QG, relying on functional renormalization group (FRG) techniques [8, 21]. Within this scheme, a so-called effective action is derived for quantum gravity and assumed to be a function of a certain scale factor k, giving the IR and UV limits of the theory for $k \to 0$ and $k \to \infty$ respectively.[7] One hope is that such a limit could be identified with a UV fixed point of the renormalization group, leading to a consistent and predictive theory of quantum gravity in spite of the perturbative non-renormalizablity, which is also called the asymptotic safety scenario. There are already strong hints that such a UV fixed point might exist for quantum gravity, while so far only the IR limit of CDT could be identified with the result from the FRG approach [21, 23]. In any case, CDT is expected to at least give an effective description of physics at the scale of a few Planck lengths, mostly independent of what theory is exactly realized [8].

2.2. Curvature in Triangulated Spacetime

There is a notion of curvature that goes back to Leonhard Euler and is still the most prevalent concept when discussing the topic: That curvature is – in some way or another – linked to the second derivative [24]. This idea becomes troubling when one considers discretized geometries like it is done in this work, where a suitable definition of a second derivative of some kind might be hard to find. Thus, in this chapter, we will address this problem and provide a solution to how curvature can still be measured in triangulated spacetime.

2.2.1. Curvature in GR

The central object in GR, one could say, is the Riemann tensor $R^{\lambda}_{\rho\mu\nu}$. This quantity solely depends on the metric, but it also relies on the concept of *Gaussian curvature*, which is an intrinsic property of a surface embedded in three dimensional space and given by the product of its two principal curvatures. The principal curvatures are the maximum and minimum measures of departure of a smooth curved surface from a flat surface tangent to it at any given point [6]. Using this, the Riemann tensor can then be thought of as the collection of Gaussian curvatures of all planes containing a given vector. This is of course just one interpretation with the most common being that $R^{\lambda}_{\rho\mu\nu}$, on any general Riemannian manifold, measures the failure of the second (covariant) derivatives to commute at any point, hinting again at the connection between curvature and the second derivative.

If we stay in the previous picture, we can define two more quantities which are used heavily in GR. The first one is the first contraction of the metric and the Riemann

[7] see [22] for further information.

tensor, also called the Ricci tensor $R_{\mu\nu} = R^{\lambda}_{\mu\lambda\nu}$. It can be thought of as an average over this collection of Gaussian curvatures. Contracting it a second time then gives the Ricci scalar or curvature scalar $R = g^{\mu\nu}R_{\mu\nu}$ [25].

2.2.2. Discrete Curvature

This section as well as section 2.3 follow closely [19, 26, 27].

In Regge calculus, GR's natural discrete continuation, curvature is measured in terms of *deficit angles*. Within this concept, curvature is located at the *hinges* of the triangulation edges, and is given by the deficit of an angle sum around such a hinge with respect to flat space. The hinge is the $(d-2)$-dimensional subspace of the given manifold. This is, for example, a vertex (0-dimensional) on a 2d-manifold built from triangles or a triangle on a 4d-manifold built from simplices. The deficit angle of, say, a vertex on a 2d-manifold is, pictorially speaking, the angle one would obtain if one where to cut open one of the neighboring edges, lay the resulting tile on a flat surface and measure the angle between the open edges (as seen in fig. 2). It is the difference of 2π (what one would expect in a flat 2d case) and the angle sum around the vertex. One feature of this prescription is that the sum over all deficit angles δ_i for a small region on a given manifold is equal to half the integral over the curvature scalar of the original smooth manifolds region, giving the relation

$$\sum A_i \delta_i = \frac{1}{2} \int R\sqrt{-g} d^4x \tag{8}$$

for the 4d case, where the A_i are the area contents of the triangles building the region summed over [6].

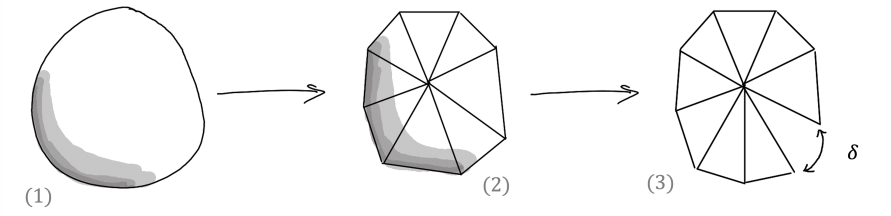

Figure 2: Deficit angle curvature

In this example, a 3d object (1) is approximated by a triangulation (2) to produce a geometry of similar curvature, indicated by the shadow. If one were to cut open the triangulation at any edge and lay the resulting tile flat on the ground (3), the angle between the open edges is called the deficit angle δ and is a measure of the curvature at the central vertex.

The problem with deficit angle curvature is that while it works well for the case of triangulated spacetime, it tends to be highly divergent in the continuum limit for general manifolds, which is a crucial step in the theory of CDT. This is the case because as the lattice spacing a approaches zero, the density of curvature defects grows while the

individual deficit angles do not average out. To remedy this, one needs to work with an alternative concept of curvature. A method that has produced promising results so far is built on another interpretation of curved spaces and was only recently introduced in the literature as *Quantum Ricci Curvature* [12, 19, 26, 27].

This method is built on the observation that on positively curved manifolds, balls (and spheres) are closer than their centers are [28].[8] This in turn can be used to define a measure of curvature in the following way: Consider two points on a smooth D-dimensional Riemannian manifold p and p' around which two spheres $S^\epsilon_p, S^\epsilon_{p'}$ of radius ϵ are built such that they contain all points that are a geodesic distance ϵ away from their centers. The centers themselves have a geodesic separation of $\delta \geq 0$. If one maps one sphere onto another using parallel transport and considers the limit $(\delta, \epsilon) \to (0, 0)$, one finds that the average distance between two points on those spheres becomes [27]

$$d\left(S^\epsilon_p, S^\epsilon_{p'}\right) = \delta\left(1 - \frac{\epsilon^2}{2D}\mathrm{Ric}(v, v) + O(\epsilon^3 + \delta\epsilon^2)\right) \tag{9}$$

where $\mathrm{Ric}(v, v)$ denotes the Ricci curvature of a unit tangent vector v at p. To make this expression practically computable in the framework of CDT, a slight change is made in that the spheres S^ϵ_p are now defined as the set of all points that are a distance ϵ apart from the center p, whether or not they really form a sphere in the topological sense.[9] Through this, the average sphere distance for a D-dimensional Riemannian manifold can be defined as [27]

$$\bar{d}\left(S^\epsilon_p, S^\epsilon_{p'}\right) \equiv \frac{1}{vol(S^\epsilon_p)}\frac{1}{vol(S^\epsilon_{p'})}\int_{S^\epsilon_p} d^{D-1}q\sqrt{h}\int_{S^\epsilon_{p'}} d^{D-1}q'\sqrt{h'}\, d(q, q') \tag{10}$$

where

$$vol(S^\epsilon_p) = \int_{S^\epsilon_p} d^{D-1}q\sqrt{h}, \quad vol(S^\epsilon_{p'}) = \int_{S^\epsilon_{p'}} d^{D-1}q'\sqrt{h'} \tag{11}$$

with the volumes of the spheres $vol(S)$, their metric determinants h and h' and the geodesic distance $d(q, q')$ between points q and q' on the spheres. Here it should be noted that $\bar{d}$ is not a proper mathematical distance, since it does not vanish in the case of $p = p'$ unless for $\epsilon = 0$. A straightforward discretized lattice implementation of eq. (10) is found in [27]

$$\bar{d}\left(S^\epsilon_p, S^\epsilon_{p'}\right) = \frac{1}{N_0(S^\epsilon_p)}\frac{1}{N_0(S^\epsilon_{p'})}\sum_{q\in S^\epsilon_p}\sum_{q\in S^\epsilon_{p'}} d(q, q') \tag{12}$$

[8] To illustrate this, it might be helpful to imagine the surface of an upright balloon with two circles drawn on it on a horizontal line. If one now draws two additional vertical straight lines along the surface of the balloon and through the centers of both circles, one might find that the points on the circles where the lines intersect are a smaller geodesic distance apart than their centers. This effect is seen solely through the curvature of the balloon. This principle is also depicted in fig. 3.

[9] On typical CDT configurations, in general, they don't, but they rather form disconnected spaces.

where the volume is denoted as the number of vertices contained in a sphere $N_0(S_p^\epsilon) = \sum_{q \in S_p^\epsilon} 1$ and summation is done over all points on the spheres. From this, one finally arrives at the following expression for the *average normalized sphere distance* [27]:

$$\frac{\bar{d}(S_p^\delta, S_{p'}^\delta)}{\delta} = c_q(1 - K_q(p, p')) \tag{13}$$

Here, the radius of the spheres has been set equal to the geodesic distance between their centers $\delta = d(p, p')$, c_q is a positive constant dependent on the geometry of the metric space and $K_q(p, p')$, capturing all dependencies on the Ricci curvature, has been dubbed the Quantum Ricci Curvature (QRC). The associated quantum Ricci scalar can be obtained by letting $p = p'$ for the 2d case, thus setting the two spheres equal to one another [26, 27]. This version of eq. (12) and eq. (13), where the average normalized sphere distance then only depends on a single point p is the one that was used in this work. There is a considerable amount of detail involved in the exact systematics of how to calculate such a quantity on a computer, which will be further explained in section 3.

On a general smooth 4d Riemannian manifold and in the limit $\delta \to 0$, the average normalized sphere distance can also be computed numerically by usage of Riemann normal coordinates as was done in [29]. Expressed as truncated power series in δ it was found to be:

$$\frac{\bar{d}}{\delta} = 1.6524 + \delta^2(-0.0469\, Ric(v, v) - 0.0067\, R + \mathcal{O}(\delta)) \tag{14}$$

where two distinct contributions appear from the Ricci tensor $Ric(v, v)$ as well as its trace, the Ricci scalar R. This numerical benchmark can act as a first marker to compare our results against.

2.3. Observables and Correlators in QFT and CDT

To narrow in on the goal of this thesis, it is crucial that a solid foundation for a possible particle interpretation of our findings is built. To start with this, recall that the main quantity of interest in QFT is most often the vacuum expectation value of some observable $\mathcal{O}$. In CDT, the corresponding regularized quantity is obtained as [8]

$$\langle \mathcal{O} \rangle_a = \frac{1}{Z_a} \sum_T \frac{1}{C_T} e^{iS[T]} \mathcal{O}[T] \tag{15}$$

and is assumed to be related to continuum observables via a certain scaling relation [8]. For our further investigations, however, we will need a concept describing two-point correlation functions or *propagators* in a similar fashion. Generally, continuum expressions for such objects can be defined in non-perturbative quantum gravity in the following way [26]:

$$\langle G[\mathcal{O}, \mathcal{O}(r)] \rangle = \int \mathcal{D}[g] e^{iS[g]} \int_M d^4x \sqrt{|g(x)|} \int_M d^4y \sqrt{|g(y)|} \mathcal{O}(x)\mathcal{O}(y)\delta(d_g(x, y) - r) \tag{16}$$

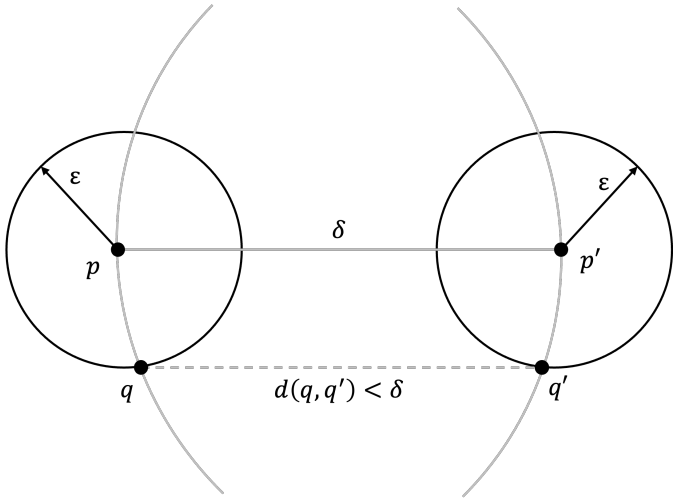

Figure 3: Spheres in positive curvature

Two spheres on a positively curved manifold. All grey lines are geodesics, appearing curved through their projection (curvature is exaggerated). Both spheres have the same radius ε and are separated by a distance δ. The two bottom points q and q' are shown to have a smaller separation $d(q, q')$ than the sphere centers, even though they are connected to those via a geodesic. This effect illustrates the curved nature of the manifold.

Here, $\mathcal{O}$ is a local scalar quantity (like the curvature scalar we will examine later on) and r is the geodesic distance between x and y. The integration over both spacetime points has to be performed in order to obtain a diffeomorphism-invariant quantity, such that an interpretation as physical object is valid. The connected version of this correlator can be obtained by substituting an operator in eq. (16) with the deviation from its average,

$$\mathcal{O}(x) \longrightarrow \left(\mathcal{O}(x) - \overline{\mathcal{O}}\big|_g\right) \tag{17}$$

where notation already hints at the fact that $\overline{\mathcal{O}}\big|_g$ also depends on the metric. In general it should be noted that two-point functions in quantum gravity depend on the metric in a threefold way, namely through the two operator values as well as the geodesic distance $d_g(x, y)$.[10]

In the discretized lattice setting of CDT, averages over the manifold become averages over triangulations and the classical propagator becomes the sum

$$G_T[\mathcal{O}, \mathcal{O}](r) \equiv \sum_{x,y \in T} \mathcal{O}(x)\mathcal{O}(y)\delta_{d(x,y),r} \tag{18}$$

To then extract the expectation values, one averages over fixed-volume ensembles of triangulations (denoted as $\langle\cdot\rangle$) before or after normalization. Using the average normalized

[10] This of course also implies that nontrivial correlations involving the unit operator should be expected since even $G[1, 1](r)$ depends on the metric through the geodesic distance r.

sphere distance defined above (abbreviated as $\bar{d}/\delta$), a possible choice for a normalized curvature correlator could thus be [26]:

$$\mathcal{G}[\bar{d}/\delta, \bar{d}/\delta, r] \equiv \frac{\langle G_T[\bar{d}/\delta, \bar{d}/\delta](r)\rangle}{\langle G_T[1,1](r)\rangle} \tag{19}$$

2.4. Geons and Dark Matter

To close the loop opened up in the introduction, the last step is to integrate the propagators we have defined in the previous section into a suitable particle picture. For this, we should first look back at what we want to arrive at in the end: a candidate particle for dark matter.

Dark matter has a long history of eluding detailed physical description. Nevertheless, piles of observational evidence for it were gathered over the course of nearly a century, with data from galaxy clusters and the cosmic microwave background strongly narrowing in on the allowed properties for dark matter. As already mentioned, it needs to satisfy two criteria first and foremost [3]:

1. If it interacts with ordinary Standard Model matter, it does so very weakly except for its gravitational interaction, justifying the adjective *dark*.
2. Nevertheless it has to be abundant in the observable universe with current estimates holding it responsible for about 84% of the total observable mass density. Furthermore it needs to be stable on cosmological timescales [2].

These allow for a vast variety of dark matter models relying on extensions of the Standard Model of particle physics, the introduction of new gauge sectors or even modified gravitational theories, though ongoing discussions are held about the constraints on those models.[11]

One interesting candidate, that would in a sense bridge the gap between different viewpoints on dark matter is the gravitational Geon, described in [31, 32] and proposed as a candidate particle among others in [10]. While classical treatments of the Geon have questioned its stability [33], the possibility of a massive and stable Geon emerging from a quantum theory of gravity is what makes it an intriguing prospect.

The original idea of a gravitational Geon is that of a bound state of gravitational waves, acquiring mass from their mutual gravitational interaction inside a spatially bounded region. For this, one must define a particle propagator via a suitable observable, such that it is invariant under gauge transformations as well as diffeomorphisms, which can be regarded as a guiding principle to building physical objects in quantum gravity [10]. A straightforward observable meeting our requirements is the Ricci scalar R, giving

$$D(x,y) = \langle R(x)R(y)\rangle \tag{20}$$

[11] see e.g. [2, 30].

for a possible particle propagator definition. Note that x and y, as was the case previously, denote spacetime events on a manifold, not the coordinates that can be associated with them. It is our working assumption that within a reasonable range of accuracy, this quantity should be well described by the correlators built from the average normalized sphere distance in section 2.3.

One caveat, which also separates this work from previous studies on such correlators, is that for a valid particle description to apply, timelike and spacelike direction need to be considered independently. As a result, the correlators we will study in the following show a spatial or temporal resolution:

$$C(r) = \langle R(\tau_0, x) R(\tau_0, y) \rangle, \quad d(x, y) = r, \tag{21}$$

$$C(\delta\tau) = \langle R(\tau_0 - \delta\tau, x) R(\tau_0 + \delta\tau, y) \rangle \tag{22}$$

Especially correlators of the first type (eq. (21)), giving the correlation between two events on a fixed timeslice τ_0 and separated by a fixed geodesic distance r will be investigated. For those, we will aim to extract a so-called screening mass from the asymptotic behavior of the spatial correlator by fitting it against an exponential function, inspired by usual methods in lattice QCD [34, 35]:

$$C(r) \longrightarrow a + b \cdot e^{-m \cdot r} \tag{23}$$

where m is the screening mass, which is assumed to hold as a proxy for the pole mass for the purpose of this study.

3. Methodology

To arrive at the results we promised in the last section, the first step is to create a sufficient amount of triangulations such that observables can be measured in a statistically meaningful way. The triangulation samples for our results were obtained through cluster computing simulations of 4d CDT, using a simulation code written by Andrzej Görlich and provided by Dániel Németh [36]. Building on this, routines to analyze the resulting geometries, measure geodesic distance and curvature in different ways and plot the results were implemented in a code base written in Python and C++, constituting the main part of this present work. The main analysis code used for computing the raw data is attached in appendix A.

3.1. Reproducing 4d CDT findings

Working with CDT, one finds quickly that it might feel quite unusual to work without any type of coordinate system. This circumstance, however, is needed to preserve diffeomorphism invariance in the simulations. As a result, data describing the configurations is stored in a sparse but comprehensive way:[12] Vertices and simplices have progressive integer labels that are stored in two lists, comprising all the necessary information to reconstruct the respective geometries. While the first list assigns a timeslice to each vertex in the triangulation, the second list describes the simplices building the geometry through integer numbers that come in blocks of ten. In each block, all numbers are to be thought of as labels with the first five denoting the vertices building the simplex and the last five numbers referencing the simplices neighbouring the current simplex.[13]

To check that our understanding of the simulations and their corresponding output reproduce the correct geometry, one of the first findings of this work was the independent validation of a central result in CDT, which is that there exists a section in parameter space where typical geometries produced by the simulation exhibit de Sitter like behavior. To assess fully whether this assertion is true, it is of course necessary to include the curvature properties of the sample universes into our considerations, which will be done later on. For now, we will reduce our explorations to suitable proxies.[14]

The mean number of vertices residing on a specific timeslice is assumed to be such a proxy, since this number can be directly related to the number of spacetime events mapped onto the triangulation. As such, a plot of these should give a first impression of the volume change of the geometries with the passing of proper time. Indeed, the characteristic shape of the de Sitter-style universes produced in the corresponding phase of CDT can already be inferred from the plot in fig. 4. Here, two regions can be roughly defined: The

[12] Further information on data storage and simulation techniques in CDT can be found in [13].

[13] In 4d CDT, 4-simplices always consist of five vertices and have five neighbors, resulting in the fitting synonym 5-cell or pentachoron [37].

[14] All of the following exemplary plots are done using a typical single configuration from the de Sitter phase with parameters ($\Delta = 0.6, \kappa_0 = 2.2$), temporal extension $\tau \in \{1...80\}$ and simplicial volume $N_4^{(4,1)} = 160000\,(160k)$.

so-called *stalk* of the universe, which consists of the two long strands at the beginning and the end of the plot where little vertices reside and the *bulk* region, surrounding the largest timeslice with respect to vertex number. Note that the corresponding proper time associated with this largest slice is not constant but changes with each Monte Carlo snapshot.

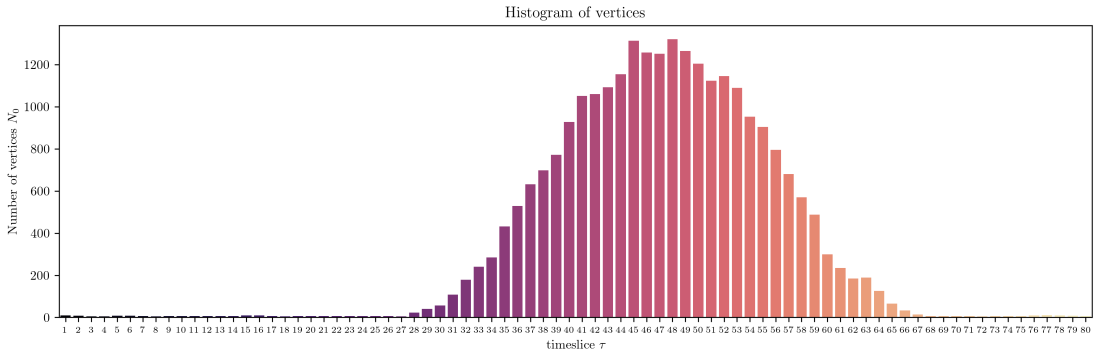

Figure 4: Histogram of vertices for sample configuration.

Another possible measure for the spatial extension of the universe can be found if one considers the geodesic distance between vertices residing on the same timeslice. The expectation is that with increasing volume, the distance between two random vertices will, on average, also increase. This has been tested for the sample configuration described above by analyzing the shortest path for $n = 100$ pairs of random vertices on each timeslice and the resulting histogram, verifying our assumption, is shown in fig. 5. The shortest paths in this discrete setting are measured as the sum of steps between the vertices, where for the current assessment, no differentiation between spacelike and timelike links was made. The method of choice for finding those paths is Dijkstra's algorithm which will be further explained in the next section.

As a last hint towards a proper interpretation of de Sitter phase universes, the lightcone growth for single vertices can be examined. If probed at the bulk of the universe for a random vertex, its lightcone is expected to grow exponentially at first and finally reach a plateau in the stalk. Such a behavior reflects the causal structure of the triangulations and is depicted in fig. 6.

3.2. Measuring Geodesic Distances and Curvature

3.2.1. Regular and Dual Lattice

All of the analyses done in the previous section relied on the regular lattice of vertices and its conversion into a linked-list representation. While this is not only computationally expensive given the storage format of the geometries described earlier, working on the regular lattice also imposes major numerical limits on the extent to which observables can

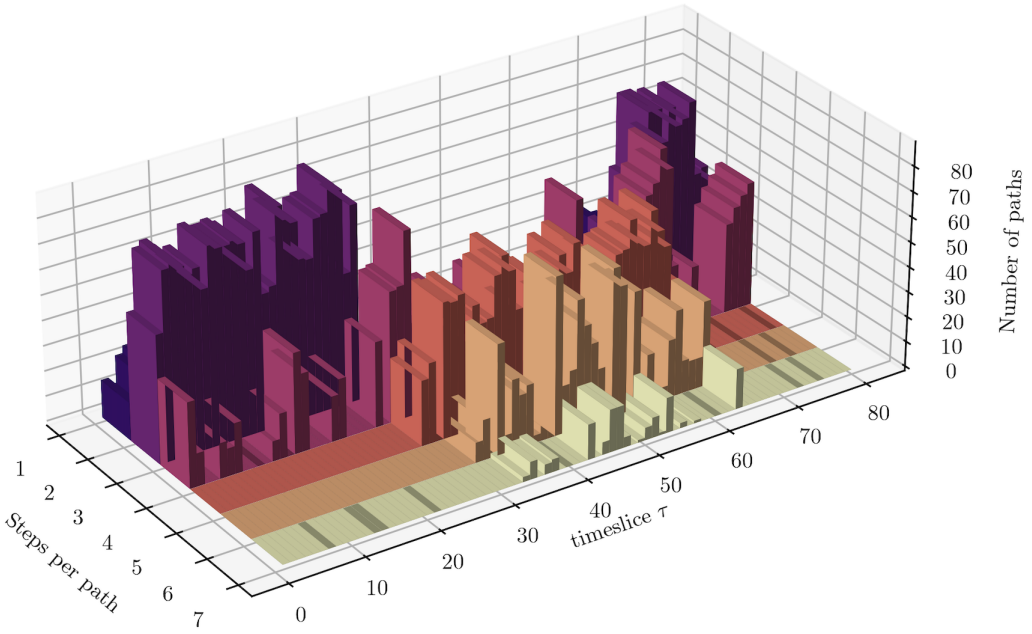

Figure 5: Histogram of shortest paths expressed as sum of steps between $n = 100$ random vertex pairs on each timeslice.

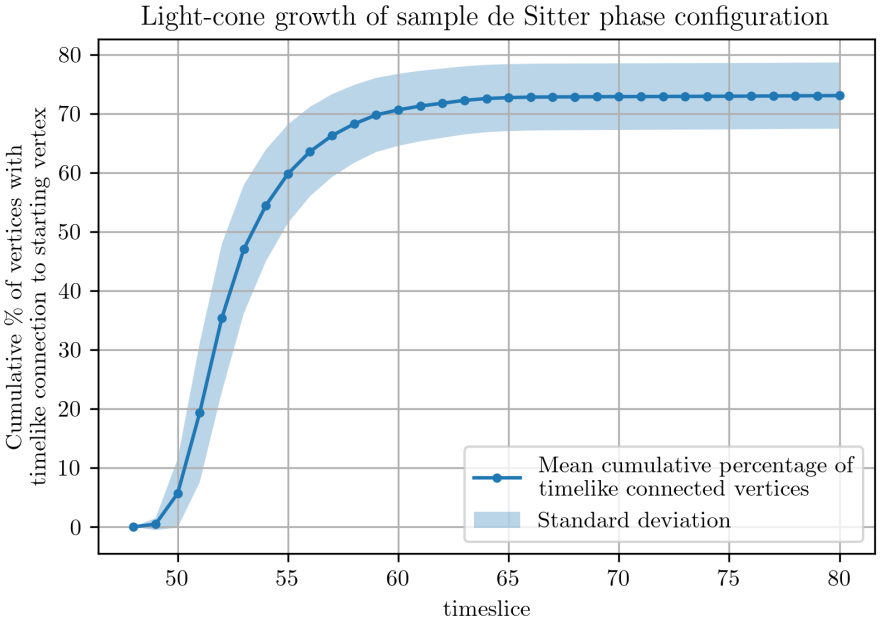

Figure 6: Lightcone structure for vertices taken from the largest timeslice of the sample configuration.

be computed on contemporarily available triangulations. Building on [19], the following observables are thus evaluated not on the regular lattice built from vertices (the vertex lattice), but on its dual, which relies on the simplices as building blocks. On it, nodes are defined as the centers of simplices and links as the lines connecting them. Working on the dual lattice instead of the regular lattice is assumed to have no impact on the final physical interpretation of findings since the physical results should not depend on the discretization used.[15] At the same time, it provides more structure to evaluate on, since the number of simplices on a typical configuration is at least an order of magnitude larger than its vertex content. This is highly needed, for example, when calculating curvature spheres as described below. Since the universes used have a finite spatial extent, those spheres would wrap around single timeslices of vertices easily when the radius used is big enough, distorting the measured curvature [19].

A minor shortcoming of working on the dual lattice is that it lacks the predefined timeslices we use on the vertex lattice, since simplices, per definition, always reside between timeslices. For our aim of investigating the differences between spatial and temporal directions in CDT, a suitable definition for dual timeslices is therefore needed. For the current work, we define a dual timeslice t as the set of all simplices whose vertices lie on timeslices $t = \tau$ and $\tau + 1$.

3.2.2. Dijkstra's Algorithm

Before we discuss the technicalities of calculating curvature on CDT triangulations, the measurement of geodesic distances needs to be addressed. Working in quantum gravity means that distances between two spacetime events x and y become expectation values. This is accounted for in CDT by design, since there is a priori no meaningful way to track single vertices across multiple triangulations from the Monte Carlo history and every physical quantity is extracted by taking averages over these. However, on a single configuration, vertices are of course labeled and distances can be computed as the sum of links connecting the points. Since the geometry one works with after Wick rotation is described by an undirected graph, a straightforward way to compute these geodesics is found in *Dijkstra's algorithm* [38].[16] A version of this algorithm is implemented in appendix A.

Using Dijkstra's algorithm, the average normalized sphere distance described in eq. (13) can be computed in a straightforward way, which is also illustrated in fig. 7. For this, a point p is picked from the triangulation (either from the regular or the dual lattice) and a sphere of radius δ is constructed as a list of points p' that have a link separation δ to p, corresponding to demanding the geodesic distance in eq. (12) be $d(p, p') = \delta$. Dijkstra's

[15] There is evidence in favor of this assumption as well as persuasive insights into why working on the dual lattice is more suitable for measuring curvature. Both can be found in [19].

[16] Note that after Wick rotation, all links in the graph have a positive length but the final results will still carry a dependence on α (the ratio between spatial and temporal link length) through the selection of geometries that has happened [8]. For our investigations, the graph is taken to be an undirected one with equal link weight.

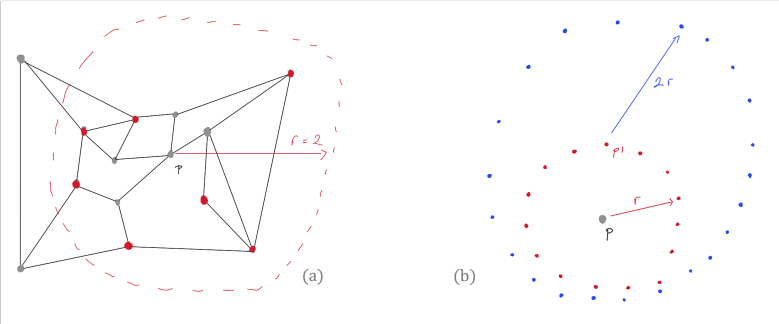

Figure 7: (a): The concept of "sphere" is stretched when working with triangulations to include any set of points with a given distance to an origin point. Here this is illustrated for a general undirected graph, where the set of red points denote the sphere around p with radius $r = 2$.
(b): Mode of computation for average sphere distance: A point p is picked from the triangulation and after computing the sphere with radius r around it, a sphere of radius $2r$ is computed around each point p' on the original sphere. Thus, the geodesic distance does not need to be computed for every pair of points but can be accessed easily using the maps produced by Dijkstra's algorithm.

algorithm computes the link distance for a given origin radially outwards, so after δ iterations, the process can be stopped and a value for each point in the ball around p is saved, from which the sphere can then be determined. The average normalized sphere distance is defined as the average distance between all points on the sphere divided by the common radius around p, so the next step consists of computing the geodesic distance (link seperation) of all point pairs and summing over them. This can, in principle, be done by using Dijkstra's algorithm to obtain a value for each possible pairing at once. However, this is computationally very expensive. A fast way to obtain the same results in less time was found to consist of the following process: For each point p' on the δ-sphere around p, let Dijkstra's algorithm run through the graph for 2δ steps. This assures, that every point on the sphere is included in this bigger "Dijkstra-ball", from which the geodesic distance for each point is readily accessed after completion. The average normalized sphere distance is then computed as the running average of those distance sums for each point divided by δ. Note that this general procedure is the same whether one chooses to work on the regular lattice or the dual lattice. The numerical implementation of this technique is found in the functions `avg_sphere_dist` and `dijkstra` in appendix A.

3.3. Building Correlation Functions

Building on the observables we have touched upon so far and the general prescription for defining correlation functions that was given in section 2.3, it is now time to tend to the details of how the correlators necessary for our pursued investigations are calculated in

CDT. The common procedure that was followed for practically all observables occurring in the current work consists of the following steps:

1. Take a sample of configurations from the Monte Carlo history of a simulation run with fixed parameters.
2. Calculate the operator value for a sample of points (most often the simplex nodes on the dual lattice) for each configuration.
3. Take the mean over all datapoints from all examined configurations.

The last step gives the final result, where we identified its error as the standard deviation of measurements divided by the square root of the number of datapoints (i.e. the standard error). For composite observables, the errors were derived through common error propagation.

As was stated already, the novelty of this work is mostly given through the fact that spatial and temporal directions are treated independently, leading to some subtle deviations from the procedure to measure the average normalized sphere distance described in the previous section. First, the dual timeslices for each configuration have been relabeled, starting with the biggest dual timeslice in terms of its vertex content and proceeding cyclically onward in the direction of proper time. This is to ensure comparability between different universes, since the exact position of the biggest slice changes for different Monte Carlo snapshots but we expected the topology to be de Sitter-like throughout the samples.[17] The second necessary change has been to constrain the method for calculating geodesic distances (i.e. Dijkstra's algorithm) to simplices from the common dual timeslice in order to obtain space-like distances only. This was done by reducing the adjacency list describing the original graph – and with it the geometric structure of the universe – to, for each simplex, include only those neighboring simplices whose constituting vertices share the same regular timeslices, such that no links exist between simplices of different dual timeslices. This reduced graph has then been used to calculate the spatial distance between two simplices on a common dual timeslice t. Calculations regarding the average sphere distance (especially constructing the δ-spheres around the simplices) resorted to the regular, whole graph, in order not to exclude information about the local curvature.

The starting point for most operators examined was the calculation of the average normalized sphere distance, for which eq. (12) and eq. (13) dictate the following equation on the dual lattice:

$$\frac{\bar{d}}{\delta}(s_i) = \frac{1}{N_0(S^\delta_{s_i})^2} \sum_{s_j \in S^\delta_{s_i}} \sum_{s_k \in S^\delta_{s_i}} d(s_j, s_k)/\delta \tag{24}$$

Here, $S^\delta_{s_i}$ describes the sphere of radius δ around a simplex s_i, N_0 counts the number of

[17] Note that the number of vertices residing on a timeslice is only a proxy for its (geometric) spatial extent and that there is some uncertainty added to the upcoming procedure by demanding the biggest dual timeslice t_{max} to correspond to the biggest regular timeslice τ_{max}.

simplices inside this sphere and the geodesic distance $d(s_j, s_k)$ is evaluated on the reduced graph described above. Using this observable and the prescription given above, the raw data for building the correlation functions in section 4 were extracted from the simulation data in the following way: For a sample of configurations taken from a simulation run, first all regular timeslices were sorted starting from the largest in terms of vertex content and ascending cyclically in the direction of proper time. Then, for each dual timeslice, a sample of simplex pairs was drawn from its population. For each pair, the spatial distance was determined and the value of eq. (24) was calculated for each simplex and a range of radii δ. The main function to compute and store the acquired data is found as `timesliced_bulk_corr` in appendix A.

3.3.1. Extracting the Ricci scalar

Looking back at the definition of the quantum Ricci scalar arising from eq. (13), it is our working assumption that the average normalized sphere distance $\bar{d}/\delta$ should provide a viable proxy for the full quantum Ricci scalar $K_q(p, p')$, but it is also clear that this methodology leaves room to improvement. To validate our assumption as well as to allow for a more precise interpretation of results, one attempt that has been made in this work is to not only rely on $\bar{d}/\delta$ to build correlation functions, but to extract the quantum Ricci scalar itself. To some extent, this can be done by making use of the series expansion encountered in eq. (9) and eq. (14). A simplified scalar quantity related to the full quantum Ricci curvature at point x (either on the regular or the dual lattice), which we will just call R and, in abuse of language, refer to as *Ricci scalar* in the following, can then be obtained by fitting the average normalized sphere distance at that point for different values of δ to the quadratic expression:

$$\frac{\bar{d}}{\delta}(s_i) = c \cdot (1 - R \cdot \delta^2) \tag{25}$$

This has been done to provide additional correlators based on R which are expected to more truly reflect the local curvature properties aimed at. The fitted range was set to $\delta \in \{6, ..., 10\}$ in order to avoid lattice artifact effects described in [26, 27].

3.3.2. Investigated Correlators

Now that the methods used for generating simulation data as well as calculating geodesics, average normalized sphere distances and Ricci scalars are in place, we can tend towards defining the correlation functions that will be investigated in the next section.

The object in eq. (19) is one of the starting points for our main results. In our framework, where correlators are evaluated on fixed dual timeslices, it depends on the spatial distance d between simplices and of course the dual time t. To account for these primary dependencies, it will be abbreviated as $\mathcal{G}_{\delta\delta}(d)|_t$ in the following. Further correlators are built mostly in the same way, using either $\bar{d}/\delta$, R or the identity operator 1 as a basis.

A list of definitions of investigated correlators is given below, where script letters denote normalized correlators:

$$G_{11}(d)|_t = G[d,t] = \langle G_T[1,1](d)\rangle|_t \tag{26}$$
$$G_\delta|_t = G[\bar{d}/\delta,t] = \langle \bar{d}/\delta\rangle|_t \tag{27}$$
$$G_R|_t = G[R,t] = \langle R\rangle|_t \tag{28}$$
$$G_{\delta\delta}(d)|_t = G[\bar{d}/\delta,\bar{d}/\delta,d,t] = \langle G_T[\bar{d}/\delta,\bar{d}/\delta](d)\rangle|_t \tag{29}$$
$$G_{RR}(d)|_t = G[R,R,d,t] = \langle G_T[R,R](d)\rangle|_t \tag{30}$$
$$\mathcal{G}_{\delta\delta}(d)|_t = \mathcal{G}[\bar{d}/\delta,\bar{d}/\delta,d,t] = \left.\frac{\langle G_T[\bar{d}/\delta,\bar{d}/\delta](d)\rangle}{\langle G_T[1,1](d)\rangle}\right|_t \tag{31}$$
$$\mathcal{G}_{\delta^2\delta^2}(d)|_t = \mathcal{G}[(\bar{d}/\delta)^2,(\bar{d}/\delta)^2,d,t] = \left.\frac{\langle G_T[(\bar{d}/\delta)^2,(\bar{d}/\delta)^2](d)\rangle}{\langle G_T[1,1](d)\rangle}\right|_t \tag{32}$$
$$\mathcal{G}_{RR}(d)|_t = \mathcal{G}[R,R,d,t] = \left.\frac{\langle G_T[R,R](d)\rangle}{\langle G_T[1,1](d)\rangle}\right|_t \tag{33}$$
$$\mathcal{G}_{R^2R^2}(d)|_t = \mathcal{G}[R^2,R^2,d,t] = \left.\frac{\langle G_T[R^2,R^2](d)\rangle}{\langle G_T[1,1](d)\rangle}\right|_t \tag{34}$$
$$\mathcal{G}_{\delta 1}(d)|_t = \mathcal{G}[\bar{d}/\delta,1,d,t] = \left.\frac{\langle G_T[\bar{d}/\delta,1](d)\rangle}{\langle G_T[1,1](d)\rangle}\right|_t \tag{35}$$
$$\mathcal{G}_{1\delta}(d)|_t = \mathcal{G}[1,\bar{d}/\delta,d,t] = \left.\frac{\langle G_T[1,\bar{d}/\delta](d)\rangle}{\langle G_T[1,1](d)\rangle}\right|_t \tag{36}$$
$$\mathcal{G}_{R1}(d)|_t = \mathcal{G}[R,1,d,t] = \left.\frac{\langle G_T[R,1](d)\rangle}{\langle G_T[1,1](d)\rangle}\right|_t \tag{37}$$
$$\mathcal{G}_{1R}(d)|_t = \mathcal{G}[1,R,d,t] = \left.\frac{\langle G_T[1,R](d)\rangle}{\langle G_T[1,1](d)\rangle}\right|_t \tag{38}$$

From these 1-point and 2-point correlation functions, connected correlators can be built readily. Following [26], one possibility to do so consists of defining the full connected curvature correlators as:

$$\mathcal{G}^c_{\delta\delta}(d)|_t = \mathcal{G}^c[\bar{d}/\delta,\bar{d}/\delta,d,t] = \mathcal{G}_{\delta\delta}(d)|_t - 2\cdot G_\delta|_t\cdot\mathcal{G}_{\delta 1}(d)|_t + G^2_\delta|_t \tag{39}$$
$$\mathcal{G}^c_{RR}(d)|_t = \mathcal{G}^c[R,R,d,t] = \mathcal{G}_{RR}(d)|_t - 2\cdot G_R|_t\cdot\mathcal{G}_{R1}(d)|_t + G^2_R|_t \tag{40}$$

In this work, we examined a slightly different object, which acts as a stochastic estimator for the aforementioned functions. This we called the directly subtracted correlator, it is based on the fluctuation operator $\Delta\mathcal{O} = \mathcal{O} - \bar{\mathcal{O}}$ and defined as follows:

$$\mathcal{G}^{\text{sub}}_{\delta\delta}(d)|_t = \frac{\langle (G_{\delta 1}(d)|_t - G_\delta|_t)(G_{1\delta}(d)|_t - G_\delta|_t)\rangle}{G_{11}(d)|_t} \tag{41}$$
$$\mathcal{G}^{\text{sub}}_{RR}(d)|_t = \frac{\langle (G_{R1}(d)|_t - G_R|_t)(G_{1R}(d)|_t - G_R|_t)\rangle}{G_{11}(d)|_t} \tag{42}$$

Correlators based on the squared operators are defined analogously.

4. Results

The following section presents the results obtained from analysis conducted on 4d CDT geometries with respect to the previously defined observables. The configuration files were taken from three Monte Carlo runs, each starting with an identical set of parameters except for the simplicial volume $N_4^{(4,1)}$ (also called V in the following). Especially the asymmetry parameter Δ and the coupling parameter κ_0 were kept fixed at ($\Delta = 0.6, \kappa_0 = 2.2$) in order to produce geometries from the de Sitter phase of CDT phase space.[18] After a general thermalization stage lasting 50.000 Monte Carlo sweeps for all three simulation runs, configurations were saved at different intervals as Monte Carlo snapshots for the three volumes:

$$\begin{aligned} V &= 80.000 = 80k, \quad \tau \in \{1, ..., 60\} \\ V &= 160.000 = 160k, \quad \tau \in \{1, ..., 80\} \\ V &= 320.000 = 320k, \quad \tau \in \{1, ..., 80\} \end{aligned}$$

Here, the temporal extension is given as the range of regular timeslices τ, while the spatial topology for the initial starting geometry has been chosen as that of a sphere for all three parameter sets.

From these data files, samples have been picked to conduct measurements regarding the average normalized sphere distance described earlier. Since this step has been especially computationally intensive, two different samples were collected for each volume: A bigger one with the number of individual triangulations $N_T = 5000$, where measurements at each simplex were taken only for a single curvature sphere radius $\delta = 6$, and a smaller sample with $N_T = 500$, where the average normalized sphere distance has been measured at each simplex for a range of radii $\delta \in \{1, ..., 10\}$. These latter samples were also the ones used for the extraction of the Ricci scalar. The radius $\delta = 6$ has been chosen for the larger datasets since it was expected to provide stable results in a regime where lattice artifacts can be neglected, while we expected the arbitrary choice of a specific δ from this regime to have a negligible impact on measured operator values [8, 19, 26]. This assumption has been tested to hold for the purpose of this study. If not stated otherwise, measurements have been conducted on samples of $n = 100$ simplex pairs on each timeslice, except for the volume regime $V = 80k$, where $n = 50$ was used.

4.1. Preliminary Observations and Remarks

To get acquainted with the measurement output, some preliminary remarks can be made based on the plots shown in fig. 8 and fig. 9. Here, fig. 8 shows, for each simplicial volume sample of 5000 universes, the number of measurement entries per timeslice and spatial distance. This gives a first impression about the shape and spatial extent of the geometries: In all volumes, short distances account for the largest part of measurements,

[18] Further information on the phase space structure of CDT can be found e.g. in [8].

especially in the intermediate dual timeslice region, which corresponds to the stalk of the universe, where we would not expect different behaviour. It is only in the bulk region – around the largest and smallest dual timeslice –, where larger spatial separations between randomly picked simplices become prevalent. This contrast seems to diminish for the larger volume ensemble with $V = 320k$, while the general statement still holds true.

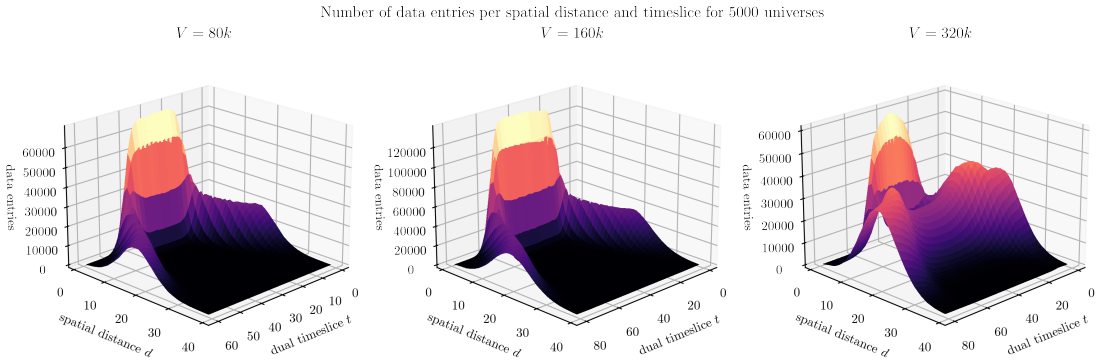

Figure 8: Measurements per spatial distance and dual timeslice for all three ensembles

In fig. 9, the mean value for the average normalized sphere distance is plotted alongside the cube root of the mean number of simplices on the respective dual timeslice. Both quantities again hint at the underlying spherical shape of the geometries under investigation.

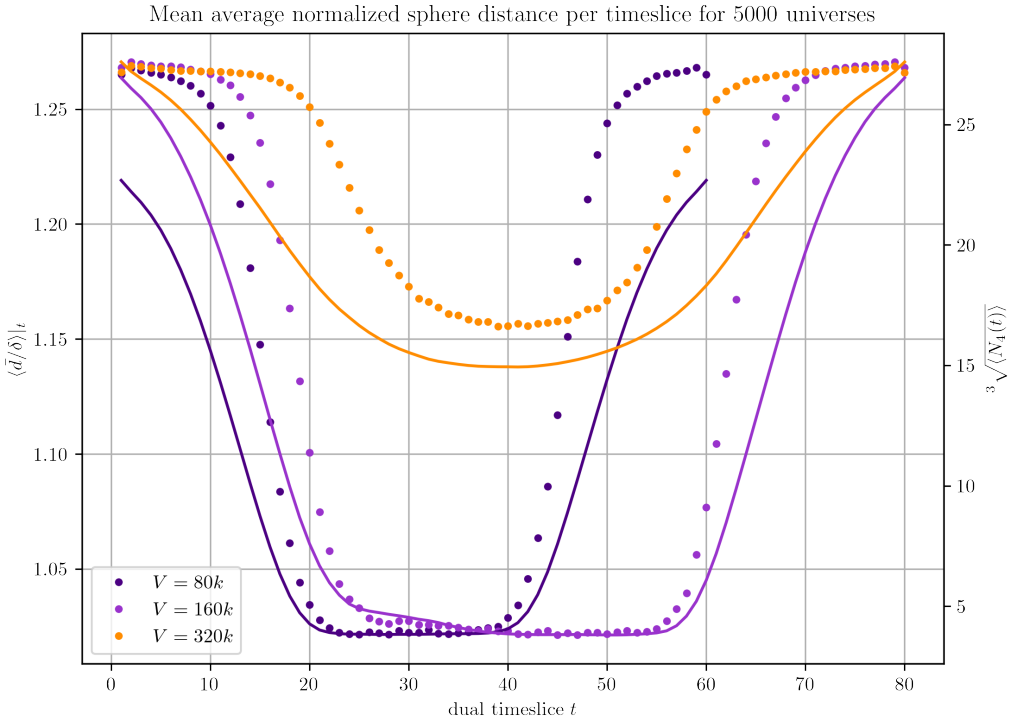

Figure 9: $G_\delta|_t = \langle \bar{d}/\delta \rangle|_t$ (dotted) and $\sqrt[3]{\langle N_4(t) \rangle}$ (solid) for all three ensembles

4.2. Curvature Correlators from $\bar{d}/\delta$

Starting with this section, results are shown for the curvature correlators described in section 3.3. All correlators are given based on the average normalized sphere distance $\bar{d}/\delta$ as well as the Ricci scalar R, for which results are given in the upcoming section.

4.2.1. 2-point correlators from $\bar{d}/\delta$

In fig. 10, results are shown for evaluating the 2-point correlation functions $\mathcal{G}_{\delta\delta}(d)|_t$ and $\mathcal{G}_{\delta^2\delta^2}(d)|_t$ on multiple volume ensembles, each with a sample size of $N_T = 5000$ triangulations, along with the respective unnormalized versions. The spatial distance d is varied as shown on the horizontal axis, while the evaluation was performed on the fixed dual timeslice $t = 1$, corresponding to the largest dual timeslice on each configuration. The varying behavior of $\mathcal{G}_{\delta\delta}(d)|_t$ for different timeslices is depicted in fig. 11.

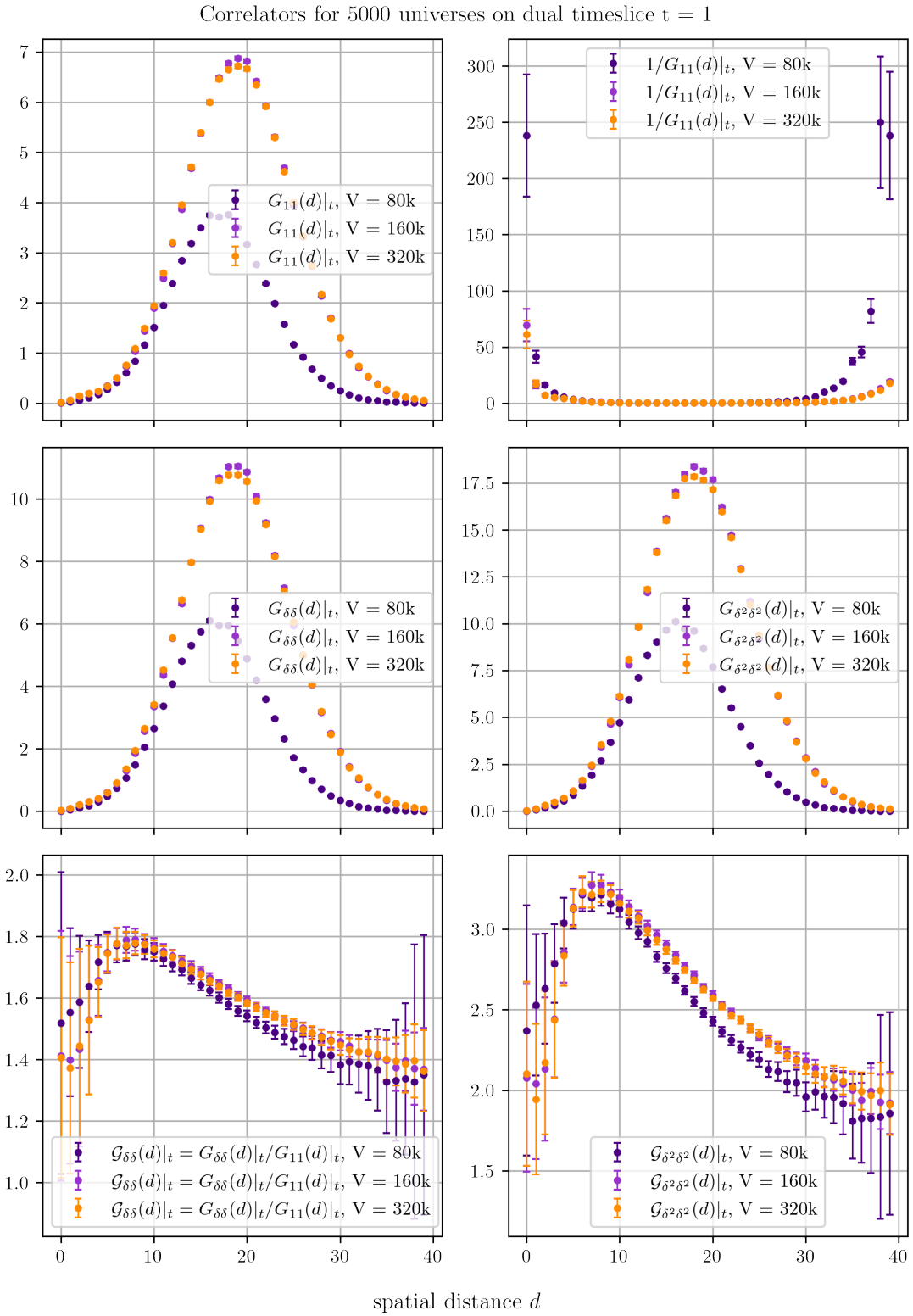

Figure 10: 2-point correlators based on $\bar{d}/\delta$

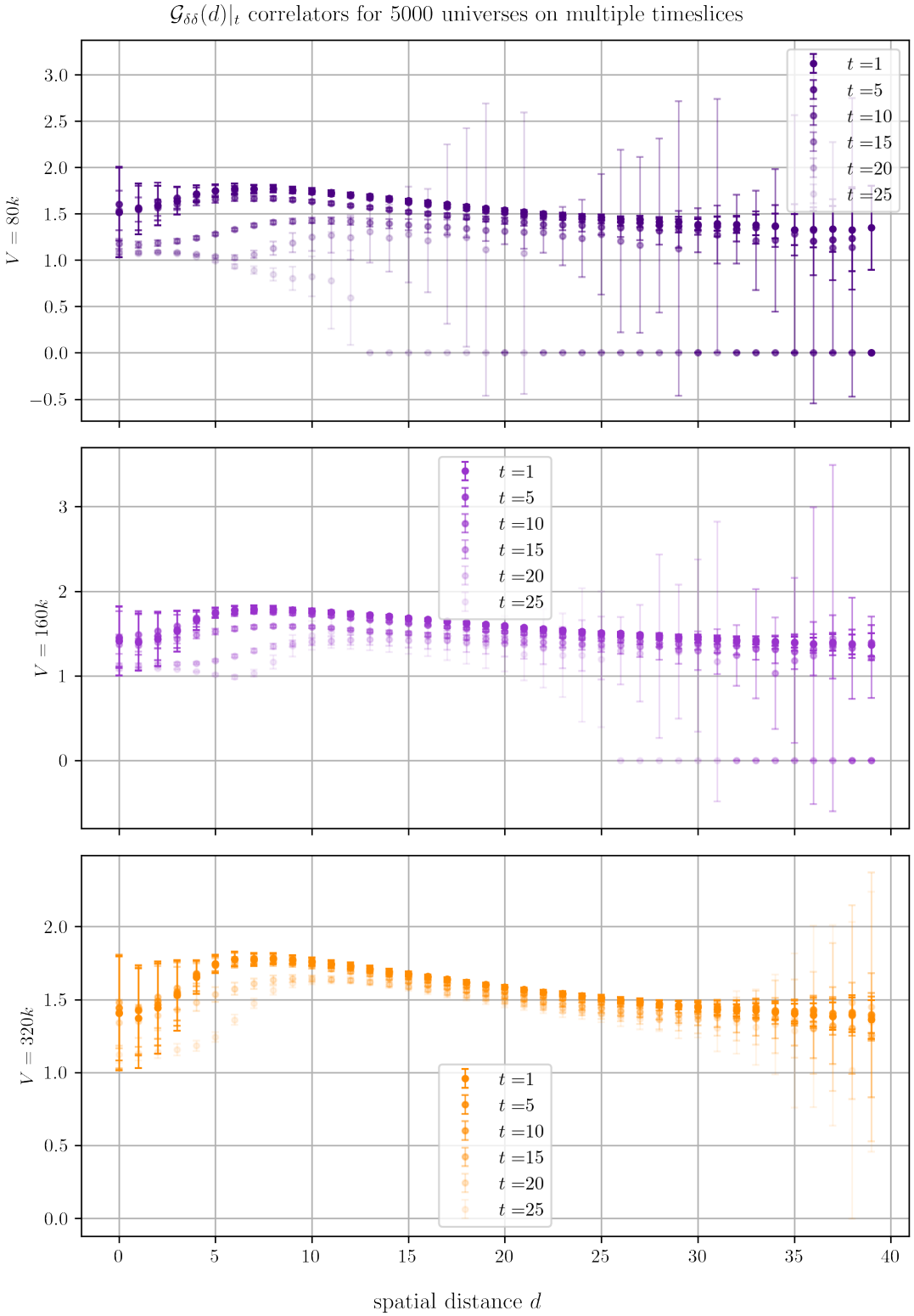

Figure 11: $\mathcal{G}_{\delta\delta}(d)|_t$ for multiple dual timeslices

4.2.2. Directly subtracted 2-point correlators from $\bar{d}/\delta$

In fig. 12 and fig. 13, results are given for evaluating the connected 2-point correlation functions $\mathcal{G}^{\text{sub}}_{\delta\delta}(d)|_t$ and $\mathcal{G}^{\text{sub}}_{\delta^2\delta^2}(d)|_t$ defined in eq. (41) on multiple volume ensembles and with the same sample size as above, $N_T = 5000$. The spatial distance d is again found on the horizontal axis, while evaluation still happened on the fixed dual timeslice $t = 1$. Each subplot corresponds to a fixed volume ensemble and was fitted against an exponential decay function

$$y = a + b \cdot e^{-m \cdot x} \tag{43}$$

where the fit parameters are given in the legends.

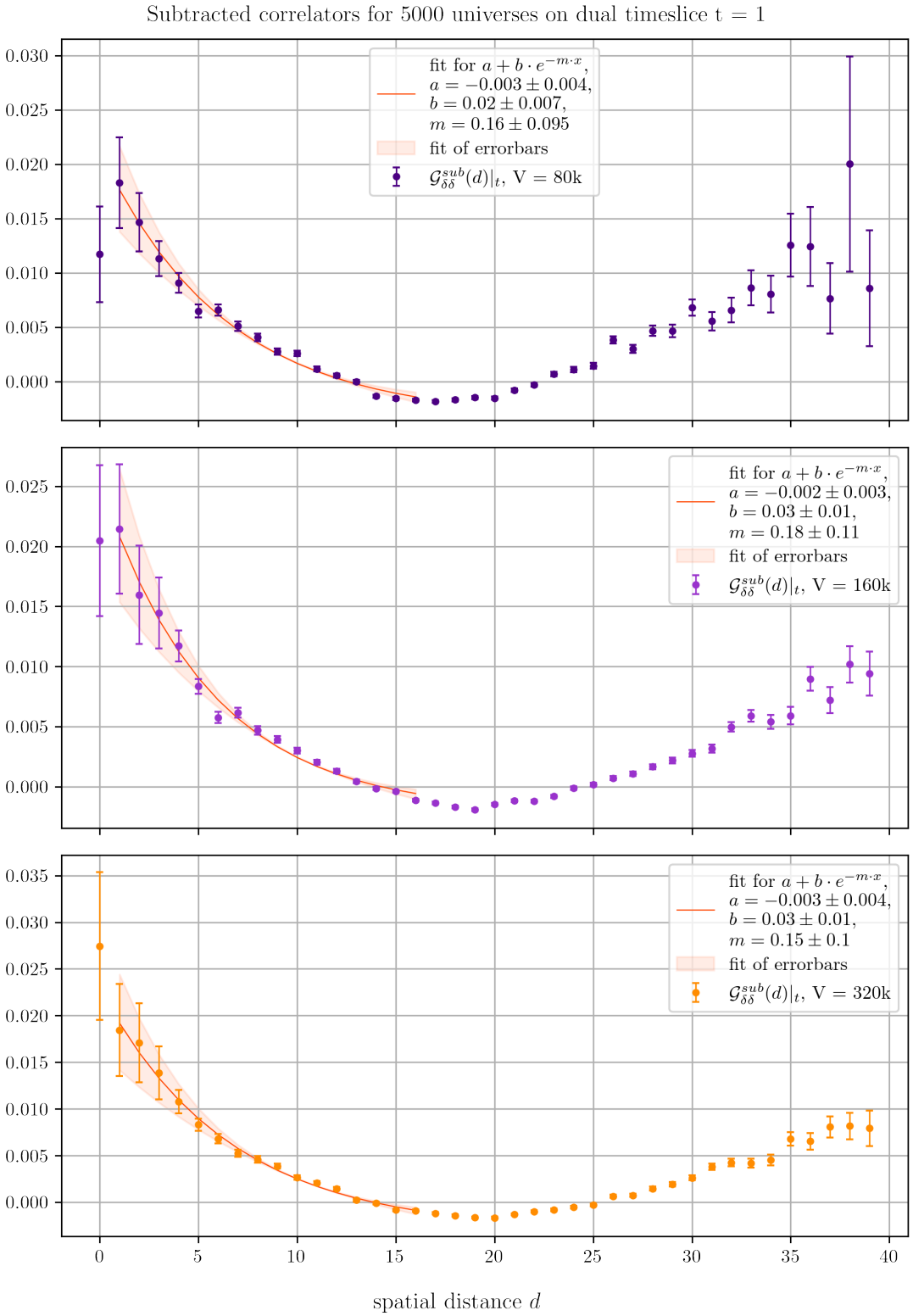

Figure 12: Directly subtracted correlators based on $\bar{d}/\delta$, evaluated on $t = 1$

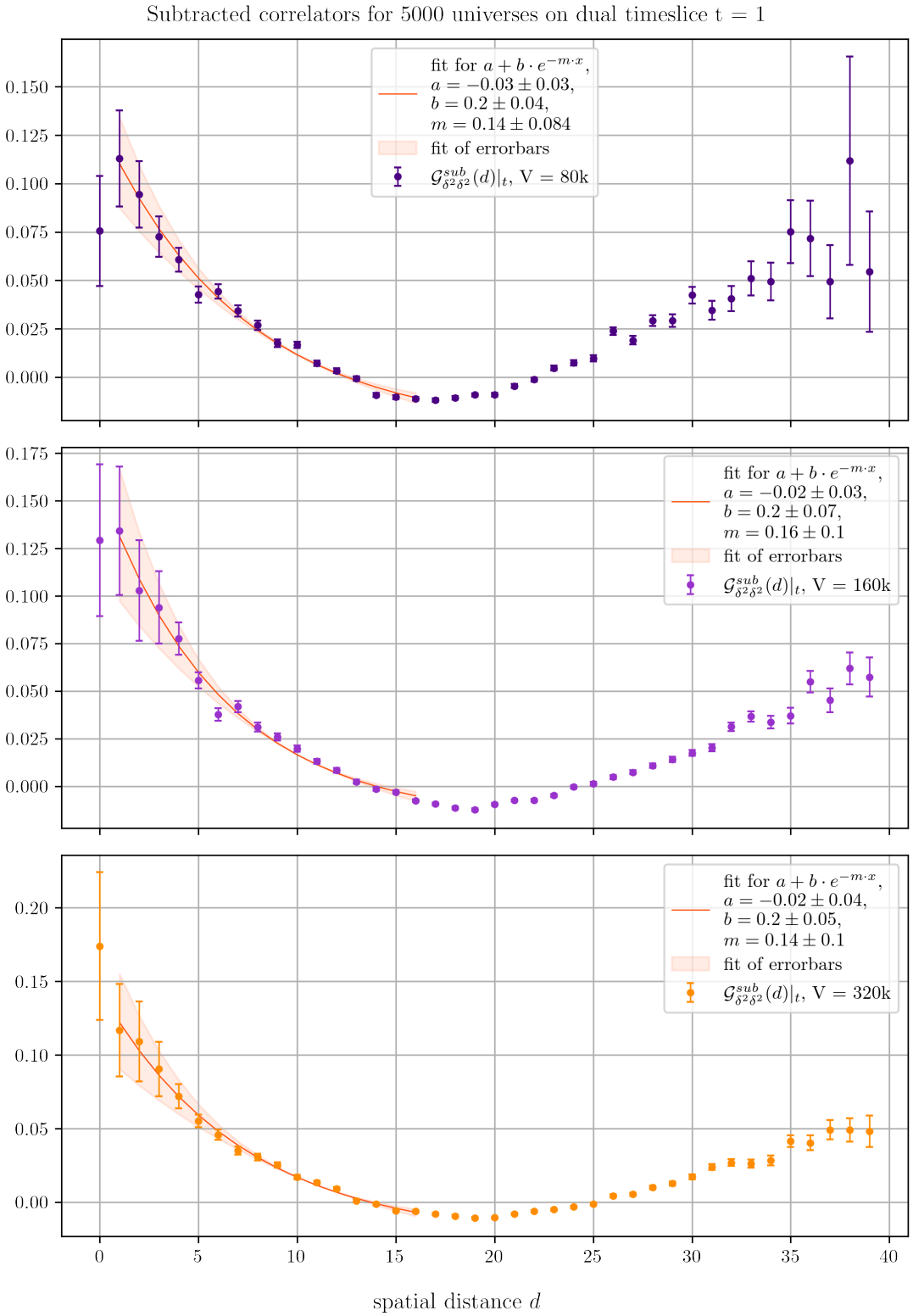

Figure 13: Directly subtracted correlators based on $(\bar{d}/\delta)^2$, evaluated on $t = 1$

4.3. Curvature Correlators from R

4.3.1. 2-point correlators from R

As in section 4.2.1, results for evaluating the 2-point correlation functions $\mathcal{G}_{RR}(d)|_t$ and $\mathcal{G}_{R^2R^2}(d)|_t$ on multiple volume ensembles, this time with a sample size of $N_T = 500$ triangulations each are shown in fig. 14. Evaluation followed the same scheme as for 2-point-correlators based on $\bar{d}/\delta$.

4.3.2. Directly subtracted 2-point correlators from R

Likewise, the directly subtracted 2-point correlators based on R can be computed readily from the definition given in eq. (42). Different to section 4.2.2, however, evaluation of these correlators was limited to the smaller triangulation sample of size $N_T = 500$, accounting for higher statistical uncertainties. Results are given in fig. 15 and fig. 16 for all three volume ensembles, again accompanied by an exponential fit whose parameters are given in the subplot legends.

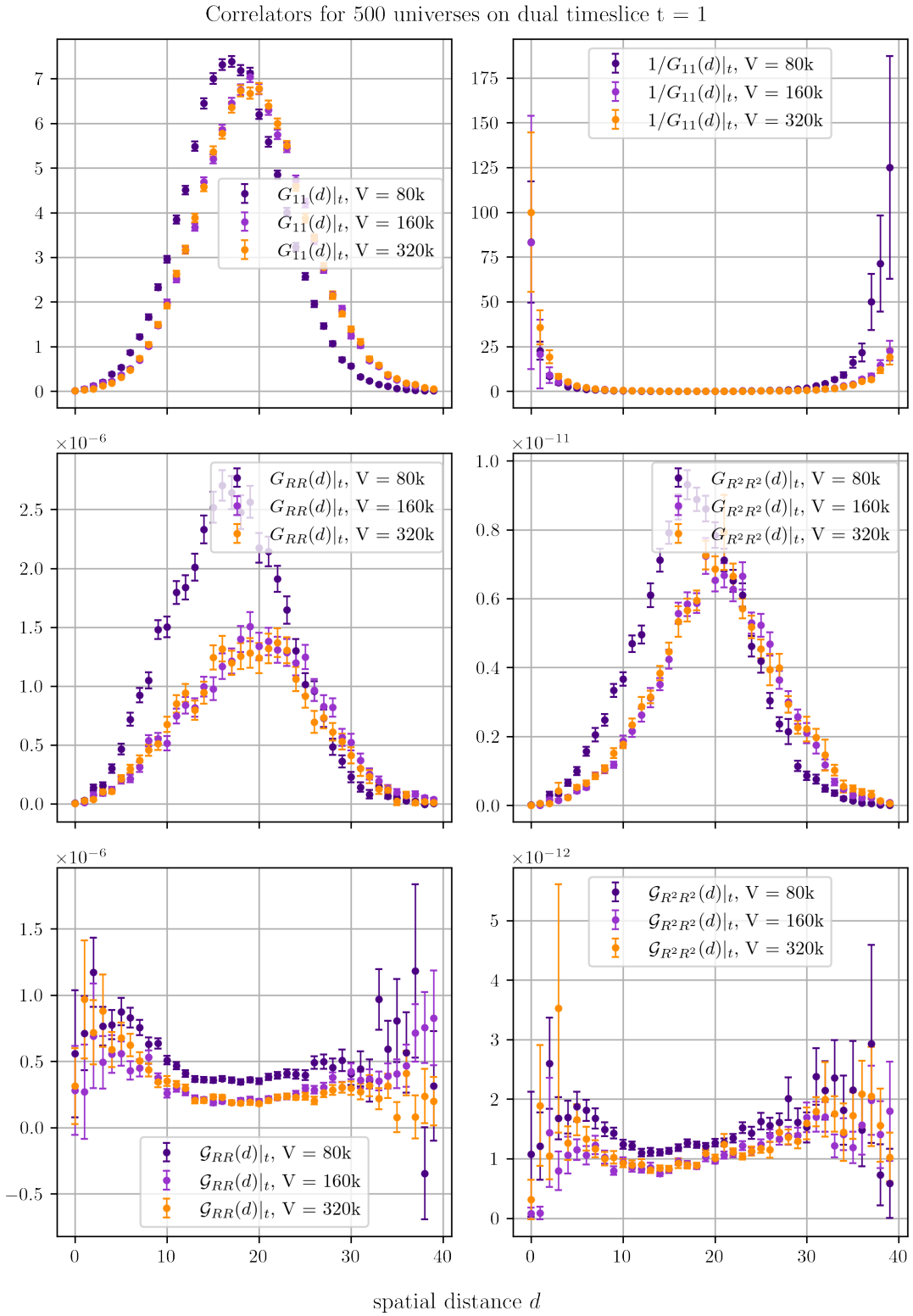

Figure 14: 2-point correlators based on R

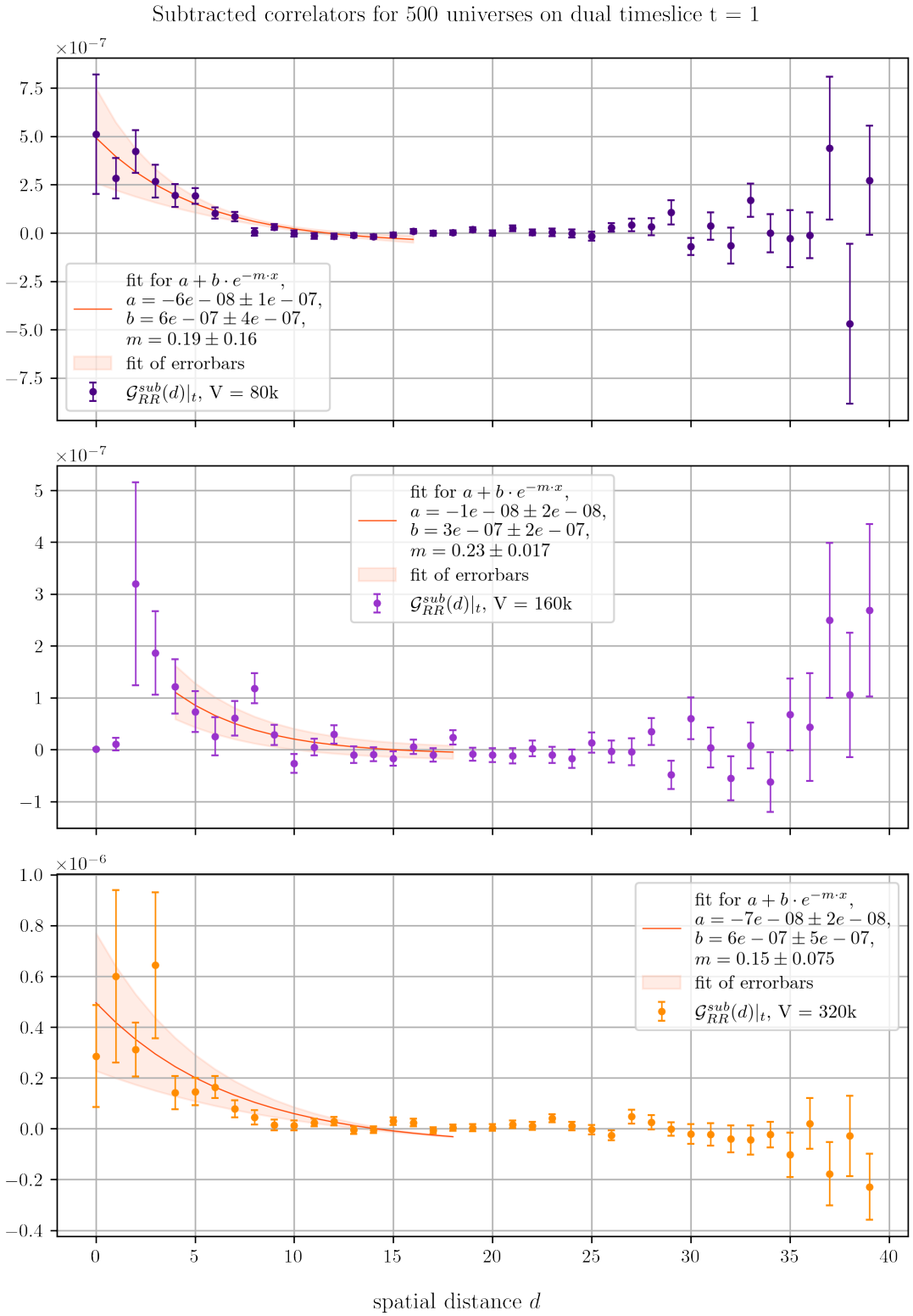

Figure 15: Directly subtracted correlators based on R, evaluated on $t = 1$

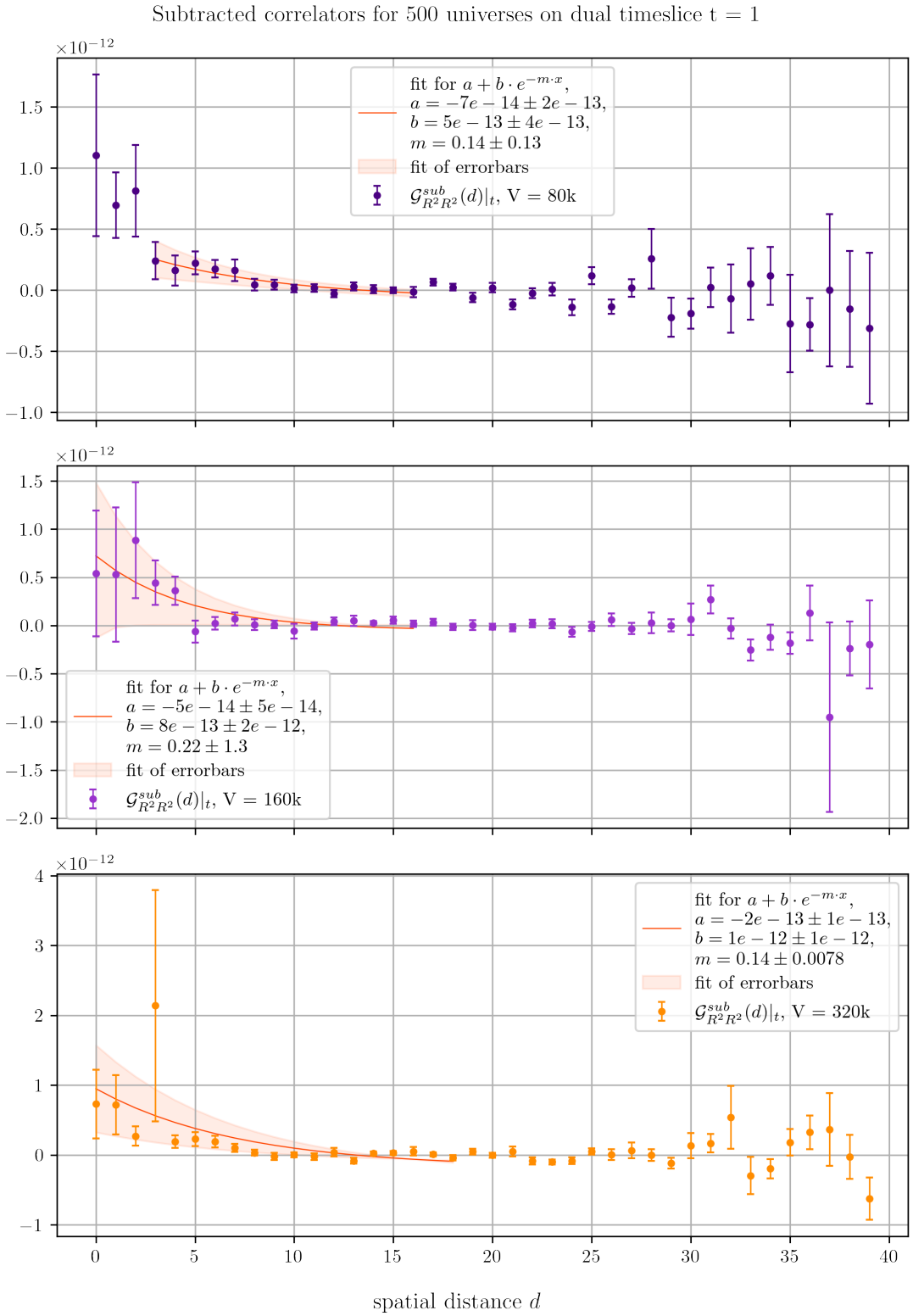

Figure 16: Directly subtracted correlators based on R^2, evaluated on $t = 1$

5. Discussion and Interpretation

The results shown in the previous section allow for a wide range of interpretations. The first conclusion to which one can arrive is that the amount of simulation data used is sufficient to make meaningful statements about the investigated correlators, especially for the larger datasets with $N_T = 5000$. This can be seen not only through the relation between mean measurement values and errorbars, but also through the consistent behavior of correlation functions across the different volume ranges.

The unsubtracted, normalized 2-point correlation functions based on $\bar{d}/\delta$ (depicted in fig. 10) exhibit a clear structure mostly independent of the volume sample used, where slight deviations can be easily attributed to differences in the spatial extent of the triangulations under consideration. While we did not perform a full analysis to compare our results against eq. (14), it is evident that the obtained values lie in the same range as suggested by the analytic approximation as well as numerical results provided in [19, 26].

The values of $\mathcal{G}_{\delta\delta}(d)|_t$ for multiple dual timeslices are shown in fig. 11. Across all volume domains, it can be seen that the shape of the correlator changes with the timeslice in at least two ways: On the one hand, statistical fluctuations start to shift towards shorter spatial separations for higher values of t, which is not surprising since the spatial extent of those slices tends to decrease rapidly, making it increasingly unlikely for large distance measurements to appear in our statistical sample. Another interesting property is that one can see a qualitatively different behavior for timeslices in the intermediate region between the bulk and the stalk, around $t \in \{15...25\}$. These regions can be associated with inflationary stages in the timelike evolution of the universes, where the change in spatial extent is especially large. This has an interesting consequence which will be adressed briefly in the following section.

The subtracted correlators based on $\bar{d}/\delta$ and $(\bar{d}/\delta)^2$ given in fig. 12 and fig. 13 were found to behave roughly in the same ways, regardless of the volume sample considered, which can be considered a hint towards a viable interpretation in terms of particle-like behavior. The correlation functions show a characteristic exponential decay, which ends in a negative minimum value[19] before increasing again and eventually succumbing to statistical noise towards the maximum spatial extent of the timeslice considered. For long distances, it seems that the behavior is not explained by an exponential function but is of rather linear character and it is an open question why that is the case.[20]

Moving on to the correlation functions built from the "Ricci scalar" R and starting with the unsubtracted 2-point correlators from fig. 14, it can be clearly seen that statistical errors are much higher for these results, since the sample size is decreased by a factor of ten. However, the general remarks made earlier for the $\bar{d}/\delta$-correlators regarding

[19] Note that this is not prohibited in curved geometries as opposed to flat-space.

[20] It was verified that especially the short distance behavior of the directly subtracted correlators matches, within errors, the one shown by the full connected correlators from eq. (39), while a full analysis including a direct comparison between the two remained beyond the scope of this work.

universality still seem to apply here. The subtracted correlators built from R and R^2 still exhibit exponential behavior for small spatial distances, while the quality of the functional fits is of course heavily restricted by the lower sample size.

The most important observation regarding all subtracted correlators considered is that the exponential decay is characterized by (within errors) agreeing positive mass values m, as can be seen from table 1, where the values for all volume ensembles and operators are summarized. While the consistency with respect to volume is a clear sign of universal behavior, as was already discussed above, it should not be overlooked that mass values also remain compatible between different basis operators. Since $\bar{d}/\delta$ and $(\bar{d}/\delta)^2$ share the same quantum numbers (as well as R and R^2 respectively), an invariant scalar in the asymptotic regime should appear in the same way in the correlation functions $\mathcal{G}_{\mathcal{O}\mathcal{O}}(d)|_t$ and $\mathcal{G}_{\mathcal{O}^2\mathcal{O}^\in}(d)|_t$. Together, these facts indicate that our results are consistent with the behavior one would expect from a massive particle and thus an interpretation in terms of a Geon appears justified.

V	$\bar{d}/\delta$	$(\bar{d}/\delta)^2$	R	R^2
80k	0.16(1)	0.14(9)	0.19(16)	0.14(13)
160k	0.18(11)	0.16(10)	0.23(2)	0.22(1.3)
320k	0.15(10)	0.14(10)	0.15(8)	0.14(1)

Table 1: Mass values m for different correlators

Taking this conclusion at face value allows for some far reaching speculations: The Geon mass, taken from the largest volume ensemble, roughly lies in the range of

$$m \approx 0.15(10) \tag{44}$$

in natural lattice units, i.e. $[m] = [1/a]$. Using the scaling relations found in [39, 40], it can be estimated that the CDT universes under consideration have a lattice spacing of $a \approx l_{pl}/0.48$ where l_{pl} is the Planck length. From this we can infer that our mass parameter m corresponds to a physical mass m_{ph} on the order of

$$m_{ph} \approx 0.15(10) \cdot \frac{0.48 \cdot \hbar \cdot c}{l_{pl}} \approx 9(6) \times 10^{17}\,\mathrm{GeV} \sim 10^{-9}\,\mathrm{kg} \tag{45}$$

which is about a tenth of the order of the Planck mass. While certainly representing the heavier side of candidate theories, this lies well within astrophysical boundaries imposed on dark matter mass ranges [2, 41].

6. Conclusion and Outlook

The aim of this thesis has been to form a connection between quantum gravity studies and the problem of dark matter by looking for a candidate particle in 4d Causal Dynamical Triangulations. For this, we started by recreating known results from previous investigations regarding the de Sitter phase of CDT to get acquainted with the intricate workings of the theory at hand. Having generated enough simulation data to work on, a specific setup was chosen to measure the curvature properties of the simulated geometries as expressed through the quantum Ricci scalar, specifically through the operators $\bar{d}/\delta$ and R defined in section 2.2.2 and section 3.3.1. To this end, we measured the value of $\bar{d}/\delta$ for pairs of simplices on fixed timeslices of the dual lattice with varying spatial distance and - for the purpose of extracting a proxy Ricci scalar R - varying curvature sphere radius δ.

Building (subtracted) 2-point functions from these data, we found a universal behavior of those correlators, expressed through an exponential decay for short distances, across different volume ensembles and for varying basis operators. Fitting these exponential decays consistently leads to values for the screening mass of $m \approx 0.15(10)$, regardless of operator choice or volume size. This hints towards a justifiable interpretation in terms of a massive particle, essentially describing a Geon. Speculating on further implications, these findings might give rise to a viable candidate for dark matter theories with a physical mass on the order of $10^{-9}\,\mathrm{kg}$.

Since many conceptual and systematic details remain to be discussed, a lot of further investigations will be needed to clarify our exploratory findings and speculative interpretation. A natural first extension of this work would be to examine correlation functions with temporal rather than spatial resolution, as mentioned in eq. (22). Since these objects are qualitatively different from a geometric point of view, they should allow for interesting new features. Another hint for deeper insights comes from the different behavior expressed by the unsubtracted correlators in the inflationary proper time range adressed earlier. Tentative results for extracting the subtracted correlator mass in these ranges show that while its value is stable for timeslices with relatively constant spatial extent, it consistently spikes upwards in the range corresponding to inflationary episodes. While these investigations remained beyond the scope of this thesis and are subject to future publication, pursuing them further might lead to valuable new hypotheses previously unexplored in the area of quantum cosmology.

In summary, it seems that the curvature correlators we investigated in 4d CDT exhibit behavior that hints towards a massive particle. Opening up many fundamental questions and allowing for far-reaching speculations, following this approach should make way for exciting new possibilities.

A. Basic Algorithms and Code

```
/*
Felix Pressler, 2025

Code for reading & analyzing CDT config files and calculating average normalized sphere distance values from them.
To use methods for a series of config files, call functions from UTILITY section.

Build with:
g++ main.cpp -o main.x -std=c++17 -O3

Contains following sections:
- READING
- DATA TYPES
- ANALYSIS
- CORRELATOR
- UTILITY
- MAIN

Routines called from UTILITY need a path to a directory containing conf files and a file index passed as command line arguments to run.
If code from ANALYSIS or CORRELATOR is to be tested on single configs, a Universe needs to be initialized first with
>>> currentU.initialize("path/to/conf.dat")
in the main function.
*/

#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>
#include <cstdint>
#include <iterator>
#include <algorithm>
#include <utility>
#include <queue>
#include <unordered_map>
#include <unordered_set>
#include <optional>

#include <limits>
```

```
38 #include <stack>
39 #include <chrono>
40 #include <random>
41 #include <string>
42 #include <filesystem>
43 #include <sstream>
44 #include <iomanip>
45 #include <ctime>
46
47 uint32_t fileprecision = 19;
48
49 /*
50 READING SECTION
51 basic functions for reading and processing config files
52 */
53 // Reads in data file
54 std::pair<std::vector<uint32_t>, std::vector<uint32_t>> Read_config(const
   ↪  std::string& filename) {
55     std::vector<std::vector<uint8_t>> data;
56     std::ifstream file(filename.c_str(), std::ios::binary);
57
58     if (!file.is_open()) {
59         std::cerr << "Error opening file!" << std::endl;
60         return {};
61     }
62
63     auto read_bytes = [&](size_t size) -> std::vector<uint8_t> {
64         std::vector<uint8_t> bytes(size);
65         if (file.read(reinterpret_cast<char*>(bytes.data()), size)) {
66             return bytes;
67         }
68         return {};
69     };
70
71     auto bytes_to_int = [](const std::vector<uint8_t>& bytes) -> uint32_t
      ↪  {
72         if (bytes.size() < 4) {
73             std::cerr << "Invalid byte size for conversion to int." <<
                ↪  std::endl;
74             return 0;
75         }
76         return *reinterpret_cast<const uint32_t*>(bytes.data());
77     };
```

```
78
79      auto add_read_bytes = [&](size_t size) {
80          std::vector<uint8_t> bytes = read_bytes(size);
81          if (bytes.empty()) {
82              std::cerr << "Error reading bytes from file." << std::endl;
83              return false;
84          }
85          data.push_back(bytes);
86          return true;
87      };
88
89      if (!add_read_bytes(4)) return {};
90      if (!add_read_bytes(4)) return {};
91      uint32_t num4 = bytes_to_int(data.back());
92
93      if (!add_read_bytes(4)) return {};
94      uint32_t num0 = bytes_to_int(data.back());
95
96      if (!add_read_bytes(4)) return {};
97      uint32_t num50 = bytes_to_int(data.back());
98
99      if (!add_read_bytes(4)) return {};
100     if (!add_read_bytes(4)) return {};
101
102     for (uint32_t i = 0; i < num0; ++i) {
103         if (!add_read_bytes(4)) return {};
104     }
105
106     if (!add_read_bytes(4)) return {};
107     if (!add_read_bytes(4)) return {};
108
109     for (uint32_t i = 0; i < 10 * num4; ++i) {
110         if (!add_read_bytes(4)) return {};
111     }
112
113     if (!add_read_bytes(4)) return {};
114
115     std::vector<uint8_t> byte(1);
116     while (file.read(reinterpret_cast<char*>(byte.data()), 1)) {
117         // data.push_back(byte);  // this line in Python code is
            ↪  commented out
118     }
119
```

```
120     std::vector<uint32_t> config_file;
121     for (const auto& item : data) {
122         config_file.push_back(bytes_to_int(item));
123     }
124 
125     if (config_file.size() < 10 || config_file.size() < num0 + 6 ||
        ↪  config_file.size() < 10 * num4 + num0 + 8) {
126         std::cerr << "Insufficient data read from file." << std::endl;
127         return {};
128     }
129 
130     if (config_file[0] == config_file[4]) {
131         std::cout << "Check: OK" << std::endl;
132     } else {
133         std::cout << "Check: Load-Error_1" << std::endl;
134     }
135 
136     if (config_file[5] == config_file[num0 + 5 + 1]) {
137         std::cout << "Check: OK" << std::endl;
138     } else {
139         std::cout << "Check: Load-Error_2" << std::endl;
140     }
141 
142     if (config_file[num0 + 5 + 2] == config_file[num0 + 5 + 2 + 1 + 10 *
        ↪  num4]) {
143         std::cout << "Check: OK" << std::endl;
144     } else {
145         std::cout << "Check: Load-Error_3" << std::endl;
146     }
147 
148     std::vector<uint32_t> list_p(config_file.begin() + 6,
        ↪  config_file.begin() + num0 + 6);
149     std::vector<uint32_t> list_s(config_file.begin() + num0 + 8,
        ↪  config_file.begin() + 10 * num4 + num0 + 8);
150 
151     return std::make_pair(list_p, list_s);
152 }
153 // Creates arrays from Read_config
154 std::pair<std::vector<std::vector<uint32_t>>,
    ↪  std::vector<std::vector<uint32_t>>> process_config(const
    ↪  std::pair<std::vector<uint32_t>, std::vector<uint32_t>>& config) {
155     const std::vector<uint32_t>& list_p = config.first;
156     const std::vector<uint32_t>& list_s = config.second;
```

```
157
158     std::vector<std::vector<uint32_t>> vertices;
159     std::vector<std::vector<uint32_t>> simplices;
160
161     std::vector<uint32_t> zero_start = {{0, 0, 0, 0, 0}};
162     vertices.push_back(zero_start);
163     simplices.push_back(zero_start);
164
165     // Ensure list_s size is a multiple of 10
166     if (list_s.size() % 10 != 0) {
167         std::cerr << "Error: list_s size is not a multiple of 10." <<
            ↪  std::endl;
168         return {};
169     }
170
171     for (size_t i = 0; i < list_s.size(); i += 10) {
172         std::vector<uint32_t> vertex_block(list_s.begin() + i,
            ↪  list_s.begin() + i + 5);
173         std::vector<uint32_t> simplex_block(list_s.begin() + i + 5,
            ↪  list_s.begin() + i + 10);
174         vertices.push_back(vertex_block);
175         simplices.push_back(simplex_block);
176     }
177
178     return {vertices, simplices};
179 }
180 // NOT IN USE ATM: Function to convert simplices into an adjacency list
    ↪  representation
181 std::unordered_map<uint32_t, std::vector<uint32_t>> build_graph(const
    ↪  std::vector<std::vector<uint32_t>>& simplices) {
182     std::unordered_map<uint32_t, std::vector<uint32_t>> graph;
183
184     for (uint32_t i = 0; i < simplices.size(); ++i) {
185         graph[i] = simplices[i];
186     }
187
188     return graph;
189 }
190
191
192
193 /*
194 DATA TYPES
```

```
195 defines Ricci struct where dijkstra() and avg_sphere_dist() work on
196 and Universe which holds on to config data and adjacency lists
197 */
198 // Constant
199 const uint32_t INF = std::numeric_limits<uint32_t>::max();
200
201 // Struct to hold data from average sphere distance
202 struct Ricci {
203     std::vector<uint32_t> distances;
204     std::unordered_map<uint32_t, uint32_t> predecessors;
205     //std::multimap<uint32_t, uint32_t> verts_per_cost; // not in use atm
206     std::vector<uint32_t> sphere;
207     uint32_t pathlen;
208
209     // Initializes the struct with the graph
210     void initialize(const std::vector<std::vector<uint32_t>>& graph) {
211         pathlen = 0;
212         distances.resize(graph.size());
213         std::fill(distances.begin(), distances.end(), INF);
214
215     }
216
217     void reset() {
218         pathlen = 0;
219         std::fill(distances.begin(), distances.end(), INF);
220     }
221
222     // Resets the sphere vector
223     void reset_sphere() {
224         sphere.clear();
225     }
226 };
227
228 Ricci temp_ricci;
229 Ricci ricci;
230
231 // Struct to hold config data and adjacency lists
232 struct Universe {
233     std::vector<std::vector<uint32_t>> simplices;         // Each simplex
      ↪  contains 5 vertices
234     std::vector<uint32_t> vertex_times;                   //
      ↪  Corresponding time for each vertex
```

```
235     uint32_t t_max;                                         // Maximal
        ↪  timeslice
236     std::vector<std::vector<uint32_t>> simplex_neighbors; // Neighbors of
        ↪  each simplex (adjacency list)
237     std::string name;                                       // Name of
        ↪  config
238     std::vector<std::vector<uint32_t>> simplex_times;     // Timeslices
        ↪  of vertices building a simplex
239     std::vector<std::vector<uint32_t>> reduced_simplices; // Reduced
        ↪  simplex_neighbors for each timeslice
240     std::vector<std::vector<uint32_t>> simplex_slices;    // Lists of
        ↪  simplices on each simplex-timeslice
241                                                             // (where slice
                                                                ↪  t contains
                                                                ↪  simplices
                                                                ↪  between t
                                                                ↪  and t+1)
242
243     // Initialize Struct from file and Ricci from struct
244     void initialize(const std::string& filename) {
245         auto config = Read_config(filename);
246         auto result = process_config(config);
247         simplices = result.first;
248         vertex_times = config.first;
249         vertex_times.insert(vertex_times.begin(), 0);
250         simplex_neighbors = result.second;
251         name = std::string_view(filename).substr(0, filename.size()-4);
252
253         t_max = *std::max_element(vertex_times.begin(),
            ↪  vertex_times.end());
254
255         get_simplex_times();
256         reduce_graph();
257         sort_simplex_slice();
258
259         temp_ricci.initialize(simplex_neighbors);
260         ricci.initialize(simplex_neighbors);
261     }
262
263     // Function to get vertex times for each simplex
264     void get_simplex_times() {
265         //std::vector<std::vector<uint32_t>> vert_times(simplices.size(),
            ↪  std::vector<uint32_t>(5, 0));
```

```
266         simplex_times.resize(simplices.size(), std::vector<uint32_t>(5,
            ↪ 0));
267
268         // Iterate over each simplex and its vertices to assign times
269         for (size_t i = 0; i < simplices.size(); ++i) {
270             for (size_t j = 0; j < simplices[i].size(); ++j) {
271                 uint32_t vertex = simplices[i][j]; // Get vertex index
272                 simplex_times[i][j] = vertex_times[vertex]; // Get
                    ↪ corresponding time
273             }
274         }
275     }
276
277
278     // Function to sort simplices into timeslices
279     void sort_simplex_slice() {
280         simplex_slices.resize(t_max+1);
281
282         for (uint32_t i = 1; i < simplex_times.size(); ++i) {
283             if (std::find(simplex_times[i].begin(),
                ↪ simplex_times[i].end(), 1) != simplex_times[i].end()) {
284                 uint32_t maxt =
                    ↪ *std::max_element(simplex_times[i].begin(),
                    ↪ simplex_times[i].end());
285                 if (maxt == 2) {
286                     simplex_slices[1].push_back(i);
287                 }
288                 else {
289                     simplex_slices[t_max].push_back(i);
290                 }
291             }
292             else {
293                 uint32_t t_min =
                    ↪ *std::min_element(simplex_times[i].begin(),
                    ↪ simplex_times[i].end());
294                 simplex_slices.at(t_min).push_back(i);
295             }
296         }
297     }
298
299     // Function to reduce adjacency list to simplices on the same
        ↪ timeslice (spacelike separation)
300     void reduce_graph() {
```

```
301         //reduced_simplices.resize(simplices.size(),
            ↪  std::vector<uint32_t>(5, 0));
302         reduced_simplices = simplex_neighbors;
303
304         for (uint32_t i=1; i < simplex_neighbors.size(); ++i) {
305             const auto& neighbors = simplex_neighbors[i];
306             for (auto& n : neighbors) {
307                 for (const auto& t : simplex_times[n]) {
308                     if (std::find(simplex_times[i].begin(),
                        ↪  simplex_times[i].end(), t) ==
                        ↪  simplex_times[i].end()) {
309                         reduced_simplices[i].erase(
310                         std::remove(reduced_simplices[i].begin(),
                            ↪  reduced_simplices[i].end(), n));
311                     }
312                 }
313             }
314         }
315     }
316
317 };
318
319 Universe currentU;
320
321
322
323 /*
324 ANALYSIS SECTION
325 dijkstra(), avg_sphere_dist() and build_sphere() are the core of the
    ↪  program
326 */
327
328 // Function to implement Dijkstra's algorithm and store results in Ricci
    ↪  struct
329 void dijkstra(Ricci &ricci, const std::vector<std::vector<uint32_t>>&
    ↪  graph, uint32_t start, std::optional<uint32_t> killdist =
    ↪  std::nullopt, std::optional<uint32_t> end = std::nullopt) {
330     //std::unordered_set<uint32_t> visited;
331     ricci.distances[start] = 0;
332
333     using P = std::pair<uint32_t, uint32_t>;
334     std::vector<P> container;
335     //container.reserve(10000);
```

```
336     std::priority_queue<P, std::vector<P>, std::greater<P>>
        ↪  pq;//(std::greater<P>(), std::move(container));
337     pq.emplace(0, start);
338
339     while (!pq.empty()) {
340         uint32_t distance = pq.top().first;
341         uint32_t u = pq.top().second;
342         pq.pop();
343
344         //if (visited.find(u) != visited.end()) continue;
345         //visited.insert(u);
346         if (end && u == end) break;
347
348         if (killdist && distance > killdist) break;
349
350         for (const auto& v : graph[u]) {
351             //if (visited.find(v) != visited.end()) continue;
352             uint32_t new_distance = distance + 1;
353
354             if (new_distance < ricci.distances[v]) {
355                 ricci.distances[v] = new_distance;
356                 ricci.predecessors[v] = u;
357                 //ricci.verts_per_cost.emplace(new_distance, v);
358                 pq.emplace(new_distance, v);
359             }
360         }
361     }
362
363     // If an end vertex is specified, reconstruct the path
364     // Currently only stores the path length as int
365     if (end && ricci.distances[*end] != INF) {
366         uint32_t at = *end;
367         while (at != start) {
368             //ricci.sphere.push_back(at);
369             //std::cout << at << " ";
370             at = ricci.predecessors[at];
371             ricci.pathlen += 1;
372         }
373         //std::cout << std::endl;
374         //ricci.sphere.push_back(start);
375         //std::reverse(ricci.sphere.begin(), ricci.sphere.end()); //
            ↪  Reverses to get the correct path order
376     }
```

```
377
378 }
379
380 // For building a sphere around a vertex, using costs from Dijkstra's
    ↪ algorithm stored in Ricci
381 void build_sphere(Ricci &ricci, uint32_t radius) {
382     ricci.reset_sphere();
383     for (int i=0; i < ricci.distances.size(); i++) if (ricci.distances[i]
    ↪ == radius) ricci.sphere.push_back(i);
384 }
385
386 // Computing the average normalized sphere distance
387 double avg_sphere_dist(uint32_t v, const
    ↪ std::vector<std::vector<uint32_t>>& graph, uint32_t r) {
388     //ricci.initialize(graph);
389     ricci.reset();
390
391     // Run Dijkstra's algorithm from vertex v
392     dijkstra(ricci, graph, v, r);
393
394     // Build the sphere of radius r
395     build_sphere(ricci, r);
396
397     // uint32_t sphere_size = ricci.sphere.size(); // gives no
    ↪ improvement
398
399     if (ricci.sphere.empty()) return 0.0;
400
401     double prefactor = 1.0 / pow(ricci.sphere.size(),
    ↪ 2);//static_cast<double>(ricci.sphere.size() *
    ↪ ricci.sphere.size());
402     double total_distance = 0.0;
403
404     // Iterate over all pairs of vertices in the sphere and sum their
    ↪ distances
405     for (size_t i = 0; i < ricci.sphere.size(); ++i) {
406         uint32_t &v1 = ricci.sphere[i];
407
408         //temp_ricci.initialize(graph);
409         temp_ricci.reset();
410
411         dijkstra(temp_ricci, graph, v1, 2*r+1);
412
```

```
413         for (size_t j = i + 1; j < ricci.sphere.size(); ++j) {
414             uint32_t &v2 = ricci.sphere[j];
415 
416             total_distance +=
                ↪  static_cast<double>(temp_ricci.distances[v2]) /
                ↪  static_cast<double>(r);
417         }
418     }
419 
420     return 2 * prefactor * total_distance; // times 2 because
        ↪  permutations (x,y) -> (y,x) should be counted
421 
422 }
423 
424 
425 
426 /*
427 CORRELATOR SECTION
428 Helper functions and functions for calculating correlator values across
    ↪  multiple config universes
429 */
430 
431 // Helper function to randomly select a simplex
432 uint32_t random_simplex(uint32_t range) {
433     static std::random_device rd;
434     static std::mt19937 gen(rd());
435     std::uniform_int_distribution<uint32_t> dis(1, range);
436     return dis(gen);
437 }
438 
439 // Helper function to find the mode of a vector (for finding the bulk of
    ↪  a universe)
440 uint32_t find_mode(const std::vector<uint32_t>& vec) {
441     std::unordered_map<uint32_t, uint32_t> frequency_map;
442 
443     // Count the frequency of each element in the vector
444     for (uint32_t num : vec) {
445         frequency_map[num]++;
446     }
447 
448     // Find the element with the highest frequency
449     uint32_t mode = vec[0];
450     uint32_t max_count = 0;
```

```
451
452     for (const auto& [value, count] : frequency_map) {
453         if (count > max_count) {
454             max_count = count;
455             mode = value;
456         }
457     }
458
459     return mode;
460 }
461
462
463 // Main correlator function for random simplices
464 std::vector<double> correlator(const std::string& filename, const
    ↪  std::vector<std::vector<uint32_t>>& graph, uint32_t n_points,
    ↪  uint32_t radius, uint32_t distance) {
465     uint32_t n = 0;
466     std::vector<double> corr;
467
468     // Save file name according to Python-style convention
469     std::string savefile = "correlator_out_" + filename + "_n" +
        ↪  std::to_string(n_points) +
470                            "_d" + std::to_string(distance) + "_r" +
                               ↪  std::to_string(radius) + ".txt";
471
472     std::ofstream file(savefile);
473     file.precision(fileprecision);
474     if (!file.is_open()) {
475         std::cerr << "Failed to open file for writing!" << std::endl;
476         return corr;
477     }
478
479     // Write header
480     file << "Universe: " << filename << ", number of points: " <<
        ↪  n_points
481          << ", distance between points: " << distance
482          << ", radius of curvature sphere: " << radius << "\n";
483     file << "si  sj  d/di  d/dj  d/di*d/dj   d/di^2*d/dj^2\n";
484     file.flush();
485
486     // Main loop to compute correlators
487     while (n < n_points) {
488         // Randomly select a simplex s_i
```

```
489         uint32_t s_i = random_simplex(graph.size());
490
491         // Run Dijkstra's algorithm from s_i
492         uint32_t killer = std::max(radius, distance) + 1;
493         dijkstra(ricci, graph, s_i, killer);
494
495         // Find simplices within a sphere around s_i
496         build_sphere(ricci, distance);
497         std::vector<uint32_t> simplex_choices = ricci.sphere;
498         if (simplex_choices.empty()) continue; // If no simplex found,
            ↪  skip this iteration
499
500         // Randomly choose another simplex s_j within the distance sphere
501         uint32_t s_j =
            ↪  simplex_choices[random_simplex(simplex_choices.size()) - 1];
            ↪  // Pick a random simplex from the list
502
503         // Compute avg sphere distances for s_i and s_j
504         double R_i = avg_sphere_dist(s_i, graph, radius);
505         double R_j = avg_sphere_dist(s_j, graph, radius);
506
507         // Compute correlators
508         double correlator_value = R_i * R_j;
509         double corr_2 = std::pow(R_i, 2) * std::pow(R_j, 2);
510         corr.push_back(correlator_value);
511
512         // Write results to file
513         file << s_i << " " << s_j << " " << R_i << " " << R_j << " "
514              << correlator_value << " " << corr_2 << "\n";
515         file.flush();
516
517         n++;
518     }
519
520     std::cout << "Univ = " << filename << ", d = " << distance << ", r =
        ↪  " << radius << ", n = " << n << std::endl;
521     return corr;
522 }
523
524 // Calculates the single-operator correlator in the bulk region
525 void bulk_corr(const std::string& filename, Universe& currentU, uint32_t
    ↪  n_points, uint32_t radius) {
526     uint32_t n = 0;
```

```
527     //auto filename = currentU.name;
528     auto graph = currentU.simplex_neighbors;
529 
530 
531     // Save file name according to Python-style convention
532     std::string savefile = "bulk_correlator_out_" + filename + "_n" +
        ↪  std::to_string(n_points) +
533                            "_r" + std::to_string(radius) + ".txt";
534 
535     std::ofstream file(savefile);
536     file.precision(fileprecision);
537     if (!file.is_open()) {
538         std::cerr << "Failed to open file for writing!" << std::endl;
539     }
540 
541     // New for bulk correlator:
542     uint32_t t_big = find_mode(currentU.vertex_times);
543     uint32_t t_max = *std::max_element(currentU.vertex_times.begin(),
        ↪  currentU.vertex_times.end());
544 
545 
546     // Write header
547     file << "Universe: " << filename << ", number of points: " <<
        ↪  n_points
548          << ", largest timeslice: " << t_big << ", max timeslice: " <<
             ↪  t_max
549          << ", radius of curvature sphere: " << radius << "\n";
550     file << "d/di  (for each slice and a random simplex, starting with
        ↪  largest)\n";
551     file.flush();
552 
553     std::vector<std::vector<double>> value_holder(n_points+2,
        ↪  std::vector<double>(t_max));
554     std::cout << "Universe: " << filename << std::endl;
555     std::cout << "t_big: " << t_big << ", t_max: " << t_max << std::endl;
556 
557     // Main loop to compute correlators
558     for (size_t t = 0; t < t_max; t++) {
559         uint32_t ind = (t_big + t) % t_max;
560         if (ind == 0) ind = t_max;
561         std::cout << ind << std::endl;
562 
563         auto sslice = currentU.simplex_slices[ind];
```

```
564             value_holder[0][t] = ind;
565             value_holder[1][t] = sslice.size();
566 
567             if (sslice.empty()) continue;
568 
569             n = 0;
570             //std::cout << ind << std::endl;
571             while (n < n_points) {
572                 // Randomly select a simplex s_i
573                 uint32_t i_i = random_simplex(sslice.size() - 1);
574                 uint32_t s_i = sslice[i_i];
575 
576                 // Compute avg sphere distances for s_i and s_j
577                 double R_i = avg_sphere_dist(s_i, graph, radius);
578                 value_holder[n+2][t] = R_i;
579 
580                 n++;
581             }
582         }
583 
584         for (const auto& row : value_holder) {
585             for (const auto& entry : row) {
586                 file << entry << " ";
587             }
588             file << std::endl;
589         }
590 
591         std::cout << "Univ = " << filename << ", t_big = " << t_big << ", r =
    ↪  " << radius << ", n = " << n << std::endl;
592     }
593 
594     // Calculates the single-operator correlator for pairs of simplices on
    ↪  each timeslice, starting with the bulk region
595     void timesliced_bulk_corr(const std::string& filename, Universe&
    ↪  currentU, uint32_t n_pairs, uint32_t radius) {
596         uint32_t n = 0;
597         //auto filename = currentU.name;
598         auto& graph = currentU.simplex_neighbors;
599         auto& reduced_graph = currentU.reduced_simplices;
600 
601 
602         // Save file name according to Python-style convention
```

```
603     std::string savefile = "timesliced_bulk_correlator_out_" + filename +
        ↪  "_n" + std::to_string(n_pairs) +
604                            "_r" + std::to_string(radius) + ".txt";
605
606     std::ofstream file(savefile);
607     file.precision(fileprecision);
608     if (!file.is_open()) {
609         std::cerr << "Failed to open file for writing!" << std::endl;
610     }
611
612     // New for bulk correlator:
613     uint32_t t_big = find_mode(currentU.vertex_times);
614
615     // Write header
616     file << "Universe: " << filename << ", number of simplex pairs per
        ↪  timeslice: " << n_pairs
617          << ", largest timeslice: " << t_big << ", max timeslice: " <<
             ↪  currentU.t_max
618          << ", maximal radius of curvature sphere (r_max): " << radius <<
             ↪  "\n";
619     file << "Timeslices with number of associated simplices (simplices
        ↪  are stored on slice of lowest time): \n";
620     std::vector<std::vector<uint32_t>> slice_count;
621     slice_count.resize(2, std::vector<uint32_t>(currentU.t_max, 0));
622     for (uint32_t t = 0; t < currentU.t_max; t++) {
623         uint32_t ind = (t_big + t) % currentU.t_max;
624         if (ind == 0) ind = currentU.t_max;
625
626         slice_count[0][t] = ind;
627         slice_count[1][t] = currentU.simplex_slices[ind].size();
628     }
629     for (const auto& row : slice_count) {
630         for (const auto& entry : row) {
631             file << entry << " ";
632         }
633         file << std::endl;
634     }
635     file.flush();
636
637     std::cout << "Universe: " << filename << std::endl;
638     std::cout << "t_big: " << t_big << ", t_max: " << currentU.t_max <<
        ↪  std::endl;
639
```

```
640     // Main loop to compute correlators
641     for (size_t t = 0; t < currentU.t_max; t++) {
642         uint32_t ind = (t_big + t) % currentU.t_max;
643         if (ind == 0) ind = currentU.t_max;
644         std::cout << ind << std::endl;
645 
646         file << "TIMESLICE: " << ind << "\n";
647         file << "s_i    s_j    distance(s_i, s_j)    {d/delta(s_i)
            ↪  d/delta(s_j)    d/d_i*d/d_j    d/d_i^2*d/d_j^2} for each
            ↪  radius 1 <= r <= r_max\n";
648 
649         auto& sslice = currentU.simplex_slices[ind];
650 
651         if (sslice.empty()) continue;
652 
653         n = 0;
654         //std::cout << ind << std::endl;
655         while (n < n_pairs) {
656             uint32_t size = sslice.size();
657             // Randomly select a simplex s_i
658             uint32_t i_i = random_simplex(size - 1);
659             uint32_t i_j = random_simplex(size - 1);
660             uint32_t s_i = sslice[i_i];
661             uint32_t s_j = sslice[i_j];
662 
663             // Compute distance between s_i and s_j
664             ricci.initialize(currentU.reduced_simplices);
665             dijkstra(ricci, currentU.reduced_simplices, s_i, size+1,
                ↪  s_j);
666             uint32_t distance = ricci.pathlen;
667 
668             // Compute avg sphere distances for s_i and s_j
669             ricci.initialize(currentU.simplex_neighbors);
670             file << s_i << " " << s_j << " " << distance << " ";
671 
672             for (uint32_t r = 6; r <= radius; r++) {
673                 double d_i = avg_sphere_dist(s_i, graph, r);
674                 double d_j = avg_sphere_dist(s_j, graph, r);
675 
676                 file << d_i << " " << d_j << " " << d_i * d_j << " " <<
                    ↪  std::pow(d_i, 2) * std::pow(d_j, 2) << " ";
677             }
678             file << "\n";
```

```
679             file.flush();
680
681             n++;
682         }
683     }
684
685     std::cout << "Univ = " << filename << ", t_big = " << t_big << ", r =
    ↪  " << radius << ", n = " << n << std::endl;
686 }
687
688 // Helper function to get all conf filenames from directory
689 std::vector<std::string> get_conf_files(const std::string& directory) {
690     std::vector<std::string> conf_files;
691
692     // Iterate over all files in the directory
693     for (const auto& entry :
    ↪  std::filesystem::directory_iterator(directory)) {
694         // Get the file name as a string
695         std::string filename = entry.path().filename().string();
696
697         // Check if the file name starts with "conf"
698         if (filename.rfind("conf", 0) == 0) {
699             conf_files.push_back(filename);
700         }
701     }
702
703     std::sort(conf_files.begin(), conf_files.end());
704
705     return conf_files;
706 }
707
708 // Helper function to output current timestamp
709 std::string get_current_time() {
710     // Get the current time from the system clock
711     auto now = std::chrono::system_clock::now();
712
713     // Convert to time_t for formatting
714     std::time_t current_time = std::chrono::system_clock::to_time_t(now);
715
716     // Convert to a tm struct for formatting (local time)
717     std::tm* local_time = std::localtime(&current_time);
718
719     // Use stringstream to format the output
```

```
720     std::stringstream ss;
721     ss << std::put_time(local_time, "%Y-%m-%d %H:%M:%S");
722 
723     return ss.str();
724 }
725 
726 
727 
728 /*
729 UTILITY SECTION
730 These are the functions that should be executed in the main function.
731 Either:
732 - time_it() for timing the code
733 - calculate[...]() for calculating correlator values from multiple conf
    ↪  files and saving them into txt files.
734 */
735 
736 // Function for timing the Code using a fixed case. Set timewhat = 0 for
    ↪  dijkstra(), any other int for avg_sphere_dist()
737 void time_it(uint8_t timewhat) {
738     std::string filename = "config.dat";
739     //auto config = Read_config(filename);
740     //auto result = process_config(config);
741 
742     //auto p = config.first;
743     //auto simplices = result.second;
744     //auto graph = build_graph(simplices);
745 
746     currentU.initialize(filename);
747     auto graph = currentU.simplex_neighbors;
748 
749     ricci.initialize(graph);
750     temp_ricci.initialize(graph);
751 
752 
753     if (timewhat == 0) {
754         auto start = std::chrono::high_resolution_clock::now();
755         dijkstra(ricci, graph, 277, 10000, 77080);
756         auto stop = std::chrono::high_resolution_clock::now();
757         auto duration =
            ↪  std::chrono::duration_cast<std::chrono::microseconds>(stop -
            ↪  start);
758         std::cout << "Time taken by dijkstra(): "
```

```
759             << duration.count() << " microseconds" << std::endl;
760         std::cout << ricci.pathlen << std::endl;
761     }
762     else {
763         auto start = std::chrono::high_resolution_clock::now();
764         double test_sph_dst = avg_sphere_dist(277, graph, 2);
765         auto stop = std::chrono::high_resolution_clock::now();
766 
767         auto duration =
        ↪  std::chrono::duration_cast<std::chrono::microseconds>(stop -
        ↪  start);
768         std::cout << "Time taken by avg_sphere_dist(): "
769             << duration.count() << " microseconds" << std::endl;
770         std::cout << test_sph_dst << std::endl;
771     }
772 }
773 
774 // For a single file: Function to calculate average sphere distance
    ↪  correlators over multiple configurations, radii, and distances
775 void calculate_singlef(const std::string& path, int file_ind) {
776     std::vector<std::string> files = get_conf_files(path);
777     std::vector<uint32_t> r_range = {6, 7, 8, 9, 10, 11, 12};  // Radii
    ↪  range
778     std::vector<uint32_t> dist_range(40); // Distance range from 1 to 40
779     std::iota(dist_range.begin(), dist_range.end(), 1);  // Fills the
    ↪  vector with 1, 2, ..., 40
780 
781     std::string conf = files[file_ind];
782 
783     currentU.initialize(path+conf);
784     auto graph = currentU.simplex_neighbors;
785 
786     std::cout << "Start: " << get_current_time() << std::endl;
787 
788 
789     for (uint32_t r : r_range) {
790         for (uint32_t d : dist_range) {
791             correlator(conf, graph, 50, r, d);
792         }
793     }
794 
795     std::cout << "Stop: " << get_current_time() << std::endl;
796 }
```

```
797
798 // Function to calculate average sphere distance correlators in the bulk
    ↪ region (using bulk corr)
799 void calculate_bulk(const std::string& path, int file_ind) {
800     std::vector<std::string> files = get_conf_files(path);
801     std::string conf = files[file_ind];
802
803     currentU.initialize(path+conf);
804     std::cout << "Start: " << get_current_time() << std::endl;
805
806     bulk_corr(conf, currentU, 50, 6);
807
808     std::cout << "Stop: " << get_current_time() << std::endl;
809 }
810
811 // For multiple files: Function to calculate average sphere distance
    ↪ correlators over multiple configurations, radii, and distances
812 void calculate(const std::string& path) {
813     std::vector<std::string> files = get_conf_files(path);
814     std::vector<uint32_t> r_range = {6, 7, 8, 9, 10, 11, 12};  // Radii
      ↪ range
815     std::vector<uint32_t> dist_range(40); // Distance range from 1 to 40
816     std::iota(dist_range.begin(), dist_range.end(), 1);  // Fills the
      ↪ vector with 1, 2, ..., 40
817
818     std::cout << "Start: " << get_current_time() << std::endl;
819
820     for (uint32_t r : r_range) {
821         for (const std::string &conf : files) {
822
823             currentU.initialize(path+conf);
824             auto graph = currentU.simplex_neighbors;
825
826             for (uint32_t d : dist_range) {
827                 correlator(conf, graph, 50, r, d);
828             }
829         }
830     }
831
832     std::cout << "Stop: " << get_current_time() << std::endl;
833 }
834
```

```
835 // Function to calculate average sphere distance correlators for pairs on
    ↪  timeslices, starting with the bulk region (using timesliced corr)
836 void calculate_timeslicewise(const std::string& path, int file_ind) {
837     std::vector<std::string> files = get_conf_files(path);
838     std::string conf = files[file_ind];
839 
840     currentU.initialize(path+conf);
841     std::cout << "Start: " << get_current_time() << std::endl;
842 
843     timesliced_bulk_corr(conf, currentU, 100, 6);
844 
845     std::cout << "Stop: " << get_current_time() << std::endl;
846 }
847 
848 /*
849 MAIN SECTION
850 Decide here which code to execute
851 */
852 
853 int main(int argc, char* argv[]) {
854 
855     //time_it(0);
856     //time_it(1);
857 
858     int file_ind = std::stoi(argv[2]);
859     //std::string path = argv[1];
860     calculate_timeslicewise(argv[1], file_ind);
861 
862     return 0;
863 }
```

References

[1] A. Zee, *Einstein gravity in a nutshell* (In a nutshell). Princeton: Princeton University Press, 2013, 866 pp., ISBN: 978-0-691-14558-7.

[2] S. Navas *et al.*, "Review of particle physics," *Phys. Rev. D*, vol. 110, no. 3, p. 030 001, 2024. DOI: 10.1103/PhysRevD.110.030001.

[3] C. Bambi and A. D. Dolgov, *Introduction to Particle Cosmology: The Standard Model of Cosmology and its Open Problems* (UNITEXT for Physics). Berlin, Heidelberg: Springer, 2016, ISBN: 978-3-662-48077-9 978-3-662-48078-6. DOI: 10.1007/978-3-662-48078-6. [Online]. Available: https://link.springer.com/10.1007/978-3-662-48078-6 (visited on 12/11/2023).

[4] C. H. Lineweaver and V. M. Patel, "All objects and some questions," *American Journal of Physics*, vol. 91, no. 10, pp. 819–825, Oct. 1, 2023, ISSN: 0002-9505. DOI: 10.1119/5.0150209. [Online]. Available: https://doi.org/10.1119/5.0150209 (visited on 11/13/2023).

[5] C. Kiefer, *Quantum gravity – an unfinished revolution*, Feb. 25, 2023. arXiv: 2302.13047. [Online]. Available: http://arxiv.org/abs/2302.13047 (visited on 11/13/2024).

[6] C. W. Misner, K. S. Thorne, J. A. Wheeler, and D. I. Kaiser, *Gravitation*, First Princeton University Press edition. Princeton Oxford: Princeton University Press, 2017, 1279 pp., ISBN: 978-0-691-17779-3.

[7] A. Zee, *Quantum Field Theory in a Nutshell* (In a nutshell), 2nd ed. Princeton, N.J: Princeton University Press, 2010, 576 pp., OCLC: ocn318585662, ISBN: 978-0-691-14034-6.

[8] J. Ambjorn, A. Goerlich, J. Jurkiewicz, and R. Loll, "Nonperturbative quantum gravity," *Physics Reports*, vol. 519, no. 4, pp. 127–210, Oct. 2012, ISSN: 03701573. DOI: 10.1016/j.physrep.2012.03.007. arXiv: 1203.3591[gr-qc,physics:hep-lat,physics:hep-th]. [Online]. Available: http://arxiv.org/abs/1203.3591 (visited on 07/31/2023).

[9] M. D. Schwartz. "Quantum field theory and the standard model," Higher Education from Cambridge University Press. ISBN: 9781108985031 Publisher: Cambridge University Press. (Dec. 15, 2013), [Online]. Available: https://www.cambridge.org/highereducation/books/quantum-field-theory-and-the-standard-model/A4CD66B998F2C696DCC75B984A7D5799 (visited on 12/10/2024).

[10] A. Maas, "The fröhlich-morchio-strocchi mechanism and quantum gravity," *SciPost Physics*, vol. 8, no. 4, p. 051, Apr. 6, 2020, ISSN: 2542-4653. DOI: 10.21468/SciPostPhys.8.4.051. [Online]. Available: https://scipost.org/10.21468/SciPostPhys.8.4.051 (visited on 12/11/2023).

[11] T. Regge, "General relativity without coordinates," *Il Nuovo Cimento (1955-1965)*, vol. 19, no. 3, pp. 558–571, Feb. 1, 1961, ISSN: 1827-6121. DOI: 10.1007/BF02733251. [Online]. Available: https://doi.org/10.1007/BF02733251 (visited on 11/16/2024).

[12] R. Loll, "Quantum gravity from causal dynamical triangulations: A review," *Classical and Quantum Gravity*, vol. 37, no. 1, p. 013 002, Jan. 9, 2020, ISSN: 0264-9381, 1361-6382. DOI: 10.1088/1361-6382/ab57c7. arXiv: 1905.08669[gr-qc,physics:hep-lat,physics:hep-th]. [Online]. Available: http://arxiv.org/abs/1905.08669 (visited on 07/31/2023).

[13] J. Brunekreef, A. Görlich, and R. Loll, *Simulating CDT quantum gravity*, Oct. 25, 2023. arXiv: 2310.16744[gr-qc,physics:hep-lat,physics:hep-th]. [Online]. Available: http://arxiv.org/abs/2310.16744 (visited on 10/27/2023).

[14] J. Ambjørn and R. Loll, *Causal dynamical triangulations: Gateway to nonperturbative quantum gravity*, Jan. 17, 2024. DOI: 10.48550/arXiv.2401.09399. arXiv: 2401.09399. [Online]. Available: http://arxiv.org/abs/2401.09399 (visited on 11/26/2024).

[15] J. Brunekreef, *Zooming in on the universe: In search of quantum spacetime*, Nov. 12, 2023. DOI: 10.48550/arXiv.2311.06910. arXiv: 2311.06910[gr-qc,physics:hep-lat,physics:hep-th]. [Online]. Available: http://arxiv.org/abs/2311.06910 (visited on 11/16/2023).

[16] J. Ambjorn, A. Goerlich, J. Jurkiewicz, and R. Loll, *CDT—an entropic theory of quantum gravity*, Jul. 15, 2010. arXiv: 1007.2560[gr-qc,physics:hep-lat,physics:hep-th]. [Online]. Available: http://arxiv.org/abs/1007.2560 (visited on 10/11/2023).

[17] J. Ambjorn, J. Jurkiewicz, and R. Loll, *Lorentzian and euclidean quantum gravity - analytical and numerical results*, Jan. 27, 2000. arXiv: hep-th/0001124. [Online]. Available: http://arxiv.org/abs/hep-th/0001124 (visited on 10/11/2023).

[18] J. Ambjorn, J. Jurkiewicz, and R. Loll, *Quantum gravity as sum over spacetimes*, version: 2, Jul. 6, 2009. DOI: 10.48550/arXiv.0906.3947. arXiv: 0906.3947. [Online]. Available: http://arxiv.org/abs/0906.3947 (visited on 12/15/2024).

[19] N. Klitgaard and R. Loll, "How round is the quantum de sitter universe?" *The European Physical Journal C*, vol. 80, no. 10, p. 990, Oct. 2020, ISSN: 1434-6044, 1434-6052. DOI: 10.1140/epjc/s10052-020-08569-5. arXiv: 2006.06263[gr-qc,physics:hep-th]. [Online]. Available: http://arxiv.org/abs/2006.06263 (visited on 09/26/2024).

[20] A. Zee, *Group theory in a nutshell for physicists* (In a nutshell). Princeton: Princeton University Press, 2016, 613 pp., ISBN: 978-0-691-16269-0.

[21] J. Ambjorn, J. Gizbert-Studnicki, A. Goerlich, and D. Nemeth, *IR and UV limits of CDT and their relations to FRG*, Nov. 4, 2024. DOI: 10.48550/arXiv.2411.02330. arXiv: 2411.02330. [Online]. Available: http://arxiv.org/abs/2411.02330 (visited on 11/23/2024).

[22] J. Berges, N. Tetradis, and C. Wetterich, *Non-perturbative renormalization flow in quantum field theory and statistical physics*, May 12, 2000. DOI: 10.48550/arXiv.hep-ph/0005122. arXiv: hep-ph/0005122. [Online]. Available: http://arxiv.org/abs/hep-ph/0005122 (visited on 11/23/2024).

[23] M. Reuter, *Nonperturbative evolution equation for quantum gravity*, May 6, 1996. DOI: 10.48550/arXiv.hep-th/9605030. arXiv: hep-th/9605030. [Online]. Available: http://arxiv.org/abs/hep-th/9605030 (visited on 11/23/2024).

[24] J. McCleary, *Geometry from a Differentiable Viewpoint*, 2nd ed. Cambridge: Cambridge University Press, 2012, ISBN: 978-0-521-11607-7. DOI: 10.1017/CBO9781139022248. [Online]. Available: https://www.cambridge.org/core/books/geometry-from-a-differentiable-viewpoint/BB6F7C972B3FA4990BBD774B8812D42B (visited on 06/04/2024).

[25] L. C. Loveridge, *Physical and geometric interpretations of the riemann tensor, ricci tensor, and scalar curvature*, version: 1, Jan. 23, 2004. DOI: 10.48550/arXiv.gr-qc/0401099. arXiv: gr-qc/0401099. [Online]. Available: http://arxiv.org/abs/gr-qc/0401099 (visited on 06/13/2024).

[26] J. van der Duin and R. Loll, *Curvature correlators in nonperturbative 2d lorentzian quantum gravity*, Apr. 26, 2024. DOI: 10.48550/arXiv.2404.17556. arXiv: 2404.17556[gr-qc,physics:hep-lat,physics:hep-th]. [Online]. Available: http://arxiv.org/abs/2404.17556 (visited on 06/13/2024).

[27] N. Klitgaard and R. Loll, "Introducing quantum ricci curvature," *Physical Review D*, vol. 97, no. 4, p. 046 008, Feb. 15, 2018, ISSN: 2470-0010, 2470-0029. DOI: 10.1103/PhysRevD.97.046008. arXiv: 1712.08847[gr-qc,physics:hep-lat,physics:hep-th]. [Online]. Available: http://arxiv.org/abs/1712.08847 (visited on 06/04/2024).

[28] Y. Ollivier, "A visual introduction to riemannian curvatures and some discrete generalizations," 2012. [Online]. Available: http://www.yann-ollivier.org/rech/publs/visualcurvature.pdf.

[29] N. Klitgaard and R. Loll, *Quantizing quantum ricci curvature*, Feb. 28, 2018. DOI: 10.48550/arXiv.1802.10524. arXiv: 1802.10524. [Online]. Available: http://arxiv.org/abs/1802.10524 (visited on 12/15/2024).

[30] E. Oks, "Brief review of recent advances in understanding dark matter and dark energy," *New Astronomy Reviews*, vol. 93, p. 101 632, Dec. 1, 2021, ISSN: 1387-6473. DOI: 10.1016/j.newar.2021.101632. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1387647321000191 (visited on 11/21/2024).

[31] D. R. Brill, "Method of the self-consistent field in general relativity and its application to the gravitational geon," *Physical Review*, vol. 135, no. 1, B271–B278, 1964. DOI: 10.1103/PhysRev.135.B271.

[32] P. R. Anderson and D. R. Brill, *Gravitational geons revisited*, Sep. 3, 1997. arXiv: gr-qc/9610074. [Online]. Available: http://arxiv.org/abs/gr-qc/9610074 (visited on 11/21/2024).

[33] B. Guiot, A. Borquez, A. Deur, and K. Werner, *Graviballs and dark matter*, Sep. 3, 2020. arXiv: 2006.02534. [Online]. Available: http://arxiv.org/abs/2006.02534 (visited on 11/21/2024).

[34] C. Gattringer and C. B. Lang, *Quantum Chromodynamics on the Lattice: An Introductory Presentation* (Lecture Notes in Physics). Berlin, Heidelberg: Springer, 2010, vol. 788, ISBN: 978-3-642-01849-7 978-3-642-01850-3. DOI: 10.1007/978-3-642-01850-3. [Online]. Available: https://link.springer.com/10.1007/978-3-642-01850-3 (visited on 11/21/2024).

[35] I. Montvay and G. Münster, *Quantum Fields on a Lattice* (Cambridge Monographs on Mathematical Physics). Cambridge: Cambridge University Press, 1994, ISBN: 978-0-521-59917-7. DOI: 10.1017/CBO9780511470783. [Online]. Available: https://www.cambridge.org/core/books/quantum-fields-on-a-lattice/4401A88CD232B0AEF1409BF6B260883A (visited on 11/21/2024).

[36] D. Németh, *Private communication*, 2023.

[37] *5-cell*, in *Wikipedia*, Page Version ID: 1247473715, Sep. 24, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=5-cell&oldid=1247473715 (visited on 11/22/2024).

[38] "Shortest paths," in *Algorithms and Data Structures: The Basic Toolbox*, K. Mehlhorn and P. Sanders, Eds., Berlin, Heidelberg: Springer, 2008, pp. 191–215, ISBN: 978-3-540-77978-0. DOI: 10.1007/978-3-540-77978-0_10. [Online]. Available: https://doi.org/10.1007/978-3-540-77978-0_10 (visited on 11/29/2024).

[39] J. Ambjorn, A. Gorlich, J. Jurkiewicz, and R. Loll, *Planckian birth of the quantum de sitter universe*, Jan. 8, 2009. DOI: 10.48550/arXiv.0712.2485. arXiv: 0712.2485. [Online]. Available: http://arxiv.org/abs/0712.2485 (visited on 12/30/2024).

[40] J. Ambjorn, A. Goerlich, J. Jurkiewicz, and R. Loll, *The nonperturbative quantum de sitter universe*, Jan. 8, 2009. DOI: 10.48550/arXiv.0807.4481. arXiv: 0807.4481. [Online]. Available: http://arxiv.org/abs/0807.4481 (visited on 12/30/2024).

[41] P. Adhikari *et al.*, "First direct detection constraints on planck-scale mass dark matter with multiple-scatter signatures using the DEAP-3600 detector," *Physical Review Letters*, vol. 128, no. 1, p. 011 801, Jan. 5, 2022, ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.128.011801. arXiv: 2108.09405[astro-ph]. [Online]. Available: http://arxiv.org/abs/2108.09405 (visited on 03/13/2025).

List of Figures

List of Tables