Probabilistic Numerical Solutions of Partial Differential Equations on Riemannian 2-Manifolds

Master's Thesis in Computer Science

Theo Rüter Würtzen

Department of Computer Science
University of Copenhagen

Supervised by

Peter Nicholas Krämer

Section for Cognitive Systems
Technical University of Denmark

Søren Hauberg

Section for Cognitive Systems
Technical University of Denmark

Sebastian Weichwald

Department of Mathematical Sciences
University of Copenhagen

Abstract

Partial differential equations are ubiquitous in physics and simulations, and generally have to be solved numerically. The field of probabilistic numerics focuses on framing computation as probabilistic inference and has led to the development of efficient and competitive probabilistic numerical solvers of ordinary differential equations, which give the solution as a stochastic process. In this thesis, we extend the use of these solvers to partial differential equations on Riemannian manifolds. We give introductions to the touched upon topics, including but not limited to discretization of Riemannian manifolds, the associated discrete differential operators, and the use of stochastic differential equations as priors. We give an algorithm to build an intrinsic triangulation of a manifold, and empirically demonstrate how certain priors can accelerate the process of solving nonlinear PDEs. We have structured the thesis as a high level guide on how to implement and follow the results, and we try to motivate the steps to the best of our efforts. We teach and explain with a focus on giving intuition through examples and analogies, and provide pointers to further reading.

Acknowledgements

I thank Nicholas for his high availability and thorough eye for detail during every supervision session and for including me in the fun sessions at the research group. I thank Søren Hauberg and Sebastian Weichwald for their enthusiasm and for enabling this project. Finally, I pat myself on the back for putting in a lot of work.

Contents

1 Overview

The thesis can broadly be seen as consisting of two separate parts that meet towards the end. One deals with Riemannian manifolds and how to convert them to a discrete form amenable to numerical computation. The other deals with ordinary / partial / stochastic differential equations and probabilistic numerical solvers. Finally these two parts converge, and we will use the discrete Laplacian and the probabilistic numerical solvers to solve partial differential equations whose spatial domain is the manifold.
The thesis touches upon many subjects, some of which are active areas of research and could have been an entire thesis by themselves. We will supplement what is not taught in a classic computer / data science curriculum, but a full coverage of the topics is beyond the scope of this thesis. Nearly all figures in the thesis were made during the project and have subsequently been included in the thesis because they were found to be helpful in understanding the topics.
The main parts of the thesis are structured as follows:

Chapter 2 is the most verbose part of the thesis and serves to get the reader into the mood and right mindset for the rest of the thesis. It gives a brief introduction to the field of probabilistic numerics and further motivates the topics of the thesis.

Chapter 3 gives an applied introduction to Riemannian manifolds, triangle meshes and the discrete exterior calculus, all with the intention of defining the discrete Laplacian on a discrete manifold.

Chapter 4 builds on chapter 3 and addresses the problem of building the Laplacian matrix given only a Riemannian metric. We contribute an algorithm that solves this problem by constructing an intrinsic triangulation of a manifold, from which one can then build the Laplacian matrix using the tools from 3. We demonstrate the algorithm on a few examples.

Chapter 5 is the second part of the thesis. It gives an introduction to ordinary and partial differential equations. We discuss the ideas of the state-space representation and discretization using the Method of Lines, which is the principle we will use in the later probabilistic numerical solver.

Chapter 6 builds on chapter 5 to introduce the probabilistic numerical solver for ordinary differential equations. We will cover how to specify a prior distribution over the solution using a stochastic differential equation, show how to manipulate the prior and posterior probability distributions, and give specific pointers on how to implement a stable solver.

Chapter 7 wraps up with a demonstration of the solver on a few examples. We show how the solver can be used to solve nonlinear PDEs, and contribute problem-specific priors that improve the speed of convergence.

2 Background

Probabilistic Numerical Methods

The field of probabilistic numerics methods frames traditional numerical tasks as problems of probabilistic inference. This approach enables the explicit modeling of uncertainty arising from limited computational resources [61], discretization errors [34], or incomplete information [7]. The book [26] is a comprehensive introduction to the field. Probabilistic numerics has yielded Bayesian approaches to solutions of linear algebra problems [24], optimization problems [40], and differential equations [54], leading to algorithms capable of not only producing a solution but also quantifying the uncertainty in that solution. Given prior beliefs about the unknown solution and a likelihood model informed by computational observations (e.g., evaluations of a differential operator), probabilistic solvers infer a posterior distribution over the solution.

Numerical Solvers for Differential Equations

Traditional numerical methods for solving differential equations, such as the Runge-Kutta [48] methods, produce deterministic approximations of the solution. One can choose a trade-off between accuracy and computational efficiency [11], expressed by the convergence rate [31] of the method.

On the other hand, Probabilistic Numerical solvers represent the solution as a distribution of functions [36] parameterized by a Gaussian process [45]. Instead of providing a single point estimate of the solution, these solvers return a probability distribution that encapsulates the uncertainty over the solution. Using a probabilistic solver necessitates the choice of a prior distribution over the solution, which affects the behaviour of the solver. This is not a drawback, but a feature, as the choice of prior can encode prior knowledge about the solution, such as smoothness or periodicity [45].

Why Uncertainty Quantification?

The uncertainty of a solution is often seen as a by-product of the numerical method, but it serves as an important enabler for dynamic decision-making, especially in safety-critical applications. For example, in a medical setting, it would allow doctors to manually prescribe a treatment or get third opinions if the model shows uncertainty about its diagnosis. At its best, it allows reasoning about the range of possible outcomes to further propagate the uncertainty through later stages of a system.
Active learning [38] is a field in which modeling uncertainty plays a big role. Under a limited budget, one can optimize to reduce uncertainty about some quantity of interest. In machine learning, it serves to identify regions of data where models are prone to errors, and can guide data collection [3]. In [34], the authors model the uncertainty of spatial and temporal discretization errors in probabilistic numerical

solvers of ODEs, which enables prioritized discretization where it will bring the most benefit. To truly be able to rely on uncertainty quantification, it must be calibrated, meaning that the uncertainty should be a good estimate of the true uncertainty. Fortunately, [6] show that probabilistic solvers of ODEs do indeed provide well- calibrated uncertainty estimates.

Why Differential Equations?

Differential equations (DEs) play a central role in the natural sciences as tools for modeling and understanding dynamic systems. They describing the evolution of systems over time and/or space and are commonly found in physics, biology, engineering and even finance. In physics, famously Schrödinger's equation models the quantum state of a system, and the Navier-Stokes equations model fluid flow. In the earth sciences, models of climate systems rely on multiple sets of DEs. In biology, reaction-diffusion equations describe processes like morphogenesis*. Modeling uncertainties here is also useful - an example that comes to mind is the chance-of-rain† in a weather forecast, or the uncertainty in the prediction of a stock price.

Differential Equations in Machine Learning

Differential equations are also finding increasing relevance in machine learning [14] [32]. Neural ODEs enable learning continuous-time models of data, offering a more flexible alternative to discrete-time approaches like the RNN [53]. This has led to the rise of physics-informed neural networks [44] that are trained to satisfy some specific DE. Diffusion models [28] and normalizing flows [47] are examples of probabilistic generative models that leverage the language of DEs to define complex distributions. Generally, these approaches require some form of a numerical solver of DEs, and using probabilistic solvers can provide an alternative to the standard ones if calibrated uncertainty estimates are needed.

Manifolds in Machine Learning

Riemannian manifolds provide a mathematical framework for studying curved spaces, generalizing the notion of Euclidean geometry ($\mathbb{R}^n$) to non-Euclidean settings. They serve also as domains for constrained optimization problems [12]. Some data is not naturally represented well in a Euclidean space, the simplest example being angles, which wrap around. Another example is the manifold of positive semidefinite matrices [60], useful when fitting Gaussian mixture models. The link between probabilistic numerical solvers and manifolds has been made before in [25] where the authors compute geodesics (shortest paths) on manifolds using probabilistic solvers. In machine learning, taking into account the geometry of the data can make simple models competitive [23] and lead to higher interpretability [1].

Why the Focus on 2-Manifolds

When one could have chosen to work with 4-dimensional manifolds, choosing 2-dimensional manifolds can seem like a boring choice. Although the continuous mathematical description of manifolds is mature well beyond 2 dimensions, the discrete counterparts are not as well-developed[16]. Triangle meshes are a common discretization of manifolds, and in 3D finite-element methods one will encounter the tetrahedral mesh. 2-dimensions are a good starting point because they can be visualized and there are lots of interesting problems to solve in 2D [51] [62] [57].

*This is the process that leads to biological cells having a specific shape.

†Apparently, this is formally known as "Probability of precipitation"

3 Riemannian Manifolds and Discrete Exterior Calculus

This section covers relevant knowledge of Riemannian 2-manifolds by motivating and explaining key concepts through definitions and examples. The intention is to give intuition and working knowledge relating to later applications and algorithms. The focus will be on 2-manifolds, but the definitions and ideas will carry into higher (or lower) dimensions with the same principles. We then move to the theory of Discrete Exterior Calculus, which provides us with a principled way of computing discrete differential operators. We will focus on the topics that lead up to the Laplacian, which will be central to the PDEs we will be solving.

Riemannian 2-Manifolds

For millenia society thought that earth was a flat, because that's what it looks like up close. If we model earth as a unit-ball, then its surface, the unit-sphere, looks locally flat. Mathematically, we say that each point on the sphere is locally topologically equivalent (homeomorphic) to $\mathbb{R}^2$.

Definition: Manifold

An n-manifold is a topological space with the property that each point has a neighborhood that is homeomorphic to an open subset of $\mathbb{R}^n$.

Although the surface of the earth lies in our physical 3-dimensional space, we humans have found a way to parametrize it in terms of just two numbers, latitude and longitude, which yield a local representation. The Equirectangular projection maps the sphere to $[-\pi/2, \pi/2) \times [-\pi, \pi]$, which are spherical coordinates that correspond exactly to the geographical coordinates Lat° and Lon°. One might have thought that this was a description of the Mercator projection, but the Mercator projection stretches the latitude near the two poles.

Definition: Coordinate Chart

A 2D coordinate chart of $\mathcal{M}$ is a pair (C, f), where f is a diffeomorphism[a]

$$f : U \to C$$

that maps open subset $U \subset \mathcal{M}$ of a 2-manifold $\mathcal{M}$ to an open subset $C \subseteq \mathbb{R}^2$, resulting in a coordinate space representation C of U.

[a] A diffeomorphism is a bijection between two differentiable manifolds such that both it and its inverse are continuous. Strictly speaking, a homeomorphism is enough here (meaning not necessarily differentiable), but since we are ultimately interested in doing calculus on our manifold, we need a differentiable representation of it.

Intuition: Coordinate Chart

We will use the surface of the earth as $\mathcal{M}$ and construct a our own coordinate chart. For simplicity, we are modeling it as a unit-sphere and will only be mapping the open[a] upper hemisphere $\subset \mathcal{M}$ to the open unit-disc C by projecting the hemisphere onto the unique plane that cuts through the equator.

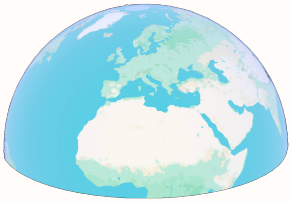


Figure 1: The northern hemisphere obtained by cutting through the equator. Texture: Google Earth

An invertible map from the unit-disc to the hemisphere is given by

$$\text{disc-to-hemisphere}(\begin{pmatrix} x & y \end{pmatrix}^\top) = \begin{pmatrix} x & y & \sqrt{1-x^2-y^2} \end{pmatrix}^\top$$

The x, y coordinates span the equatorial plane and the third coordinate is the height of the hemisphere. The inverse of this map corresponds exactly to projecting the hemisphere onto the equatorial plane.

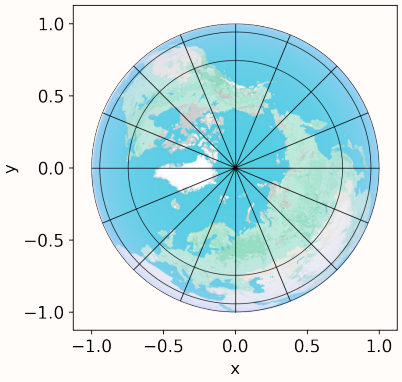


Figure 2: The northern hemisphere projected down onto the equatorial plane. The north pole is placed at local coordinates $(0, 0)$. Three circles of latitude are shown, equally far apart on the hemisphere - note how our map does not render them equally spaced. We will further study the properties of this map in a later example.

[a] This means we exclude the equator.

Metric Tensor

Good maps are characterized by being amenable to direct measurements. A weakness of the Equirectangular Projection is the distortion of landmasses at the poles. They are disproportionately big when compared to a countries such as Kenya at the equator. This difference in scale invalidates direct measurements of lengths using a conventional ruler. The Metric tensor

quantifies the local distortions induced by the map, and is the primary tool for measurements on coordinate charts. Before giving a definition, we will further motivate the Metric tensor by pointing out what one takes for granted in Euclidean space (specifically, $\mathbb{R}^{2\ddagger}$). $\mathbb{R}^2$ is a vector space, meaning that linear combinations of the basis-vectors e_1 and e_2 are still members of $\mathbb{R}^2$. In a vector space we can additionally define a norm and an inner product. The Euclidean norm is defined as $||(x,y)|^\top|_2 = \sqrt{x^2+y^2}$ and is used to define the distance $d(\vec{a},\vec{b}) = ||\vec{a}-\vec{b}||_2$. The Euclidean inner product $\langle \vec{a},\vec{b}\rangle = a_1b_1 + a_2b_2$ relates to the angle θ (radians) between vectors through $\frac{\langle \vec{a},\vec{b}\rangle}{||\vec{a}||||\vec{b}||} = \cos(\theta)$.
In contrast to $\mathbb{R}^2$, the surface of the earth is not a vector space. There can be no set of basis vectors, because there is no defined notion of addition or scaling of coordinates. We do not add or scale pairs (Lat°., Lon°.), and we don't have a consistent way to assign meaning to it. Although we have coordinates, they by themselves don't lead to distances or angles, because we don't have the inner product or norm defined. Before diving into the metric tensor, we will have to define the notion of the tangent space.

Definition: Tangent space

The tangent space $\mathcal{TM}_p$ at a point p on a 2-manifold $\mathcal{M}$ is the 2-dimensional real vector space consisting of all tangent vectors to $\mathcal{M}$ at p. Each tangent vector can be associated with the velocity of a curve on $\mathcal{M}$ passing through p.

The tangent space is a local "linearization" of the manifold around a point p. On the sphere, the tangent space of the north pole is the unique tangent plane that intersects only the north pole. Moving from p along a vector from the tangent space will in general take one away from the manifold.

Definition: Metric Tensor

When using a 2-dimensional coordinate system, the 2D metric tensor is a symmetric, positive-definite 2×2 matrix $g(p) = M \in \mathbb{R}^{2\times2}$ where p is a point on the two-dimensional surface specified in local coordinates. M provides a way to measure lengths, angles by describing how the infinitesimal length ds of a change in local coordinates $t = \begin{pmatrix} dx & dy \end{pmatrix}^\top \in \mathcal{TM}_p$ is computed as

$$\begin{aligned} ds^2 &= M_{11}dx^2 + 2M_{12}dxdy + M_{22}dy^2 \\ &= \begin{pmatrix} dx & dy \end{pmatrix} \begin{bmatrix} M_{11} & M_{12} \\ M_{12} & M_{22} \end{bmatrix} \begin{pmatrix} dx \\ dy \end{pmatrix} \\ &= t^\top M\, t \end{aligned}$$

and angle θ between tangent vectors $t, t' \in \mathcal{TM}_p$ are computed as

$$\cos(\theta) = \frac{t^\top M\, t}{\sqrt{t^\top M\, t}\sqrt{t'^\top M\, t'}} \tag{1}$$

The explicit dependance of the metric tensor $g(p)$ on the coordinate p is often omitted in notation, but we will keep it explicit.

Definition: Riemannian Metric

A 2D Riemannian metric g maps each local coordinate $p \in C$ to a metric tensor

$$g : C \to \mathbb{R}^{2\times2}$$

To compute gradients and curvature (see later section), the metric must be at least twice differentiable.
Informally, ds, dx, and dy are the same symbol as dx in $\int_0^1 f(x)\,dx$. They are 'differentials'. Intuitively, if we were to relate the integral back to the Riemann sum, dx is the size of the interval in our partition of $[0,1] \subset \mathbb{R}$. In the current case, by having ds depend on the position in local coordinates (through the dependence of the metric on p), we can assign "bigger partitions" to some areas, which lets us weigh different areas differently. We will use dependence to counteract the non-uniform stretching induced by mappings like Equirectangular projection or our projection of the hemisphere. Positive-definiteness in the definition ensures that all squared infinitesimal distances ds^2 will remain positive, no matter the entries in g.

Intuition: Pythagorean Theorem

g can be thought of as encoding a local version of the Pythagorean Theorem. If $g = I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, then the formula for infinitesimal distance reduces to exactly

$$ds^2 = dx^2 + dy^2$$

which is similar to $c^2 = a^2 + b^2$. In fact, I_2 is the metric tensor of $\mathbb{R}^2$ independently of position p. This reflects the fact that the Pythagorean theorem holds at all positions in $\mathbb{R}^2$, not just around the origin or some other specific point. If the entries of g change, we will get different identities.

Intuition: Visualizing the Metric Tensor

In cartography, a common way to visualize the degree distortion of the chart is using the Tissot Indicatrix. It shows how a patch of equal area and shape on the earth will be represented on the map. A good approximation of the Tissot Indicatrix can be computed using the metric tensor, by showing (scaled[a]) eigenvectors and -values of the inverse metric tensor as an ellipse, centered at the point of evaluation. ▶

‡ $\mathbb{R}^2$ is technically the set of all ordered pairs of real numbers, but in the context of differential geometry, it is also the Euclidean plane with the Euclidean metric.

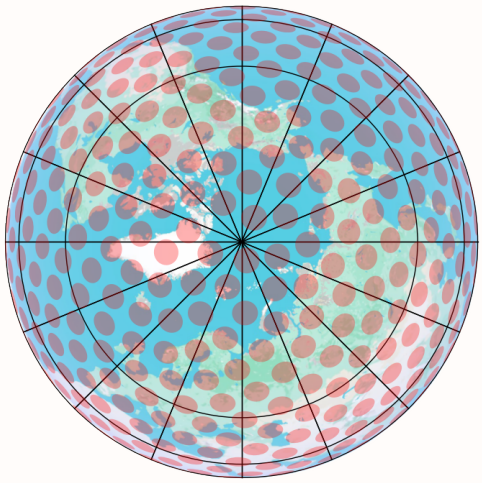


Figure 3: The hemisphere-map with Tissot Indicatrices at evenly spread locations on the surface of the earth. The radius of the ellipse in a certain direction can be interpreted as the speed at which someone traveling at constant speed on hemisphere would be seen traveling at when viewed from the map.

Another way visualizing 2D metric tensors are with a pseudo-mesh, which is explored in [59].

[a]The eigenvalues and vectors have been scaled down with a factor of 0.003 for this figure.

Intuition: Rank of Metric Tensor

The determinant of the 2D Metric Tensor reflects how areas on the surface are scaled locally. By positive-definiteness, the rank of the 2D metric tensor $g(p)$ will always be 2 and the determinant strictly positive. However, positive-semidefinite (rank-deficit) metric tensors can be interesting to study for intuition.

Consider a metric tensor $\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}$, for which the infinitesimal length-formula reduces to $ds = dx$. With this metric, locally the length of infinitesimal $\begin{pmatrix}dx & dy\end{pmatrix}^\top$ depends only on dx with no contribution from dy. At this point the space is locally one dimensional, with space being collapsed in direction of the second coordinate basis-vector. For this metric, the $\cos(\theta)$ formula (1) will yield only 0 and π, which are the only two possible angles between parallel vectors.

For a rank 0 matrix (which is the 2x2 zero matrix), $ds = 0$, and angles are undefined as a division by zero occurs.

Measuring the length of a path with the Metric Tensor

If a sailor plans a route (not necessarily straight) on their chart, they will be keen to know the length it represents. From the distortion of the maps seen earlier, we have determined that we cannot simply measure directly, as lengths are not guaranteed to have been represented faithfully. To measure exactly the length of a route, we will have to integrate the infinitesimal length ds of the velocity vector of the line over the whole span of the line.

We will parameterize the route r with parameter $t \in (0, T)$ to get $r : (0, T) \to C \subset \mathbb{R}^2$. r is in fact a 1-manifold, and the parameterization a choice of local coordinate system.

Symbolically, the length can be written simply as $L(r) = \int_r ds$ where ds is the infinitesimal length obtained from the metric tensor $g(p)$. But this is not very computable definition. By using the local parameterization we get:

$$L(r) = \int_{[0,T]} \sqrt{\nabla r(t)^\top g(r(t))\, \nabla r(t)}\, dt \qquad (2)$$

Even for many seemingly simple cases of g and r this integral has no analytic solution, because it turns out to be an instance of an elliptic integral, which are integrals of the form $\int \sqrt{P(t)}$, where P is some polynomial of t. There will be an instance of this in chapter 4, where we will resort to numerical methods.

Intuition: Computing the Metric on the chart induced by the 3D hemisphere

The Riemannian metric on the chart induced by the 3D hemisphere can be computed as

$$g(p) = J(p)^\top J(p) \in \mathbb{R}^{2\times 2}$$

where $J(p) \in \mathbb{R}^{3\times 2}$ is the Jacobian matrix of disc-to-hemisphere$(\cdot)$ at point p, which maps tangent vectors $\in \mathbb{R}^2$ to tangents vectors $\in \mathcal{TM} \subset \mathbb{R}^3$. The Jacobian comes out as

$$J\left(\begin{pmatrix}x\\ y\end{pmatrix}\right) = \begin{bmatrix}1 & 0\\ 0 & 1\\ \frac{-x}{\sqrt{1-x^2-y^2}} & \frac{-y}{\sqrt{1-x^2-y^2}}\end{bmatrix}$$

which gives the final metric of

$$g_{\text{hemisphere}}\left(\begin{pmatrix}x\\ y\end{pmatrix}\right) = \frac{1}{1-x^2-y^2}\begin{bmatrix}1 & xy\\ xy & 1\end{bmatrix}$$

This is symmetric, positive definite and well-defined for $x^2 + y^2 < 1$, which is exactly the open unit-disc. At the center, $(0, 0)$, the metric tensor coincides with the Euclidean metric, reflecting the fact that the north-pole did not get distorted during the projection, as it was already aligned with the plane we project onto.

Definition: Riemannian Manifold

A Riemannian Manifold is a pair $(\mathcal{M}, g)$ of a smooth manifold $\mathcal{M}$ with Riemannian metric g.

Definition: Isometric Immersions and Embeddings, Ambient Space

An isometric immersion of a Riemannian 2-Manifold $(\mathcal{M}, g)$ into a 2-dimensional submanifold $\hat{\mathcal{M}} \in \mathbb{R}^3$ is a map $f : \mathcal{M} \to \hat{\mathcal{M}}$ such that the distance between any two points in $\mathcal{M}$ under g is the same as the distance between their images under f measured along the surface of $\hat{\mathcal{M}} \in \mathbb{R}^3$. If f is additionally a bijection, meaning that $\hat{\mathcal{M}}$ does not self-intersect, f is an embedding. If it self-intersects it is an immersion. The space that the manifold is mapped into is referred to as the ambient space. Throughout this thesis, where applicable, the ambient space is $\mathbb{R}^3$.

For our purposes, having an immersion will be sufficient. The embedding is useful for visualization purposes, but is not necessary for the numerical methods we will be using. By construction, disc-to-hemisphere$(\cdot)$ is an isometric embedding of the Riemannian Manifold given by

$$(\text{open unit-disc},\ g_{\text{hemisphere}})$$

Another interesting metric we will be working with later is the hyperbolic metric, which is defined on the hyperbolic plane $\mathbb{H}^2$.

Intuition: The Hyperbolic Metric

The metric of the 2-dimensional hyperbolic plane $\mathbb{H}^2$ is defined as

$$g_{\text{hyperbolic}}\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = \frac{1}{1 - x^2 - y^2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

with $\begin{pmatrix} x & y \end{pmatrix}^\top$ as local cartesian coordinates within the unit-disc. The hyperbolic metric is used in the Poincaré disk model of the hyperbolic plane, which, unlike the sphere, is unbounded. It is also not isometrically embeddable in $\mathbb{R}^3$ (Hilbert's Theorem). Subsets of it can however be embedded. The authors of [15] give a method to embed a portion of the hyperbolic plane in $\mathbb{R}^3$ and have a great visualization[a].

Figure 4: The surface of kale leaves is hyperbolic. The veins of the leaf are approximately geodesics, which are the shortest paths between points on the leaf. Similarly to how an orange peel cannot be flattened to a sheet without tearing, the kale leaf cannot be flattened to a plane without folding in on itself. One has constant positive gaussian curvature, the other negative.

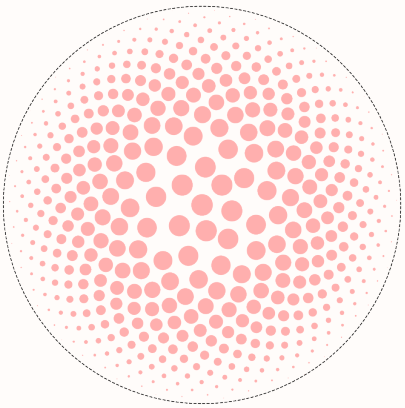


Figure 5: Tissot Indicatrices shown on Poincaré disc model of the hyperbolic plane. The metric is isotropic, meaning not depending on direction of travel, so the patches are circular. Note that in this figure the indicatrices are *not* uniformly spaced like they were on the hemisphere.

[a] For copyright reasons, we will have to make do with the surface of kale leaves, which is a common example of hyperbolic geometry in nature anyway.

Boundaries

Something we have so far been evading is the question of boundaries on a 2-manifold. The boundary of a 2-manifold $\mathcal{M}$ is denoted $\partial\mathcal{M}$ and is a 1-manifold. The boundary of a boundary is always empty. Boundaries are common in the context of numerical solutions of PDEs because of our bounded computational power. If one has an unbounded manifold (say, $\mathcal{M} = \mathbb{R}^2$ or $\mathbb{H}^2$), one must solve the PDE on a subset of the manifold instead of the whole, which means that a boundary is introduced. However not all manifolds must be truncated like this. The sphere or torus or surfaces of everyday objects are all bounded 2-manifold without a boundary, which means the PDE can usually be solved on the entire domain, and complications with boundaries are avoided.

Intrinsic and Extrinsic Properties

Properties of manifolds can be grouped into two categories. Extrinsic properties require the manifold to be viewed as part of the ambient space. The normal vector, which by definition is orthogonal to the tangent space at each point, and position in ambient space are both extrinsic properties whose definition require the

ambient space.
Intrinsic properties are independent of any embedding and arise solely from the manifold's internal structure. Lengths of paths and angles of tangent vectors are examples.

Representing Manifolds on a Computer

We now move to the practical side, anticipating the future involvement of a computer. We will start by introducing the two representations of Riemannian 2-manifolds of interest: Manifolds given by a local coordinate system and a metric, or manifolds represented as a triangulation with triangle meshes. Technically, we are representing them as 'Simplicial Complexes' [16], but the term 'triangulation' will be more familiar and simplify the explanation.

Triangulation

Extrinsic Geometry

A triangle mesh is a collection of vertices $\{V_i\}_{i=0}^{|V|}$ and a set of faces $\{F_i\}_{i=0}^{|F|}$, with $V_i \in \mathbb{R}^3$ and $F_i \in V \times V \times V$. A face is a an ordered triple of vertices, but even permutations of a face are considered the same face. We say that the face is oriented in one of two ways - this is determines the ambiguity of the normal vector. Edges are defined implicitly by the faces. A triangle mesh is most commonly represented with each vertex having coordinates in the ambient space. To tie this representation to a specific metric, it is most natural to assume that it represents an isometric immersion (or embedding if the mesh has no self-intersections) of a 2-manifold. This common representation of a triangulation is called an extrinsic triangulation of the 2-manifold.

Intrinsic Geometry

The extrinsic triangulation allows one to compute edge-lengths from the norm of the difference between two adjacent vertices. We can however also work with an intrinsic triangulation, wherein no reference to the ambient space is made. This representation stores a set of edges $\{E_i\}_{i=0}^{|E|}$ and their lengths $\{e_i\}_{i=0}^{|E|}$. The faces are stored like previously, but defined through an ordered triple of edges instead of vertices. Like faces, edges are an ordered tuple with two possible orientations that must be kept track of. If a face (E_i, E_j, E_k) has associated lengths (e_i, e_j, e_k), there are some restrictions that the lengths must satisfy:

$$0 < e_i < e_j + e_k$$

$$0 < e_j < e_i + e_k$$

$$0 < e_k < e_i + e_j$$

This is the strict variant of the triangle inequality, ensuring that all lengths define triangles that can be embedded in $\mathbb{R}^2$ with non-zero area.
Many advantages of intrinsic triangulations are explored in depth in [55]. The following are useful formulas for computing areas and angles of faces in an intrinsic triangulation, and were necessary for the implementation of chapter 4.

Intuition: Heron's Formula for calculating Areas of Faces

Given a face (E_i, E_j, E_k) the area can be computed as

$$\text{Area} = \sqrt{s(s-e_i)(s-e_j)(s-e_k)} \tag{3}$$

where $s = \frac{e_i+e_j+e_k}{2}$ is the semiperimeter of the triangle.

A good rule of thumb is that if a property of the manifold cannot be computed from edge-lengths alone, it is not extrinsic [55]. The normal vector or dihedral angles across triangles are examples of these.

Intuition: The Law of Cosines for calculating Interior Angles of Faces

Given again a face (E_i, E_j, E_k), the cosine of the interior angle α opposite of edge E_i can be computed as

$$\cos(\alpha) = \frac{e_j^2 + e_k^2 - e_i^2}{2e_j e_k}$$

In a subsequent chapter, chapter 4, we will dive much deeper into intrinsic triangulations, and will give more motivation on why they are useful and when they arise. If confused, think of a triangulation as meaning an extrinsic triangulation for now, which is what is commonly meant when one says "triangle mesh".

Manifold Triangle Mesh

When given an extrinsic triangulation, it is most natural to assume that it represents an isometric immersion (or embedding if the mesh has no self-intersections) of a section of a 2-manifold. Furthermore, for a triangle mesh to represent a 2-manifold, there are some restrictions to how the faces can be connected. An edge must be associated with $n \in \{1, 2\}$ faces, with the case $n = 1$ is realized only at boundary faces. Mesh structures with $n > 2$ can be permitted with some tricks that involve "inflating" the mesh (see [56]), but we will not cover them here and work only with manifold meshes. An additional constraint is that a face cannot share an edge with itself.

Definition: Dual Mesh

In an extrinsic triangulation, the dual of a triangle mesh $(V,\ F)$ is a new mesh $(V',\ F')$ with

$$|V'| = |F| \text{ and } |F'| = |V|$$

The vertex V_i' is placed at the center of face F_i of the original mesh, and new faces F_i' are centered at V_i. The two dominant definitions of a triangular center are the barycenter and circumcenter, which lead to the notions of barycentric dual mesh and the circumcentric dual mesh. The barycenter B_i of face $F_i = (V_i, V_j, V_k)$ is simply defined as $B_i =$ ▶

$\frac{1}{3}(V_i + V_j + V_k)$, while the circumcenter is the center of the circumcircle, which is the unique circle that passes through all three vertices.
In the dual mesh, two faces that share an edge E_i in the original mesh will become two dual vertices connected by the dual edge E_i'.

Intuition: Choice of Dual Mesh

The choice of dual will influence edge-lengths and dual face areas (see following figures).

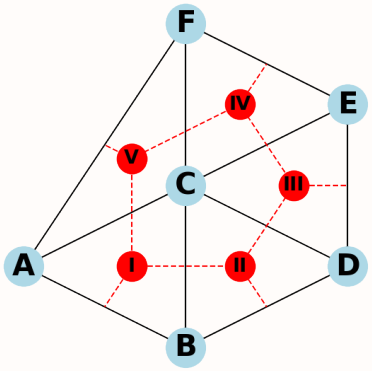


Figure 6: Barycentric Dual mesh. The dual vertices in red are placed at the barycenter of each face, guaranteeing a central position and positive edge-lengths. For clarity, edges (I, II) and (B, C) are dual, while polygon (I, II, III, IV, V) is dual to vertex C.

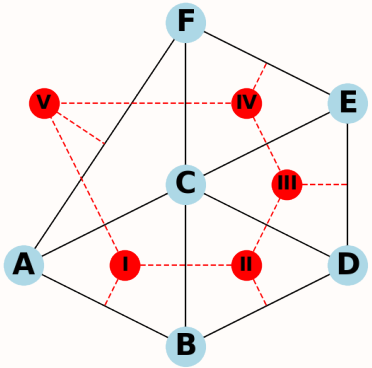


Figure 7: Circumcentric Dual mesh. Each dual vertex is placed at the circumcenter of the face. For obtuse triangles such as FCA, the circumcenter V lands outside the face. The length of the dual edge is then considered to be negative. The circumcentric dual has the property that pairs of edges and dual edges are orthogonal. Contrast this with the barycentric dual edge pair (A, C), (I, V).

In an intrinsic triangulation, the mapping of vertices to dual faces and faces to dual vertices is the same, but the circumcenter and barycenter cannot be defined explicitly as there are no stored vertex coordinates. One must instead store the length of the dual edge. This length depends on the position of the vertices (barycenter vs circumcenter) and the edge-lengths of the original mesh. With intrinsic edge-flips [55], one can build an intrinsic Delaunay triangulation, which guarantees that the circumcentric dual mesh has positive edge-lengths. This leads to high numerical stability and well-conditioned matrices as described in [5]. In this thesis we stick to constructing good triangulations in the first place and will not dive into edge-flips.

Building a Laplacian from a Triangle Mesh

This section will describe how to discretize the differential geometry of manifolds. The goal is to approximate the continuous differential operators on a manifold with discrete operators on a mesh.

In 2D Euclidean space, the Laplacian is given by

$$\Delta f(x, y) = \frac{d^2}{dx^2} f(x, y) + \frac{d^2}{dy^2} f(x, y)$$

In chapter 5 we give the discrete version for Euclidean space, which has a quite simple form on a rectangular grid. When working in a coordinate space with a metric g, the Laplacian cannot be computed directly using the coordinate representation, but must take into account the stretching and deformations induced by the metric. The Laplace-Beltrami operator is the generalization of the Laplacian to Riemannian manifolds.

Definition: Laplace-Beltrami Operator

$$\Delta_g f(x) = \frac{\sum_{i=1}^{n} \frac{\partial}{\partial x_i}\left(\sqrt{|\det(g(x))|}\, g(x)^{-1} \nabla f(x)\right)}{\sqrt{|\det(g(x))|}}$$

For notational convenience, we will always refer to the Laplace-Beltrami operator as simply the Laplacian and write Δ, as the domain and metric is clear from context.

How one would approximate this quantity from a discrete set of points is not clear. The theory of Discrete Exterior Calculus (DEC) is a powerful tool for this purpose. Although the results we will present in the remainder of this chapter are not new, the theory is not widely known and is not covered in standard textbooks on numerical methods or the standard curriculum in computer science, and so we will spend some time giving a rough overview to motivate the formulas we will use.

Crash Course: Exterior Algebra

For a more thorough introduction, see [16]. Exterior Algebra, also known as Grassmann Algebra, generalizes traditional vectors to k-vectors, which form a vector space within each degree k. A 0-vector is identical to a scalar, and a 1-vector corresponds to a traditional vector in the space.

The binary antisymmetric wedge product $\wedge$ combines an n-vector a with a m-vector b to form the $(n+m)$-vector $a \wedge b$. Antisymmetry is characterized by

$$a \wedge b = -b \wedge a$$

The binary change in sign is related to the two possible orientations of the normal vector of a face or the orientation of an edge. In $n = 3$ dimensions, taking the wedge product $v_1 \wedge v_2 = w$ of two 1-vectors yields a 2-vector w (a 'bi-vector', something akin to an oriented plane), the magnitude of which is the same as the magnitude of the traditional cross-product of the vectors: $|w| = |v_1 \times v_2|$, which is equal with the signed area of their spanned parallelogram. Three vectors can be $\wedge$'ed together to form a 3-vector, which can be interpreted as an oriented volume. We will however not need to go beyond 2-vectors when working with 2-manifolds.
The unary 'Hodge star' operator $\star$ maps a k-vector to an $(n-k)$-vector. In a sense it maps to a dual representation of the geometric object, as $\star w$ is $1-$vector representing the normal vector of the parallelogram w. The notion of duality is strengthed by the relation $\star \star w = w$. For $n = 2$, $\star$ maps 0-vectors to 2-vectors and vice-versa, while 1-vectors are mapped to the 1-vectors that are rotated by 90° degrees. The strong connections to the ideas of the dual mesh should be noted. The Exterior Algebra is a language that is well-suited for describing the geometry of manifolds, and is used in the theory of differential forms, which we will cover next.

Crash Course: Exterior Calculus

For more intuition, see [16]. For an in-depth coverage relating also to PDEs, see [10]. In standard multivariable calculus, the differential ds represents an infinitesimal arc length element along a curve. For a 1-dimensional manifold $\mathcal{M}$, the total length can be expressed as $\int_{\mathcal{M}} ds$. Here, ds is derived from the 1-forms dx and dy, which serve as basis elements in the local coordinate system. These 1-forms encapsulate how distances are measured in each coordinate direction.

In exterior calculus, we extend the concept of differentials to higher-degree forms by utilizing the previously defined wedge product. The 2-form $dx \wedge dy$ represents an infinitesimal area element. Consequently, the surface area of a 2-dimensional Euclidean patch $R \subset \mathbb{R}^2$ can be expressed as $\int_R dx \wedge dy$. If R is locally parametrized by a 2-dimensional coordinate system, this integral can be evaluated as a nested integral $\int_{y_1}^{y_2} \int_{x_1(y)}^{x_2(y)} dx\ dy$[§], which computes the area over the specified region.
A function that maps a surface to a continuously varying k-form is called a differential k-form. The unary exterior derivative operator d maps a differential k-form to a differential $(k+1)$-form. For a differential 0-form ϕ (a scalar field), the exterior derivative $\mathrm{d}\phi$ corresponds to the gradient of ϕ in the sense that $\mathrm{d}\phi(\mathbf{v}) = \mathbf{v}^\top \nabla \phi$ for any tangent vector $\mathbf{v}$. This illustrates the close relationship between the exterior derivative and the traditional gradient operator.
The wedge product $\wedge$ and Hodge star $\star$ treat k-vectors and k-forms in a similarly. When applied to differential k-form, the Hodge star maps it to a differential $(n-k)$-form, where the Hodge-star can be though of being applied elementwise at each point in space. The wedge product can wedge together two differential forms by taking the elementwise wedge product of the two forms for each point in space.
k-vectors and k-forms are dual[¶] in the sense that a 1-vector v can be related to a 1-form ν through the unary musical operators $\flat$ ("flat") and $\sharp$ ("sharp"),

$$v^\flat = \nu \iff v = \nu^\sharp$$

Unlike the gradient, which is limited to scalar fields, the exterior derivative provides a building block for various differential operators, such as curl (curl $F = \star \mathrm{d} F^\flat$) and divergence (div $F = \star \mathrm{d} \star F^\flat$). Intuitively, the idea that the curl and divergence are related by the Hodge star should seem sensible, as these are in a sense orthogonal measures.
We have finally arrived to a formula for the Laplace-Beltrami operator, which is the main focus of this entire chapter.

Definition: Laplace-Beltrami Operator, Exterior Calculus Form

The Laplacian of a scalar-valued[a] function f is defined as

$$\Delta f = \star \mathrm{d} \star \mathrm{d} f$$

which is again a scalar function. As is also the case in the Euclidean definition, the Laplacian can also be expressed as the divergence of the gradient,

$$\Delta f = \mathrm{div} \nabla f$$

exressing the divergence with $\star \mathrm{d} \star$ and the gradient operator with d.

[a] This can also be generalized to k-vector-valued functions g, for which the Laplacian takes the form $\Delta g = \star \mathrm{d} \star \mathrm{d} g + \mathrm{d} \star \mathrm{d} \star g$. We will not dig deeper here.

The notation hides a lot of complexity and changes in objects (as good notation should). For newcomers like us, it is instructive to take a look under the hood to convince oneself that what is going is sensible in terms of input/output.

Intuition: Investigating the form of the Laplacian

We will unpack the definition of the Laplacian of f, $\star \mathrm{d} \star \mathrm{d} f$. We start with f, a scalar-function, or, equivalently, a differential 0-form. $\mathrm{d}f$ takes our differential 0-form to a differential 1-form. For ambient ▶

[§] This expression should not be surprising, but is mentioned to relate the abstract object $dx \wedge dy$ to something that feels more familiar.

[¶] For an analogy to better get a sense of this symmetry, think of a ruler: the ruler can be used to measure the length of an object (in cm), but the object can also be used to measure the length of the ruler (in object-length-units)

space dimension n, $\star \mathrm{d} f$ then maps this to a dual representation as a differential $(n-1)$-form, and another application of the exterior derivative then gives $\mathrm{d} \star \mathrm{d} f$, which has again incremented the degree to a differential n-form. A final application of $\star$ then takes us to the dual representation, resulting in a differential $(n-n)$-form, a 0-form, back to a scalar field, as one would expect from the Laplacian. Note that the choice of the ambient space dimension n did not matter, as it was cancelled out in the end. The Laplacian is a scalar field, no matter the dimension of the ambient space, which emphasizes the intrinsic nature of it.

This formula for the Laplacian does not seem like it has brought us any closer to a discrete representation of the Laplacian. However, the exterior derivative d and Hodge star $\star$ operators can be discretized in a way that is consistent with the geometry of the mesh. This is the main result of Discrete Exterior Calculus, which we will now introduce.

Crash Course: Discrete Exterior Calculus

Discrete Exterior Calculus (DEC) is a relatively new (2003, [27], [19]) theory that fully embraces the discrete nature of triangle meshes. It is a discrete version of the exterior calculus, with corresponding discrete versions of the Hodge star and exterior derivative, all of which are directly applicable to manifold triangle meshes. We will give the main results relevant to the probabilistic numerical solver on manifolds, which are the interpretation of the mesh as a manifold and how to build the Laplacian matrix.
In DEC, one uses the "exact input" hypothesis [55], which states that the mesh is not interpreted as an approximation of a manifold, but represents the manifold itself. The mesh forms a partition of the manifold into faces, and each triangular face is assumed to be an isometric embedding of a triangular patch of the manifold. The mesh in its entirety then consists of many patches, each of which are endowed with a local Euclidean metric. A local coordinate system per face can be defined using barycentric[‖] coordinates. The length e_a of an edge (V_i, V_j) between two vertices will thus represent the actual length of the path along the manifold between the two vertices.

[‖]Barycentric coordinates are local coordinates for simplices. A point, straight line, triangle, and tetrahedron are instances of a 0-, 1-, 2-, and 3-simplices respectively. For $|V|$ vertices $(v_1, \ldots, v_{|V|}) \in \mathbb{R}^{n \times |V|}$ in some $k \geq |V|$-dimensional ambient space, local coordinates $\Lambda = (\lambda_1, \ldots, \lambda_{|V|}) \geq 0$ with $\sum_{i=1}^{|V|} \lambda_i = 1$ correspond to point $p(\Lambda) = \sum_{i=1}^{|V|} \lambda_i v_i$ which is a convex combination of the vertices, parametrizing the interior and boundary.

Intuition: Discretizing differential 0- and 1-forms onto a Manifold Triangle Mesh

Scalar fields (differential 0-forms) on the manifold can be discretized onto the mesh by sampling the scalar value of $\alpha(V_i)$ at each vertex V_i. A discretized scalar function is then stored as a $|V|$-dimensional vector $\bar{\alpha}_i = \alpha(V_i)$. This is how we will be representing scalar functions on the manifold, including the later PDE solution (ch. 5 & 6).
We can also represent a discrete differential 1-form β on the manifold. β is dual to a vector field and can almost be thought of as one. An edge (V_a, V_b) is mapped to a parameterized path

$$r_i(t) = (1-t)V_a + {}^a t V_b \in \mathcal{M} \qquad t \in (0,1)$$

The discretized differential 1-form along all edges E_i becomes then a vector $\bar{\beta} \in \mathbb{R}^{|E|}$ with

$$\begin{aligned} \bar{\beta}_i &= \int_{[0,1]} \beta(r_i(t))(\frac{dr_i}{dt}(t))\, dt \\ &= \int_{[0,1]} \beta(r_i(t))(V_b - V_a)\, dt \end{aligned}$$

The orientation of the edge determines the sign of $\bar{\beta}_i$ and is therefore necessary to keep track of. $\bar{\beta}_i$ can be interpreted as a degree of "alignment" between the vector field and the edge.
In summary, differential 0-forms (scalar functions) are sampled onto the 0-dimensional components of the mesh, the vertices. Differential 1-forms ("vector fields") are sampled onto the 1-dimensional components, the edges. Even differential 2-forms (bivector fields) can be sampled onto the 2-dimensional components, the faces. The three are therefore stored, respectively, as real vectors of length $|V|$, $|E|$, and $|F|$.

[a]This linear combination is permitted because we know the straight line (in ambient space coordinates) between two neighboring vertices is the edge, a linear subspace of the manifold triangle mesh.

The discrete exterior calculus analogue of the Laplacian is the DEC Laplacian L, also known as the cotan-Laplacian.

Definition: DEC Laplacian / cotan-Laplacian

In DEC, each of the operators $\star \mathrm{d} \star \mathrm{d}$ become a matrix, and so the discrete Laplacian is a matrix as well. Specifically, the equation turns into

$$L = \star_0^{-1} \mathrm{d}_1 \star_1 \mathrm{d}_0 \in \mathbb{R}^{|V| \times |V|}$$

where $\star_0^{-1}$ is simply the matrix inverse of $\star_0$. This formula is by [5] for 2-manifolds, but can also be derived through a variational formulation as in the finite element method. Formulas exist also for $n-$dimensional manifolds [17]. L has the property that each row-sum is zero, which means that constant functions (vectors with the same entry everywhere) are in the nullspace of L, just like the continuous variant.

The four terms in the definition of the DEC Laplacian will be covered next.

Definition: d_0

The discrete exterior derivative $\mathrm{d}_0 \in \{-1,0,1\}^{|E|\times|V|}$ is a matrix that takes a discrete differential 0-form to a discrete differential 1-form. This is akin to how the standard exterior derivative increases the degree of a differential form. The value of the 1-form at edge is the difference of the values at the two vertices, where the sign depends on the orientation of the edge

$$\mathrm{d}_{0ij} = \mathbf{1}[E_i = (V_j \to \cdot)] - \mathbf{1}[E_i = (\cdot \to V_j)]$$

$\mathbf{1}$ is the indicator function. Each entry is in $\{-1,0,1\}$. Because an edge has only two endpoints, a row in d_0 will have at most two non-zero entries of opposing sign. Each column will have nonzero entries, equal to the number of edges incident to vertex i.

If d_0 is left-multiplied onto a discrete scalar function defined on the vertices, the result is a vector field represented on the edges. This is the discrete analog of the gradient operator.

Definition: $\star_1$

The Hodge-star from 1-forms in (n=2)-dimensional space $\mathbb{R}^2$ is a matrix $\star_1 \in \mathbb{R}^{|E|\times|E|}$ that takes a discretized vector field $\bar{\phi} \in \mathbb{R}^{|E|}$ defined over the edges and returns a discretized vector field over the edges of the dual mesh. $\star_1\bar{\phi}$ represents the discretization of the dual vector field. The matrix is a diagonal matrix that holds on each entry

$$\star_{1ii} = \frac{e_i'}{e_i}$$

with e_i' as the length of the dual edge. $\star_1\bar{\phi}$ can be interpreted as a vector field that flows everywhere orthogonally to the original vector field (rotated 90° everywhere).

Constructing $\star_1$ is the most computationally expensive part of building the Laplacian, as it requires the computation of the dual edge-lengths. These and other related quantities can however all be precomputed and stored efficiently in sparse matrix structures for later use.

Intuition: Calculating the Circumcentric Dual Edge-Lengths

In extrinsic geometry with vertex coordinates given in ambient space, the circumcentric dual mesh can be constructed explicitly, from which the dual edge-lengths can be easily computed. In intrinsic geometry however, we can use the cotangent weights to compute the dual edge-lengths.
Assume vertices A, B, C, D with triangles ABC, BCD with shared edge BC with length $e = |BC|$. The dual edge-length e' is the distance between the two circumcenters of the neighboring triangles. We can decompose this into the sum of the distance from edge BC to each circumcenter. The circumradius R can be calculated as ([55])

$$R = \frac{|AB||BC||CA|}{4\Delta} = \frac{|AB|}{2\sin\theta}$$

with Δ the area of the triangle (given by Heron's formula (3)). The second equality is an alternative expression by the law of sines with $\theta = \angle CAB$. The distance to the segment AB

$$R\cos(\theta) = \frac{|BC|}{2}\frac{\cos\theta}{\sin\theta} = \frac{|BC|}{2}\cot\theta$$

In a symmetric manner we calculate the distance from BC to the circumcenter of BCD as $\frac{|BC|}{2}\cot\gamma$ for opposing angle $\gamma = \angle CDB$, yielding final dual edge-length

$$e' = \frac{|BC|}{2}(\cot\theta + \cot\gamma)$$

Finally, this results in

$$\star_{1ii} = \frac{e_i'}{e_i} = \frac{1}{2}(\cot\theta + \cot\gamma)$$

for appropriate values of θ and γ related to edge at index i. At a boundary edge, one of the terms in the sum will be defined as zero.
This formula is what gives rise to the name "cotan-Laplacian".

With the dual edges in place, we now move to the final two matrices in the DEC Laplacian.

Definition: d_1

The exterior derivative matrix $\mathrm{d}_1 \in \{-1,0,1\}^{|F|\times|E|}$ is a matrix that takes a discretized differential 1-form on edges and maps it to a discrete differential 2-form on faces. Similarly to d_0, the value of the 2-form at face F_i is sum of the values at the edges that make up the face, with the sign depending on the orientation of the face.

$$\begin{aligned}\mathrm{d}_{1ij} = \mathbf{1}[E_j = (V_a \to V_b) \in F_i = (V_a, V_b, V_c)] \\ - \mathbf{1}[E_j = (V_b \to V_a) \in F_i = (V_a, V_b, V_c)]\end{aligned}$$

Keep in mind that even permutations of a face are considered the same face, so $(V_a, V_b, V_c) = (V_c, V_a, V_b) = (V_b, V_c, V_a)$. Rows in d_1 are related to faces and will have three non-zero entries in $\{-1,0,1\}$, and each column, related to an edge, will have one or two non-zero entries (one if it is a boundary edge).

Definition: $\star_0$

The Hodge-star from 0-forms in (n=2)-dimensional

space $\mathbb{R}^2$ is a matrix $\star_0 \in \mathbb{R}_+^{|F|\times|V|}$ that takes a discretized differential 0-form and returns a discretized 2-form over the faces of the dual mesh. The matrix is a diagonal matrix that holds in each entry

$$\star_{0ii} = A_i$$

with A_i the area of the associated dual face of the vertex i. In the DEC Laplacian, $\star_0$ is inverted, but since the matrix is diagonal, this is a simple operation.

Intuition: Dual Face Areas

The discrete Hodge-star is still an open question and a topic of research ([49]). While the discrete exterior derivative is in a sense *exact* on the manifold ([16]), the discrete Hodge-stars $\star_0$, $\star_1$ are only an approximation, which stems from the ambiguity of the two possible choices of a dual mesh.
The barycentric dual area of a vertex is simply $\frac{1}{3}$ the sum of the areas of the adjacent triangles, while the circumcentric dual takes a bit more work. However, in practice ([19]) on a Delaunay triangulation, the circumcentric dual area is well approximated by the barycentric dual area, and so this is also the method used in this thesis.

With all the preceding definitions in place, the DEC Laplacian can now be obtained as the matrix $L \in \mathbb{R}^{|V|\times|V|}$.

Bringing it together: Computing the Discrete Laplacian of a Simple Mesh

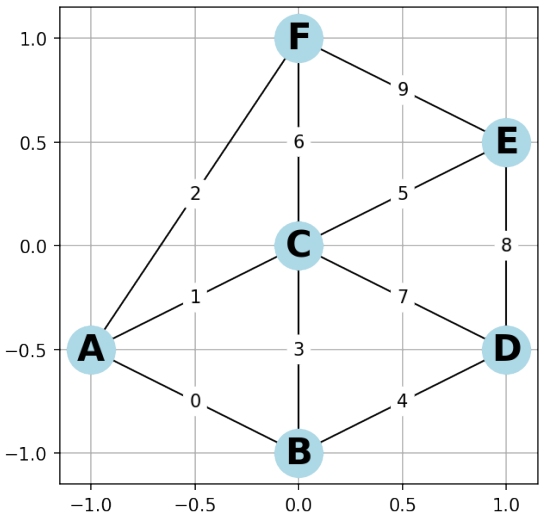


Figure 8: The mesh we will be working with. Edge index is labeled on the edge

In this example, we work with the small manifold mesh in fig 8. It is planar, but this is just for convenience of visualization. We have a total of 6 vertices and 10 edges. The vertices are ordered alphabetically as $V = (A, B, C, D, E, F)$. We choose to both direct and order edges alphabetically, so that we have $E = \Big((A \to B), (A \to C), (A \to F)\ (B \to D), (C \to D), \ldots, (E \to F)\Big)$. The order is not important, as long as it is consistent.

$\mathrm{d}_0 \in \{-1, 0, 1\}^{10\times 6}$ is computed by the formula given earlier, and the result is

$$\mathrm{d}_0 = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

$\mathrm{d}_1 \in \{-1, 0, 1\}^{5\times 10}$ is computed by the formula given earlier, and the result is

$$\mathrm{d}_1 = \begin{bmatrix} 1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Both d_0 and d_1 depend only on the topology of the mesh. For $\star_0$ and $\star_1$, we will need to consider the actual edge-lengths and face areas. The dual areas turn out to be

$$\star_0 = \mathrm{diag}(\begin{pmatrix} 2/6 & 2/6 & 5/6 & 2/6 & 2/6 & 2/6 \end{pmatrix}^\top)$$

while the ratio of the dual edge to the primal edge-length is $\star_1$,

$$\star_1 = \mathrm{diag}(\frac{1}{8}\begin{pmatrix} 2 & 8 & -2 & 6 & 2 & 4 & 4 & 10 & 3 & 2 \end{pmatrix}^\top)$$

The negative entry for edge 2 is due to the circumcentric dual, see figure 7. With the use of aforementioned edge-flips [55], the mesh can be locally rewired in a way that will ensure that all dual edge-lengths are positive, but we will not do this.
Finally, multiplying these matrices together yields the Laplacian matrix $L = \star_0^{-1}\mathrm{d}_1 \star_1 \mathrm{d}_0 =$

$$\begin{bmatrix} 3 & -0.75 & -3 & 0 & 0 & 0.75 \\ -0.75 & 3.75 & -2.25 & -0.75 & 0 & 0 \\ -1.2 & -0.9 & 4.8 & -0.6 & -0.6 & -1.5 \\ 0 & -0.75 & -1.5 & 3.375 & -1.125 & 0 \\ 0 & 0 & -1.5 & -1.125 & 3.375 & -0.75 \\ 0.75 & 0 & -3.75 & 0 & -0.75 & 3.75 \end{bmatrix}$$

L can be used immediately as a drop-in replacement of the Laplace-operator Δ in the Method of Lines mentioned in chapter 5 which this is exactly how we will use it in the later chapters. There is the issue of how it handles boundaries - one can encode Neumann boundary conditions which is covered in high detail in [16] - in our experiments, we will stick to the simpler Dirichlet boundary conditions, both of which are defined in chapter 5.

4 Building an Intrinsic Triangulation from a Metric

We will now describe a way to build an intrinsic triangulation of a Riemannian 2-Manifold $(\mathcal{M}, g)$, which enables the use of tools from Discrete Exterior Calculus, the cotan-Laplacian in which we are most interested in here. An intrinsic triangulation represents the intrinsic geometry - not an embedding of $\mathcal{M}$ into some ambient space. This is emphasized by the fact that we will be working with vertex coordinates in 2 dimensions, and will be interested in the edge-lengths under g, not the Euclidean distances between the vertices.

Computing Edge Lengths through the Coordinate Space

Let the metric $g : C \to \mathbb{R}^{2\times 2}$. We will be triangulating the local-coordinate domain $C \subset \mathbb{R}^2$ of the metric g, but using g to measure the properties (lengths, areas) of the triangles. The triangulation will be an intrinsic triangulation of the manifold by encoding the metric in the edges and therefore having no notion of ambient space.
In the introduction we have given the general formula for computing lengths of paths under the metric g. We will here consider the edges of the mesh as paths, and compute their length using this same formula.
For an edge $E = (A, B)$, we will parametrize it as

$$r_i(t) = (1-t)A + tB$$

for $t \in [0, 1]$. The measured length along the edge from A to B, $e_i = m(A, B)$ is then computed as

$$\begin{aligned} m(A,B) &:= \int_0^1 \sqrt{\nabla r_i(t)^\top g(r_i(t)) \nabla r_i(t)}\, dt \\ &= \int_0^1 \sqrt{(B-A)^\top g(r_i(t))(B-A)}\, dt \end{aligned}$$

By abbreviating the integrand as

$$f(t) = \sqrt{(B-A)^\top g(r_i(t))(B-A)}$$

we can get a simple formula for a numerical approximation using the trapezoidal rule:

$$\begin{aligned} m(A,B) &= \int_0^1 f(t)\, dt \\ &\approx \frac{1}{n+1}\left(\frac{f(0)+f(1)}{2} + \sum_{t=1}^{n-1} f\left(\frac{t}{n+1}\right)\right) \end{aligned}$$

The trapezoidal rule is parametrized by the number of steps n to take. The more steps, the more accurate the approximation.

What's the Catch?

If we naïvely start applying this to a mesh overlaid onto the local coordinate system C, we might think that all is good and that we will have built a nice triangulation. However, if our mesh is not dense enough with respect to the rate of change of g, we might end up with *Non-Planar* triangles.

Definition: Non-Planarity

A key property of intrinsic triangulations are that each triangle can be locally embedded into a nonempty subset of $\mathbb{R}^2$. This is enforced by the strict triangle inequality, which states that any two edges together are longer than the third edge, and that all edges are of nonzero length. When a triangle does not satisfy the strict triangle inequality but has positive lengths, we will here refer to it as non-planar, because it cannot be embedded into the plane. The ratio of the longest edge to the shortest edge in a planar triangle will always be in the interval $(1, 2)$.

So? Well, for starters, Heron's formula (3) for the area of a triangle will yield imaginary areas for non-planar triangles[**]. The operators of discrete exterior calculus will thus not be well-defined, and we will not be able to compute the Laplacian.
In Euclidean space, all measurements along straight lines between three points (A, B, C) in will result in planar triangles[††]. However, when we are measuring with the metric g along straight line $(A \to B)$ in the C, we will generally not be taking the shortest path from A to B on the manifold. If the measured length $||AB||_g$ satisfies

$$||AB||_g > ||AC||_g + ||CB||_g$$

(which is *the* violation of the triangle inequality), $A \to C \to B$ will be a shorter path. This is the essence of non-planarity, which motivates the need for the triangle inequality.
To deal with non-planar triangles, one can subdivide them. There are many subdivision schemes of triangles, such as red-green refinement [43], butterfly subdivision [20], or the loop scheme [39]. We will use longest-edge bisection[2], which is simple to implement. When we repeatedly halve the longest edge of a non-planar triangle, the resulting triangles will eventually become planar, because any triangle can be made planar by shortening the longest edge enough.

Intuition: Fan Model

To better understand the phenomenon of non-planarity, we suggest a mental model of the geometry encoded by the triangle in fig 9 as part of a folded fan. The two shortest edges $(1, 1.5)$ form the straight edges, while the longest one is scrunched up between them.

▶

[**]This is how the author bumped into the concept in the first place and had to think about all of this.

[††]If (A, B, C) are collinear, the area of the triangle will be zero, but the standard triangle inequality is not violated.

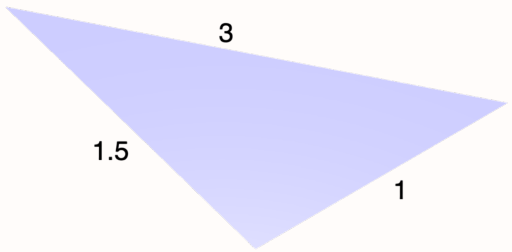


Figure 9: A non-planar triangle with edge lengths shown. Do not get fooled by the fact that it is being drawn in a plane - lengths are not to scale. The plane is merely the coordinate representation of the surface.

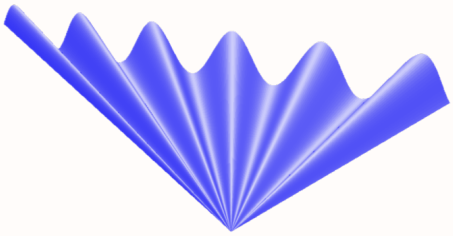


Figure 10: A Euclidean embedding of the non-planar triangle requires curved geometry, as the triangle inequality is not satisfied. This figure shows one interpretation of a manifold that would result in the non-planar measurements.

Intuition: Extrinsic vs Intrinsic Triangulation of the Fan model

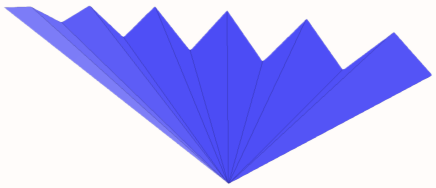


Figure 11: An extrinsic 12-triangle isometrically embedded approximation of the fan model. The mesh stores only vertex positions. In ambient space, each vertex "sits" exactly on the manifold, while the planes spanned inbetween are linear approximations and will generally move away from the manifold.

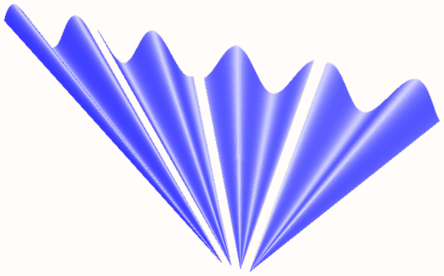


Figure 12: An intrinsic 4-triangle representation of the fan model. The mesh stores edge-lengths only. We render the triangulation onto the Euclidean embedding (fig 10) to highlight the fact that edge-lengths are measured along the manifold, not in straight lines through ambient space. The manifold has been cut along the 3 interior edges of a 4-triangle mesh to show how it is implicitly partitioned.

We will now give our algorithm that builds the intrinsic triangulation. The algorithm is split into two phases: setup and refinement. The setup phase will compute the edge-lengths of the initial mesh and mark triangles that are non-planar, have edges that are too long or too short, or have an area that is too large. The refinement phase will then iteratively split the longest edge of the marked triangles and inspect the new triangles for the same properties. The algorithm will terminate after a fixed number of iterations or when there are no more marked triangles.

Algorithm: Setup Intrinsic Triangulation

Input:

M : Initial mesh that partitions the coordinate space C.

g : Metric, mapping from any point in C to 2x2 matrix.

ρ : Desired maximal edge ratio in any triangle, $1 < \rho < 2$

$A_{\max}$: Desired maximal area of any triangle.

Step 1: Compute edge-lengths.

for each edge $E_i = (A, B)$ **in** M:

Numerically compute the edge-length as $e_i = m(A, B)$

Step 2: Inspect triangles.

Initialize **T** as the empty set

for each triangle T **in** M:

Add T to **T** if
the ratio $e_{\text{long}}/e_{\text{short}}$ exceeds ρ or
the area exceeds $A_{\max}$

Output:

$\mathbf{\Delta}_0$: Mesh M with computed edge-lengths.

T : Set of marked triangles.

For the subsequent refinement phase, it will be useful to have a figure to refer to that explains the naming.

Definition: Local Triangle Naming scheme

For a given triangle, we refer to edges as $E_{\text{long}}, E_{\text{medium}}, E_{\text{short}}$ with the property $e_{\text{long}} \geq e_{\text{medium}} \geq e_{\text{short}}$.

The vertex opposing E_{long} edge is **C**orner, the vertex on the shortest edge **S**hort, and the remaining vertex on the medium-length edge **M**edium. The vertex across E_{long} in a potential adjacent triangle is **O**pposite.
In the figure below, let T=SMC be the reference triangle. We have E_{long}=SM, E_{medium}=CM, and E_{short}=CS.

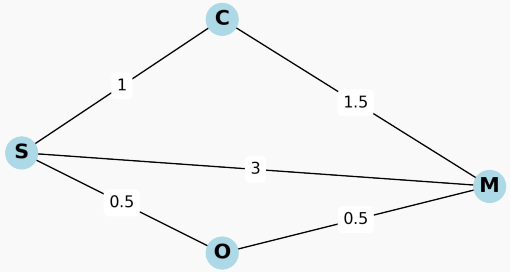


Figure 13: The reference triangle patch illustrating the naming scheme of the upcoming algorithms. This figure additionally serves as an example output $\mathbf{\Delta}_0$ of the setup phase. Both triangles would be in the output $\mathbf{T}$ due to their non-planarity.

Algorithm: Refine Intrinsic Triangulation

Input:

$\mathbf{\Delta}_0$: Initial intrinsic triangulation with edge-lengths.

$\mathbf{T}$: Set of marked triangles.

g, ρ, $A_{\max}$: As described in setup.

$\mathbf{i}_{max}$: Maximal number of subdivisions.

$\mathbf{i} \leftarrow 0$

while $\mathbf{T}$ is not empty **and** $\mathbf{i} < \mathbf{i}_{max}$:

$\mathbf{i} \leftarrow \mathbf{i} + 1$

Select the marked triangle $T = SMC \in \mathbf{T}$ with the largest perimeter and remove T from $\mathbf{T}$.

Step 1: Split longest edge.

Find Euclidean midpoint of longest edge E_{long}, define it as new vertex at N.

Remove previous edge MS, insert new edges NS, MN.

With metric g, compute $|NS| = m(N,S)$ and $|MN| = e_{\text{long}} - |NS|$

Step 2: Bisect faces.

Split T and the adjacent triangular face T'=SOM opposite E_{long} to get triangles SON, NOM, NMC, NCS.

Remove T' from $\mathbf{T}$.

Compute new edge-lengths CN and NO: $|CN| = m(C,N)$, $|NO| = m(N,O)$.

Step 3: Inspect new triangles.

Evaluate the area and aspect ratio of SON, NOM, NMC, and NCS.

Based on ρ, and $A_{\max}$, mark triangles for refinement by adding them to $\mathbf{T}$.

Output:

$\mathbf{\Delta}$: Refined Intrinsic Triangulation, everywhere satisyfing the triangle inequality[a]

[a] This is not the case if the max iterations have been set too low.

During subdivision, we are constructing an intrinsic partition of the manifold into triangles. A triangle might require multiple bisections before it becomes planar. Note that, before any splitting, we didn't have just a bad approximation - we had *no* approximation. Splitting was necessary to work with the geometry.

Results

Triangulating Gaussian Bump

We define our own metric similarly to the hemisphere metric example in section 3. The map f from coordinate space $C = (-1, 1) \times (-1, 1)$ is given as

$$f(\begin{pmatrix} x \\ y \end{pmatrix}) =$$

$$\exp\left(-\left(\begin{pmatrix} x & y \end{pmatrix}\begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.35 \end{bmatrix}^{-1}\begin{pmatrix} x \\ y \end{pmatrix}\right)\right)$$

Figure 14 shows the initial coarse mesh. The algorithm is run for 500 steps with $A_{\max} = 1/8$ and $\rho = 1.2$. The output is shown in figures 15. The mesh is now more dense around the peak of the bump, where the curvature is higher. The mesh is not an embedding of the manifold, but an intrinsic triangulation. An embedding using f is shown in fig 16, giving an extrinsic representation. We might not generally have access to f, and this figure is for illustration purposes only. Notice that triangles that appear very thin and stretched in the intrinsic triangulation are not so after being embedded isometrically.

Triangulating subset of the Hyperbolic Plane

We showcase the algorithm on the Poincaré disc model of $\mathbb{H}^2$ and compare with a hyperbolic tiling. As discussed in an earlier section, we cannot triangulate an unbounded domain, so we truncate the unit disc and look only within the disc of radius 0.97. Algorithm run for 500 steps with $A_{\max} = \infty$ and $\rho = 1.9$. Setting $A_{\max} = \infty$ leads to the parameter being ignored, while a high setting of ρ makes the refinement phase focus just on making planar triangles, which means that all initial iterations are likely to focus on non-planar triangles only.
Figure 17 shows the input mesh $\mathbf{\Delta}_0$, which covers the 0.97-disc somewhat uniformly. Three triangles on the way to the border have their edge-lengths marked. This is the length under the hyperbolic metric. The outermost triangle does not satisfy the triangle inequality in this initial mesh, as $1.34 + 2.04 < 6.08$. Figure 18 shows the output of the algorithm, an intrinsic triangulation of a subset of $\mathbb{H}^2$. The high triangle density at the edges highlights the hyperbolic expansion of space, which necessitates triangle subdivision. All triangles are congruent. The triangles in our tiling are not congruent and will depend on the initial triangulation, but $A_{\max}$ ensures that we make splits around the edge proportional to the area. A common visualization of the hyperbolic plane is given in fig 19, in which all triangles are congruent. This visualization bears resemblance to the one obtained in figure 18 and illustrates the hyperbolic expansion of space which necessitates triangle subdivision.

Potential Improvements

The metric tensor will be evaluated in the same vertex many times if the edge is split multiple times. One

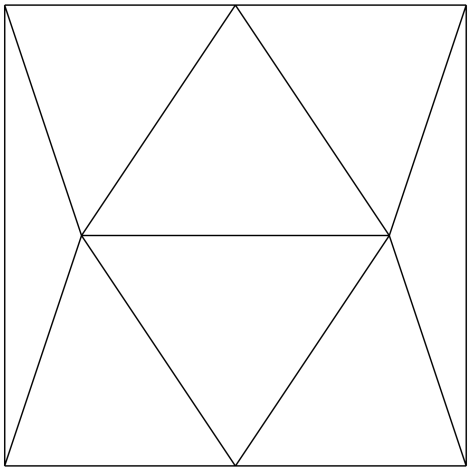


Figure 14: Input mesh.

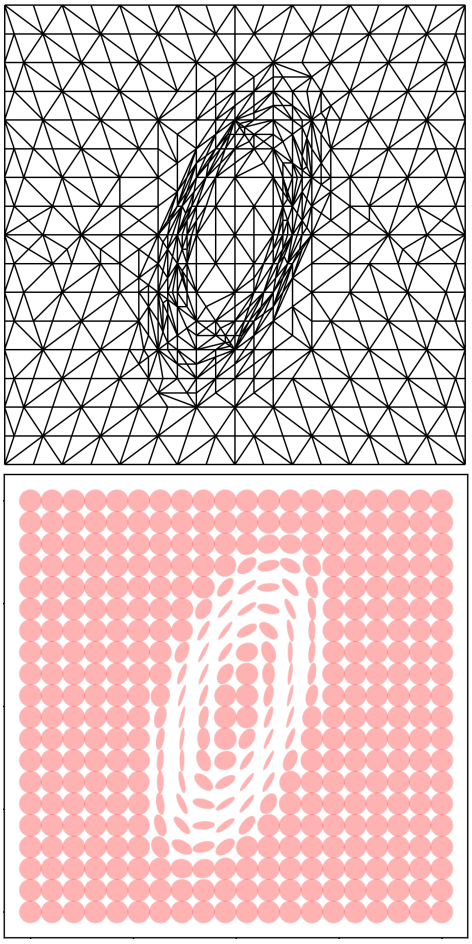


Figure 15: Output mesh and Tissot's Indicatrices.

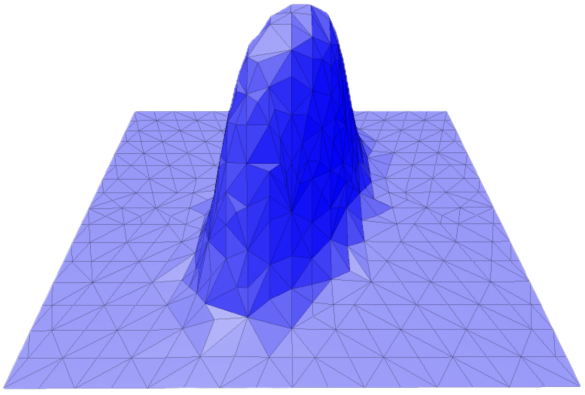


Figure 16: Mapping the refined mesh through the embedding function f.

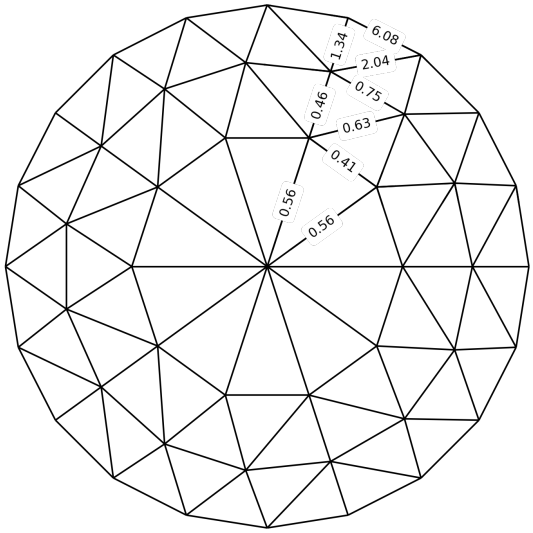


Figure 17: Input mesh.

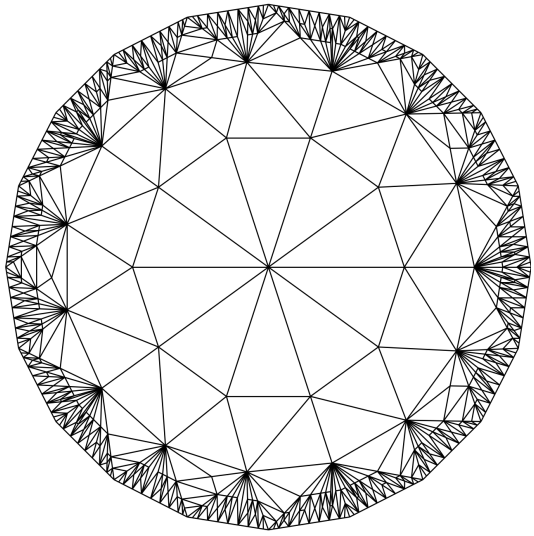


Figure 18: Output mesh

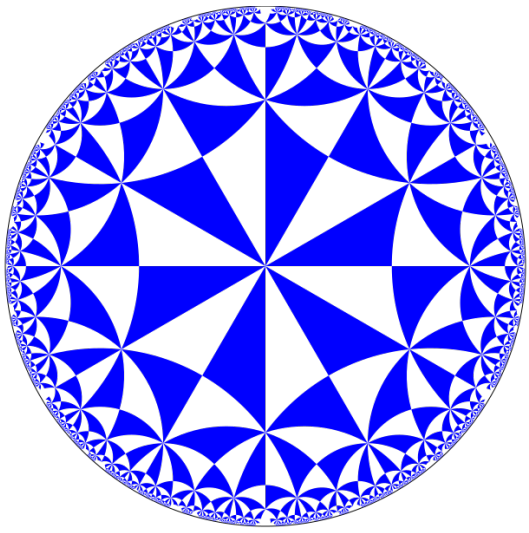


Figure 19: Tiling of the hyperbolic plane from [29].

could cache the metric tensor at each vertex to avoid redundant computations.

One could, assuming the fan-model, also compute the distance from the midpoint of the longest edge to the opposing corner as $(1 + 1.5)/2 = 1.25$. Under that assumption it will yield exact distances. This could be done as the last split in each triangle so the approximation error does not propagate.
An alternative approximation is that the metric will changes linearly between the two points of an edge. In this case the metric tensor must be computed only at the endpoints. With caching, one would have to compute it only once per vertex.

Additionally, instead of bisecting the longest edge at its Euclidean midpoint (in coordinate space), one can run an incremental search along it to determine the midpoint in coordinate space under the metric g. This which would be a more reasonable choice for the new vertex and might save a few subdivisions. It can even be done at the same cost of the current algorithm, since, once the midpoint is found, one will know that the lengths of the two constituent edges are equal and sum to the original length.

After the refinement phase, one can run edge-flipping ([55], [5]) to further improve the quality. This is an operation applicable to intrinsic triangulations that swaps diagonal edges of a quadrilateral if it improves the quality of the triangulation.

Alternatively to edge-flipping, one could use a different subdivision scheme, such as red-green refinement. This is more complex, but might yield better results as it is free to place new vertices in the interior of the triangle, not just on the edges.

Generally, the greedy subdivision scheme could be improved upon or replaced. If one zooms in on the picture in figure 16, one notices a so-called "rat-nest", which is a vertex with many incident edges. This is a common problem in greedy subdivision schemes, and will generally lead to elongated triangles. The bisection method can never get rid of a rat-nest after it has made one.

Experiment with the Laplacian

We will perform an experiment with the Laplacian of this mesh in chapter 7, after having defined the probabilistic numerical solver in the following two chapters.

5 Partial Differential Equations and the Method of Lines

This section covers the relevant background knowledge about ODEs and PDEs, representing them in a canonical state-space representation, and how to discretize PDEs to ODEs using the Method of Lines. To relate to the Laplacian matrix from chapter 3, we also briefly introduce Finite Difference Methods, used in Euclidean settings to approximate spatial derivatives.

Definition: Ordinary Differential Equation, First Order

An ordinary differential equation (ODE) is a differential equation where the solution $u : \mathbb{R} \to \mathbb{R}^n$ is defined implicitly by relating it to its own derivatives through $f : \mathbb{R}^n \to \mathbb{R}$:

$$\frac{d}{dt}u(t) = f(u(t))$$

u is a function of only one scalar variable (usually interpreted as time t), and f is a function depending on u. If the ODE is vector valued, one also refers to solving it as integrating a vector field f.

Authors will often omit the explicit dependence on t and write $\frac{d}{dt}u = f(u)$[‡‡]. It is implied that this relation must hold for all times t.

Definition: Order of ODE

The order of an ODE is the degree of the highest derivative of u involved in the equation. An n'th ODE can be written as

$$\frac{d^n}{dt^n}u(t) = f(u(t), u^{(1)}(t), \dots, u^{(n-1)}(t))$$

The notation $u^{(i)}$ is short-hand for the i-th derivative. We can always rewrite an n'th order ODE into a system of n first-order ODEs, which is useful as one then need not engineer a new solution method for each choice of n.

Intuition: Converting a nth order ODE to system of first-order ODEs

Suppose we have the nth order ODE

$$\frac{d^n}{dt^n}u = f(u, u^{(1)}, \dots, u^{(n-1)}) \tag{4}$$

By defining $n-1$ auxiliary functions v_i for $i \in \{1, \dots, n-1\}$

$$\begin{aligned}\frac{d}{dt}u &= v_1\\ \frac{d}{dt}v_i &= v_{i+1}\end{aligned}$$

We may equivalently express 4 as

$$\begin{aligned}\frac{d}{dt}u &= v_1\\ \frac{d}{dt}v_1 &= v_2\\ &\vdots\\ \frac{d}{dt}v_{n-2} &= v_{n-1}\\ \frac{d}{dt}v_{n-1} &= f(u, v_1, \dots, v_{n-1})\end{aligned}$$

which can be interpreted as a vector valued first order ODE with solution $\vec{u}$

$$\frac{d}{dt}\vec{u} = \frac{d}{dt}\begin{pmatrix} u\\ v_1\\ \vdots\\ v_{n-1}\end{pmatrix} = \begin{pmatrix} v_1\\ v_2\\ \vdots\\ f(u, v_1, \dots, v_{n-1})\end{pmatrix} \tag{5}$$

This is called a state-space representation of the ODE, where the state $\vec{u}(t)$ is a vector containing the solution and its derivatives at time t. To get the solution $u(t)$, one can simply discard the remainder of the state.

If f is a linear function of u and its derivatives, the ODE is linear. The right-hand side of 5 can then expressed as a matrix-vector product.

Intuition: 2nd order *Linear* ODE to first-order ODEs

We inspect the scalar second order linear ODE[a] with constants $k > 0$ and $l > 0$

$$\frac{d^2}{dt^2}u = -ku - l\frac{d}{dt}u(t)$$

The corresponding form of 5 will turn out as

$$\frac{d}{dt}\vec{u} = \frac{d}{dt}\begin{pmatrix} u\\ v\end{pmatrix} = \begin{pmatrix} v\\ -ku - lv\end{pmatrix} \tag{6}$$

$$\iff \frac{d}{dt}\begin{pmatrix} u\\ v\end{pmatrix} = \begin{bmatrix} 0 & 1\\ -k & -l\end{bmatrix}\begin{pmatrix} u\\ v\end{pmatrix} \tag{7}$$

$$\iff \frac{d}{dt}\vec{u} = A\vec{u} \tag{8}$$

The matrix A captures the system dynamics. The '1' in the off-diagonal encodes the integral-derivative relationship between u and v.

[a] This is the dampened harmonic oscillator (a one-dimensional spring model). k represents the spring stiffness constant from Hooke's law, and l is a damping coefficient that brings the system to a halt over time. With this physical perspective, one can mentally rename $u(t)$, $u'(t)$, and $u''(t)$ respectively to the more descriptive names $p(t)$ (position), $v(t)$ (velocity), and $a(t)$ (acceleration).

[‡‡] This is abbreviation is similar to how one might write the equation $h(a) = f(a)+3\cdot g(a)$ as $h(a) = (f+3g)(a)$ or $h = f+3g$, treating functions as members of a vector space.

Definition: Initial Value Problem for ODE

Given the initial state $\vec{u}(0)$ and system dynamics

$$\frac{d}{dt}\vec{u}(t) = A\vec{u}(t)$$

simulate the state evolution until final finite time T. Return either $\vec{u}(T)$ or the full trajectory $\vec{u}(t)$ for $t \in [0, T]$.

For Linear ODEs, closed form solutions to the initial value problem exist. They can be used as a reference to verify numeric solvers.

Intuition: Analytical solutions of First Order Linear ODEs

The linear scalar ODE with scalar coefficient a

$$\frac{d}{dt}u(t) = au(t)$$

has solution

$$u(t) = \mathrm{e}^{at}u(0)$$

This pattern generalizes to n-dimensional ODEs with matrix coefficient $A \in \mathbb{R}^{n\times n}$

$$\frac{d}{dt}\vec{u}(t) = A\vec{u}(t)$$

which has the solution

$$\vec{u}(t) = \mathrm{e}^{At}\vec{u}(0) \tag{9}$$

e^{At} is the matrix exponential, and is defined only for square matrices. $\mathrm{e}^{At} = \sum_{n=0}^{\infty} \frac{t^n}{n!}A^n$.

Rarely can IVP problems be solved analytically, hence the need for numerical methods.

Definition: Numerical solution of ODE

A numerical solution of an ODE is an approximation of $u(t)$ at finite timesteps $t \in \mathbb{T} = (0, t_1, t_2, \ldots, T)$.

We are now ready to move on to Partial Differential Equations.

Definition: Partial Differential Equation

A partial differential equation (PDE) differs from an ODEs in that the solution u can depend continuously on $d = |\{t, x_1, \ldots, x_{d-1}\}| = |\{t, \mathbf{x}\}|$ variables, $u : \mathbb{R}^d \to \mathbb{R}^n$. The total derivative $\frac{d}{dt}$ is replaced with the partial derivative $\frac{\partial}{\partial t}$ and vector field $f : \mathbb{R}^n \to \mathbb{R}^d$ can additionally depend on the partial derivatives.

$$\frac{\partial^n}{\partial t^n}u(t,\mathbf{x}) = f\Big(u(t,\mathbf{x}), \left\{\frac{\partial^q}{\partial t^q}u(t,\mathbf{x})\right\}_{q=1}^{n-1}, \left\{\frac{\partial^q}{\partial x^q}u(t,\mathbf{x}) \;:\; x \in \mathbf{x}\right\}_{q=1}^{m}\Big)$$

The order of a PDE is the highest partial derivative involved, $\max(n, m)$. In this thesis we generally work with PDEs over 2-dimensional space, so we have $\mathbf{x} \in \mathbb{R}^2$ and time $t \in (0, T)$ as $u(t, \mathbf{x})$.

PDEs can make the time-evolution of a system depend on spatial properties such as the curvature, captured by the Laplacian.

Often the domain $\mathcal{M}$ has a border $\partial\mathcal{M}$ which the solution u will have to interact with. The spatial partial derivatives in f will be well-defined everywhere on the interior of $\mathcal{M}$ but not on $\partial\mathcal{M}$, and we will have to prescribe a value to make the problem well-posed. We will cover two types of boundary conditions (BCs), the Dirichlet BC and Neumann BC.

Definition: Dirichlet Boundary Conditions

Dirichlet BCs prescribe the value of u on the border with boundary values $g : \partial\mathcal{M} \to \mathbb{R}$:

$$\begin{aligned} \frac{\partial}{\partial t}u(t,x) &= f(\cdot,\ldots,\cdot) && x \in \mathcal{M} \\ u(t,x) &= g(x) && x \in \partial\mathcal{M} \end{aligned}$$

Dirichlet BCs amount to "pinning" the solution to a specific value at the boundary.

Definition: Neumann Boundary Conditions

Neumann BCs prescribe the directional spatial derivative in the direction of the outer normal $n(x)$ of the boundary. Let $d : \partial\mathcal{M} \to \mathbb{R}$:

$$\begin{aligned} \frac{\partial}{\partial t}u(t,x) &= f(\cdot,\ldots,\cdot) && x \in \mathcal{M} \\ \nabla_x u(t,x)^\top n(x) &= d(x) && x \in \partial\mathcal{M} \end{aligned}$$

Neumann BCs are a bit less intuitive, but allow one to set the "slope" at the boundary. We will only be considering Dirichlet BCs in this thesis.

A simple instance of a PDE is the heat equation, which we will return to as a running example. It is defined with the Laplace operator, a differential operator involving partial derivatives. In the scope of this thesis, the Laplacian is taken only with respect to spatial coordinates $\mathbf{x}$.

Intuition: Heat Equation on $(0, 1) \subset \mathbb{R}$

The heat equation describes the evolution of temperature $u(t, \mathbf{x})$ at a point $\mathbf{x}$ and time t.

$$\frac{\partial}{\partial t}u = -\Delta u \tag{10}$$

$-\Delta u(t, x)$ becomes large when $u(t, \mathbf{x})$ is generally colder than the average temperature in the neighborhood of $\mathbf{x}$ and vice versa. The heat equation then states that each point should move towards the average local temperate. When $t \to \infty$, each point in $u(\mathbf{x}, t)$ will become the average of its neighboring points.

For one dimension, the heat equation becomes

$$\frac{\partial}{\partial t}u(t,x) = -\frac{\partial^2}{\partial x^2}u(t,x)$$

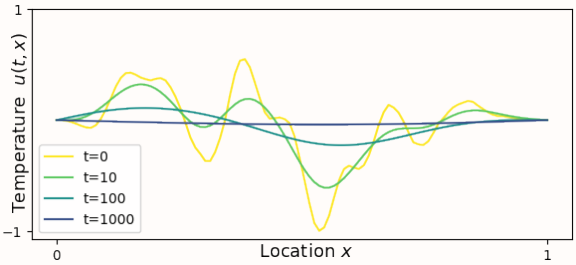


Figure 20: Numerical solution of the heat equation on $(0,1) \subset \mathbb{R}$, with Dirichlet BCs $u(0,0) = u(0,1) = 0$. The solution is shown at four selected timesteps. As time increases, the temperature distribution gets smoother, until it reaches 0. Since the temperature moves towards the average, and the boundary is fixed at zero, the temperature must go to zero.

Method of Lines

The challenge with ODEs was the continuous parameter t, which we overcame by discretizing across the temporal dimension. However, our PDEs now additionally depend on continuous spatial parameters $\mathbf{x}$. The Method of Lines (MOL) [52] works by converting the PDE to a system of ODEs, for which existing ODE solvers can be used. To convert to a system of ODEs, the only continuous parameter u may depend on is t. By discretizing space into finite set of points $\mathbb{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, the continuous spatial input has been reduced to discrete indexing into n-vector $\mathbf{u}(t)$

$$\mathbf{u}(t) = \big(u(t,\mathbf{x}_1), \ldots, u(t,\mathbf{x}_n)\big)^\top$$

containing the solution at each of the collocation points. However, having discretized space into a countable set of points, the differential operators acting on the spatial dimension are no longer defined.

Definition: Finite Difference Methods

Finite Differences (FD) methods approximate differential operators from regularly spaced finite points $\mathbb{X}$ of space. Standard differential operators are defined through limits, but with a finite number of spatial points, the limit must be approximated. This yields finite difference operators, which can be applied to a function evaluated at the same points $\mathbb{X}$ to approximate its spatial derivative in thesespoints.
Like continuous differential operators, FD operators are only guaranteed to be defined on the interior of the domain. Supplying boundary conditions is therefore still necessary.

Definition: Collocation points

In finite difference methods, collocation points are the spatial points where the finite difference approximations of derivatives are computed.

For linear differential operators the finite difference method will also be linear and can thus be written as a matrix. The exact form of the FD method depends on the discretization. In Euclidean space, rectangular grids yield simple formulas depending on the grid-spacing. On curved manifolds however, the grid-method breaks down. The methods from Discrete Exterior Calculus (ch. 3) give formulas for FD-like operators on discretized manifolds (triangle meshes).

Intuition: 1D Euclidean Finite Differences, Laplacian

In one dimension, from a finite set of collocation points

$$\mathbb{X} = (x_1 < \cdots < x_n) \subset (0,1)$$

spaced h apart, one can approximate the second derivative of $u : (0,1) \to \mathbb{R}$ on the interior x_i with the formula[36]

$$\frac{\partial^2}{\partial x^2}u(x_i) \approx \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{2h}$$

This leads to matrix (with Dirichlet BCs = 0)

$$L\mathbf{u} = \frac{1}{h^2}\begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -2 & 1 \\ 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix}\mathbf{u}$$

The first and final empty rows encode the Dirichlet BC.

The discrete differential operator encodes the spatial structure of the domain, and can be used to convert the PDE to a system of ODEs. Note that, *once the differential operator is discretized, there is no reference to the underlying metric or topological structure. This information is encoded exclusively in the differential operator,* which means that the Method of Lines can be applied to any PDE on any domain, provided the differential operator can be discretized. We will give a formula for the Laplacian on a triangle mesh in chapter 3.

Intuition: Method of Lines applied to the Heat Equation

The heat equation (10) makes use of the linear differential operator Δ, the Laplacian. By discretizing into spatial points $n = |\mathbb{X}|$ and deriving a suitable matrix $L \in \mathbb{R}^{n \times n}$ estimating Δ, we get the system ▶

of n ODEs

$$i \in \{1, .., n\}: \quad \frac{d}{dt} u(t)_i = -L_i u(t) \in \mathbb{R}$$

where L_i the ith row of L. We can write it more eloquently in vector-form by stacking all the spatial points to get

$$\frac{d}{dt} u(t) = -Lu(t) \in \mathbb{R}^n \tag{11}$$

The solution in figure 20 was obtained using this exact method, for $\mathbb{X} = \{0, \frac{1}{100}, \frac{2}{100}, \ldots, \frac{99}{100}, 1\}$ and the FD Laplacian matrix from the previous box.

Intuition: Method of Lines applied to the Wave Equation

The wave equation describes the evolution of the height of a wave $u(t, \mathbf{x})$ at a point $\mathbf{x}$ and time t.

$$\frac{\partial^2}{\partial t^2} u = -\Delta u \tag{12}$$

Here, the vector field describes the second-time derivative of u. We proceed with the Method of Lines:
Discretize space into $n = |\mathbb{X}|$ spatial points, compute matrix L approximating Δ. Similarly to the heat equation, we end up with the system of n 2nd order ODEs

$$\frac{d^2}{dt^2} u(t) = -Lu(t) \in \mathbb{R}^n$$

As we prefer feeding 1st order ODEs to our solvers, we will convert our n second order ODEs to $2n$ first-order ODEs with the stacking equation (5)

$$\frac{d}{dt} \vec{u} = \frac{d}{dt} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} v \\ -Lu \end{pmatrix} \in \mathbb{R}^{2n}$$

By linearity of L we now apply the idea of eq. 6 to write everything as a single system matrix

$$= \begin{bmatrix} \mathbf{0} & \mathrm{I}_n \\ -L & \mathbf{0} \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = A\vec{u_i}$$

A is of dimension $2n \times 2n$ but highly sparse. The identity matrix I_n plays the same role as the "1" in eq. 6.

The sparse block-pattern in high derivative-order system matrices will be repeated in chapter 6 where we will specify prior distributions over state-space encoded PDE solutions. For good measure, we will take a few more examples of linear PDEs and give their resulting system matrix.

Intuition: MOL on different PDEs and their 1st Order ODE Representation

In the following table, $\mathbf{I}$ is the $n \times n$ identity matrix, and $\mathbf{0}$ the $n \times n$ zero matrix.

Linear PDE	System Matrix	$\vec{u}$
$\frac{\partial}{\partial t} u = -\Delta u$	$\begin{bmatrix} -L \end{bmatrix}$	$\begin{pmatrix} u \end{pmatrix}$
$\frac{\partial^2}{\partial t^2} u = ku - d\Delta \frac{\partial}{\partial t} u$	$\begin{bmatrix} \mathbf{0} & \mathbf{I} \\ k\mathbf{I} & -dL \end{bmatrix}$	$\begin{pmatrix} u \\ v \end{pmatrix}$
$\frac{\partial^2}{\partial t^2} u = a\Delta \frac{\partial}{\partial t} u$	$\begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & aL \end{bmatrix}$	$\begin{pmatrix} u \\ v \end{pmatrix}$
$\frac{\partial^3}{\partial t^3} u = \Delta u$	$\begin{bmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \\ L & \mathbf{0} & \mathbf{0} \end{bmatrix}$	$\begin{pmatrix} u \\ v_1 \\ v_2 \end{pmatrix}$
$\frac{\partial^4}{\partial t^4} u = \Delta u + a\frac{\partial^2}{\partial t^2} u$	$\begin{bmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \\ L & \mathbf{0} & a\mathbf{I} & \mathbf{0} \end{bmatrix}$	$\begin{pmatrix} u \\ v_1 \\ v_2 \\ v_3 \end{pmatrix}$

Notice the presence of the off-diagonal series of $\mathbf{I}$. For an order q PDE (or ODE), the upper $q \cdot n$ rows of the system Matrix will have this pattern to encode the derivatives.

Approximating the spatial derivatives is a source of error. The probabilistic numerical solver in this thesis is only capable of modeling the solution continuously in the temporal domain. The Method of Lines proceeds in two independent steps, spatial discretization and temporal discretization. The way these two steps introduce approximation error are however *not* independent. The error from a coarse spatial resolution can not be alleviated by arbitrarily increasing the temporal resolution and vice-versa. In [34] the authors propose the *Probabilistic Numerical Method of Lines*, which quantifies the spatial discretization error with probabilistic finite-difference methods. This method works only in Euclidean space, and developing a similar method for manifolds is an open research question.

6 Probabilistic Numerical Solver for ODEs

This chapter builds on chapter 5. We have now covered almost all prerequisites to characterize probabilistic numerical solvers of ODEs. A probabilistic numerical (PN) solver of ODEs is fundamentally different than a classic numerical solver. We will use the model first defined in [54] and cover a few tricks from [36]. A PN solver starts with a prior distribution over the solution function u, which is then conditioned on information about the vector field f. The resulting posterior distribution is referred to as a PN solution of the ODE ([36], [7]). To get to this point, we will first need to specify our prior.

Specifying a Prior Distribution

Prior distributions over functions are a natural way to model uncertainty in the solution of an ODE. We will use a prior distribution $\mathcal{P}$ over the full state-space representation of u, containing u and its first q time-derivatives.

$$\vec{U}(t) := \begin{bmatrix} U & \frac{d}{dt}U & \cdots & \frac{d^q}{dt^q}U \end{bmatrix}^\top \sim \mathcal{P}$$

We will choose a prior that enables a tractable computation of the later posterior, which limits our choice. We will focus on Gauss-Markov [26] processes, which are a Gaussian processes [45] that additionally have the Markov property.

Definition: Markov Property for Stochastic Processes

A stochastic process U has the Markov property if for all $i < s < t$

$$U(t) \perp U(i) \mid U(s)$$

In words, the future is independent of the past given the present. From the perspective of $U(t)$, $U(s)$ must contain all relevant information about the past, since it is independent of all preceding states $U(i)$.

This constraint on the prior leads one to consider Linear Time-Invariant Stochastic Differential Equations (LTI SDEs). One could also have used time-varying linear SDEs, as in [36], but it is less simple.

Definition: Linear Time-Invariant Stochastic Differential Equation

We will consider n-dimensional linear SDE written in the form

$$dX(t) = FX(t)dt + \sigma LL^\top dW(t) \tag{13}$$

where $X(t) \in \mathbb{R}^n$ is the state, $F \in \mathbb{R}^{n\times n}$ is the system matrix, σ a positive scalar parameter, $L \in \mathbb{R}^n$ and $LL^\top$ is the diffusion matrix, and $W(t) \in \mathbb{R}^n$ is a standard Wiener process[a]. [54]. This is known as the Itô form of the SDE. The solution of the SDE is a stochastic process $X(t)$ [50]. Time-invariant means that the coefficients F and L are constant, that is, do not depend on t.

[a] This is a Wiener process with zero mean and standard normal Gaussian increments.

LTI SDEs are a desirable choice as our prior $\mathcal{P}$, as they have the Markov property and have closed form solutions for the posterior distribution under Gaussian initial conditions and a linear Gaussian observation model. We will be considering LTI SDEs where L is all zeros except for the last entry, which is 1. Additionally, the initial time will be $t = 0$.

Solving LTI SDEs

For an initial Gaussian distribution $\mathcal{N}(\mu_0, \Sigma_0)$, the distribution of the LTI SDE at future times will remain Gaussian, and can be expressed in closed form. For the linear SDE in eq. 13, the following holds

$$\mathbb{E}[U(t)] = e^{Ft}\mathbb{E}[U(0)] \tag{14}$$

Which says that, in expectation, it will satisfy the differential equation encoded by F exactly. This is essentially the same expression as the analytical solution to a linear ODE (eq. 9). The covariance matrix between states at time t and s is given by [54] as

$$\mathrm{Cov}[U(t), U(s)] = e^{Ft}\Sigma_0 e^{F^\top t} + \int_0^{\min(s,t)} e^{F(t-\tau)} L\sigma^2 L^\top e^{F^\top(t-\tau)} d\tau \tag{15}$$

The solution of the LTI SDE is a Gaussian Process with the above mean and covariance function ([26], [45]). For the purposes of this thesis, this is mostly a fun fact - we will not be working with the full Gaussian Process or building its Gramian matrix, but will be clever about using the Markovianity of the process.

Intuition: Motivating Linear SDEs

A LTI SDE can be obtained by injecting Gaussian noise into a deterministic time-invariant ODE. We start off with the harmonic oscillator with an external force $f(t)$

$$\frac{d^2}{dt^2}u = -ku - l\frac{d}{dt}u + f(t)$$

If we do not know the external force and believe it to be independent of previous times, we can model it as a Gaussian noise term $\xi(t) \sim \mathcal{N}(0, \sigma^2)$, giving the linear SDE

$$\frac{d^2}{dt^2}U = -kU - l\frac{d}{dt}U + \xi(t)$$

This again states that the second derivative is a combination of the first derivative and the function itself, plus noise. However, the noise is not continuous, which conflicts with the definition of $\frac{d^2}{dt^2}U$. We circumvent this issue by combining the SDE notation with a state-space representation of U as

$\vec{U} = [U \ \frac{d}{dt}U]^\top$

$$d\vec{U} = \begin{bmatrix} 0 & 1 \\ -k & -l \end{bmatrix} \vec{U}dt + \sigma \begin{bmatrix} 0 \\ 1 \end{bmatrix} dW(t)$$

Here, $dW(t)$ is the "increment" of a 1-dimensional Wiener process, and is the SDE notation for a value drawn from $\mathcal{N}(0, 1)$ ([54]).
Below is shown this spring model. The initial distribution of the state is chosen as

$$\vec{U}(0) \sim \mathcal{N}([1, 0]^\top, \text{diag}([0.2^2, 0.1^2]))$$

We will return to this example at the very end of this chapter.

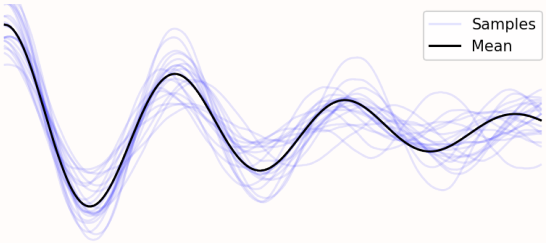


Figure 21: 25 samples from the linear SDE describing the stochastic spring model with random independent force. The random force has $\sigma = 0.1$ and we have $k = 1$, $l = 0.2$.

Discretizing the Prior in Time

For a finite set of timesteps, the joint distribution of the finitely many states $\vec{U}(t)$ across timesteps becomes a multivariate Gaussian, as explored extensively in [45]. In the spirit of the Method Of Lines (sec. 5), we will therefore consider a finite set of times $\mathbb{T} = \{0, h, 2h, 3h, \ldots, T\}$, where h is the timestep size and T is the final time. For simplicity, we choose a fixed timestep, but it can be heterogenous or even adaptive, see [36].
The Markov property then enables the discrete time stochastic recurrence relation [26] using eq. 14 and 15

$$\vec{U}(t + h) \mid \vec{u}(t) \sim \mathcal{N}(A\vec{u}(t), Q) \tag{16}$$

with discrete-time transition matrices

$$A = e^{Fh} \tag{17}$$

$$Q = \int_0^h e^{Ft} L\sigma^2 L^\top e^{F^\top t} dt \tag{18}$$

The formula 17 can be computed numerically quite simply with JAX [9] or other numerical linear algebra libraries. 18 is difficult, but using Matrix Fraction Decomposition it too can be reduced to a matrix exponential.

Definition: Matrix Fraction Decomposition

The following algorithm for A and Q is adapted from [50] but simplified for our fixed-time-step and time-invariant case.
Assume the SDE to be discretized is of the form as in eq. 13. Compute the matrix exponential M of the block-matrix

$$M = \exp\left(h \begin{bmatrix} F & \sigma^2 LL^\top \\ 0 & -F^\top \end{bmatrix}\right) \in \mathbb{R}^{2n \times 2n}$$

Extract the upper left block of M as $A \in \mathbb{R}^{n \times n}$ and the upper right as $C \in \mathbb{R}^{n \times n}$. Finally, compute Q as $Q = CA^\top$.

We can now form the conditional distribution of the state at each time given the state at the previous time. For brevity, we will from here on refer to as the discrete-time states $\vec{U}(i \cdot h)$ with integer subscript index $\vec{U}_i$. When combining the conditional distribution (eq. 16) with an initial Gaussian distribution

$$\vec{U}_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$$

we can form the factorized joint distribution of the process across all discrete times as

$$P(\vec{u}_\mathbb{T}) = P(\vec{u}_0, \vec{u}_1, \ldots, \vec{u}_{|T|}) =$$

$$\mathcal{N}\Big(\vec{u}_0;\ \mu_0, \Sigma_0\Big) \prod_{t=1}^{|\mathbb{T}|} \mathcal{N}\Big(\vec{u}_t;\ A\vec{u}_{t-1}, Q\Big)$$

This is an instance of a Linear Gaussian Model [33], which is a tractable and chain-structured Bayesian network [42], depicted below.

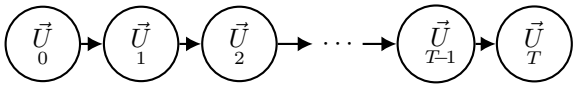


Figure 22: The Bayesian network of the discrete-time states $\vec{U}_i$.

Using the distribution as a Prior

To use the discrete-time distribution of functions as a prior distribution, we need to integrate further information into the system that we can then update the prior with. We will extend the Bayesian network with potential observations O_i of each state $\vec{U}_i$. Although we might not know the complete state at each instant, there might be properties that we do know, such as the derivative of the function at some time.
To build on the network in fig. 22, we will define the conditional distributions of a partial observations of some of the time instances. These partial observations at time i are expressed as $O_i \sim \mathcal{N}(m(\vec{U}_i), \Sigma_m)$, where m is an affine[§§] function of $\vec{U}_i$. m will be thought of as a measurement model, although this interpretation can bring along physical assumptions that are not always

[§§]The measurement model has to be an affine function of the state, as the joint distribution will only be multivariate Gaussian under linear transformations.

helpful. We will usually have $\Sigma_m = \mathbf{0}$[¶¶], but nonzero noise can be useful, as in [34][***]. Each O_i is a function of only the state $\vec{U}_i$, and so the extended Bayesian Network takes the following form The specific choice

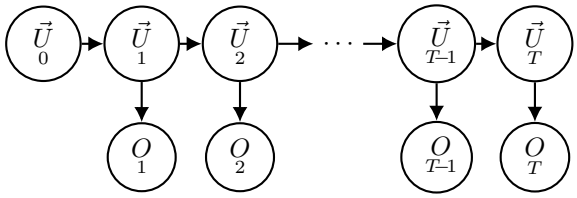


Figure 23: The full Bayesian network we will be working with. One can then condition on values of some or all observations O_i and compute the posterior distribution of the states $\vec{U}_i$

of measurement model will depend on the problem. It can abstractly be modeled as the Information Operator.

Definition: Information Operator

The Information Operator ([8]) is a measurement model that maps specific functions of interest to zero. Zero is an arbitrary choice made for convenience, it could have been any other number. Information operators are generally not affine, but they can, using the Jacobian, be linearized around the predicted mean of the unobserved state with a first order Taylor approximation. This is a topic for the Kalman Filter section.

To get a sense of the usual prior distributions used in probabilistic numerical solvers, we will introduce two common choices, the Integrated Wiener process and the Integrated Ornstein-Uhlenbeck process.

¶¶This case is reffered to as the "Dirac likelihood" [7] because it places all mass on a single event.

***Here, the authors use it to model uncertainty in the ODE description itself, stemming from a coarse discretization of an underlying PDE.

Definition: Integrated Wiener Process

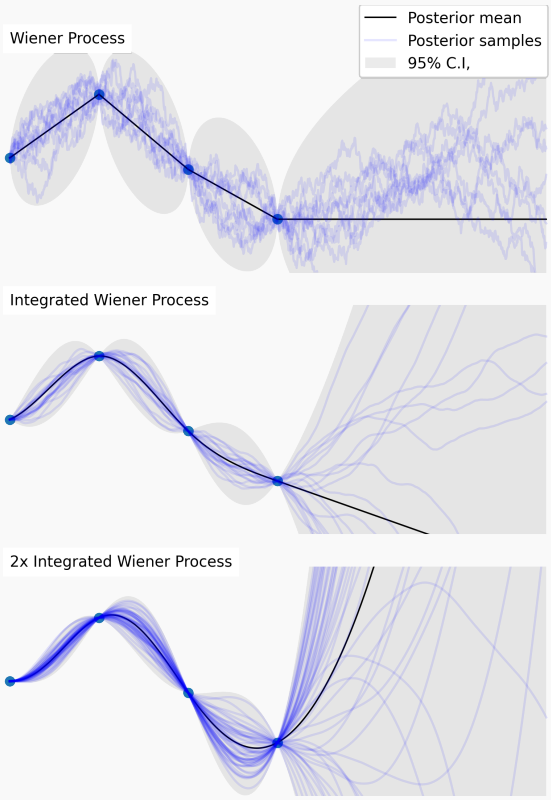


Figure 24: Three graphs of the [x-Integrated] Wiener Process. The processes are conditioned to pass through four different points, marked in blue, the derivative is however unspecified.

The Wiener Process, also known as Brownian motion, is continuous but nowhere differentiable. In terms of eq. 13 its state-space representation is 1-dimensional with

$$F = 0 \qquad L = 1$$

The Integrated Wiener Process is the integral of the Wiener Process. The state-space representation is 2-dimensional with

$$F = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The Twice Integrated Wiener Process has 3-dimensional state-space with

$$F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad L = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The posterior means of the q-times integrated Wiener process are 2q + 1-ic splines ([26]) - see the figure above. There is in principle nothing preventing us from using higher order integrated Wiener processes, but numerical issues can be encountered already at $q = 3$. We will tackle this later.

Definition: Ornstein Uhlenbeck Process

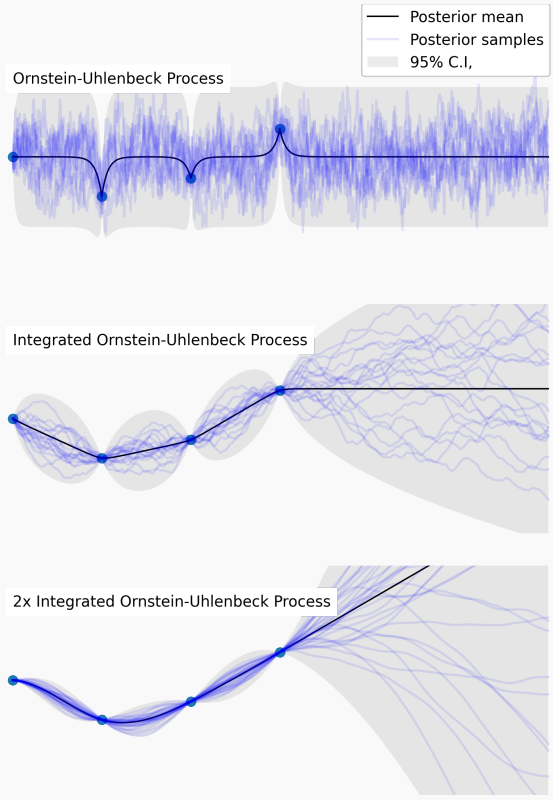


Figure 25: Draws from the Integrated Wiener Process, Conditioned

The Ornstein-Uhlenbeck Process depends on rate parameter $a > 0$. In terms of eq. 13 its state-space representation is 1-dimensional with

$$F = -a \qquad L = 1$$

The Integrated Ornstein-Uhlenbeck is the integral of the Ornstein-Uhlenbeck. The state-space representation is 2-dimensional with

$$F = \begin{bmatrix} 0 & 1 \\ 0 & -a \end{bmatrix} \qquad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The Twice Integrated Ornstein-Uhlenbeck has 3-dimensional state-space with

$$F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -a \end{bmatrix} \qquad L = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

In [7] they suggest using the Ornstein-Uhlenbeck process to encode prior information to solve particularly difficult "stiff" DEs.
The Ornstein-Uhlenbeck process is mean-reverting [26], the derivative of the Integrated Ornstein-Uhlenbeck reverts to zero, and the curvature of the Twice Integrated Ornstein-Uhlenbeck reverts to zero.

Defining the state projection matrices will be helpful for notation. They appear commonly in the literature on probabilistic numerical solvers.

Definition: State Projection Matrices E_q

We want an convenient way of selecting a specific order derivative q from the state $\vec{U}_i \in \mathbb{R}^{n\cdot(1+q)}$. We can do so by projecting the full state onto the q-th derivative with the projection matrices $E_q \in \mathbb{R}^{n\times n\cdot(1+q)}$, which are block matrices

$$\begin{aligned} E_0 &= \begin{bmatrix} \mathbf{I}_n & \mathbf{0} & \cdots & \cdots & \mathbf{0} \end{bmatrix} \\ E_1 &= \begin{bmatrix} \mathbf{0} & \mathbf{I}_n & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} \\ &\cdots \\ E_q &= \begin{bmatrix} \mathbf{0} & \cdots & \cdots & \mathbf{0} & \mathbf{I}_n \end{bmatrix} \end{aligned}$$

They have the property that $E_q\vec{U} = \frac{d^q}{dt^q}U$ and that they are linear transformations.

Definition: ODE Residual Measurement Function

An instance of an Information Operator that we will make heavy use of is the ODE residual measurement function. It will map all functions that satisfy the ODE to zero. For an example linear ODE

$$\frac{d^2}{dt^2}U(t) = AU(t)$$

it is defined as

$$\mathcal{R}\vec{U}(t) = E_2\vec{U}(t) - AE_0\vec{U}(t) \in \mathbb{R}^n$$

which is expressing the residual of the ODE in terms of the state-space representation. A solution of the ODE will have $\mathcal{R}\vec{U}(t) = \mathbf{0}$.

If definition of $\mathcal{R}\vec{U}(t)$ refers to some derivative E_q, then the prior distribution must model at least q derivatives for it to be defined. By conditioning the prior residual of $\mathcal{R}\vec{U}(t)$ to be zero at all times $t \in \mathbb{T}$, we are "weeding out" the functions that surely do not satisfy the ODE, forcing the state-space representation of the function to satisfy the ODE. The associated observation model with the residual operator as measurement function is deterministic and given as

$$O_i \sim \mathcal{N}(E_2\vec{U}(t) - AE_0\vec{U}(t), \Sigma_h)$$

The residual operator can be used to encode linear *PDEs* as well:

Intuition: More examples of Residual Measurement for PDEs

One can also encode PDEs with the residual operator by first converting it to an ODE using the method of lines. We will convert the examples from section 5. ▶

Linear PDE	$\mathcal{R}\vec{U}$
$\frac{\partial}{\partial t}u = -\Delta u$	$E_1\vec{U} + dLE_0\vec{U}$
$\frac{\partial^2}{\partial t^2}u = ku - d\Delta\frac{\partial}{\partial t}u$	$E_2\vec{U} - kE_0\vec{U} + dLE_1\vec{U}$
$\frac{\partial^2}{\partial t^2}u = a\Delta\frac{\partial}{\partial t}u$	$E_2\vec{U} - aE_1\vec{U}$
$\frac{\partial^3}{\partial t^3}u = \Delta u$	$E_3\vec{U} - LE_0\vec{U}$
$\frac{\partial^4}{\partial t^4}u = \Delta u + a\frac{\partial^2}{\partial t^2}u$	$E_4\vec{U} - LE_0\vec{U} - aE_2\vec{U}$

Intuition: Function Values at specific times

How did we generate the figures in fig. 24 and get the posterior to pass through the initial four points? For all priors, we chose an initial determinisitc distribution $\vec{U}(0) \sim \mathcal{N}(\mathbf{0}^{(q+1)}), \mathbf{0}^{(q+1)\times(q+1)}$ and define the affine observation function $h(\vec{U}_i) = E_0\vec{U}_i \in \mathbb{R}$ with $E_0 = [1\ 0\ 0]^\top$.
The four blue points $P_t \in \mathbb{R}$ at times $t \in \{0, \frac{1T}{8}, \frac{2T}{8}, \frac{3T}{8}\}$ are reflected in the posterior by conditioning the prior on the data points $E_0\vec{U}_t = P_t$. The posterior will then pass through these points. The derivative is however not specified, and which lets the prior "speak" and impose regularity [a].

[a] C^q regularity for the q-times integrated Wiener Process [26]

We have so far postponed the actual task of computing the posterior. This is what we will cover now.

Kalman Filter and Rauch-Tung-Striebel Smoother

The sparse connections of the Gaussian Bayesian network in fig. 23 can be exploited. Together, the Kalman Filter and RTS Smoother form a message passing algorithm ([33]) for computing the conditional distribution of linear Gaussian models. The chain-structure allows for a very efficient ordering of the messages, going from $i = 0$ to $i = |T|$ in a forward pass and from $i = |T|$ to $i = 0$ in a backward pass.

Definition: Forward Pass, Kalman Filter

The Kalman Filter [30] computes, for a linear Gaussian graphical model like in fig. 23 and observations o_i (for all or some i), the distribution

$$p\left(\vec{U}_i \,\middle|\, o_{1:i}\right) \qquad i \in \{1, \ldots, |\mathbb{T}|\} \tag{19}$$

in increasing order of i. This is the posterior distribution of each state, given all observations up to that time. These distributions have no knowledge of future information. However, eq. 21 includes the terminal time distribution,

$$p\left(\vec{U}_{|\mathbb{T}|} \,\middle|\, o_{1:|\mathbb{T}|}\right) \tag{20}$$

which is the posterior of the final state given information about *all* the observations (since there are no future observations). If one is interested only in the posterior at the final time, then the Kalman Filter forward pass is enough.

The term "filter" refers to accumulating [uncertain] information about past states into some estimate of the current state. An exponential moving average is also a filter.

Intuition: Choosing the noise scale σ^2

In our definition of the prior LTI SDE, we left the parameter σ unspecified. This parameter is the noise scale, and it is a hyperparameter that must be chosen. [36] derives a maximum likelihood estimate of σ, which can be computed cheaply during the forward pass. The worries of hand-tuning σ are therefore removed, and the σ that maximises the probability of the observations is chosen. Our implementation uses this method.

Intuition: Sampling the Posterior

If one intends to run the backward pass or sample, the forward pass must additionally return the "backwards" conditional distributions

$$p\left(\vec{U}_i \,\middle|\, \vec{U}_{i+1}, o_{1:i}\right)$$

which define a chain-structured Bayesian network like fig 22, with the arrows pointing in the reverse direction. One can then use ancestral sampling, starting with a sample from 20 and sampling from the backwards distributions to get a sample from the full posterior. Ancestral sampling involves iterating through the states in the backwards direction, and is thus best left for the backward pass. This is how we draw posterior sample in figures 24 and 25.

Definition: Backward Pass, Rauch-Tung-Striebel Smoother

Given the forward posteriors and backward transitions from the Kalman filter forward pass, the RTS smoother [46] runs a backward pass that computes the full posterior distribution of the states,

$$p\left(\vec{U}_i \,\middle|\, o_{1:|\mathbb{T}|}\right) \qquad i \in \{|T|, \ldots, 0\} \tag{21}$$

This is the marginal posterior distribution of each state, given *all* observations.

After a forwards and backwards pass, all information

about the observations O_i has reached every state, which ensure that the final distribution reflects the posterior distribtution given all observations, not just the preceding observations.
Together, they facilitate computing the posterior in time $O(|\mathbb{T}|n^3)$ ([54]), where n is the size of the state-space representation of the ODE. Note the linear dependence on amount of timesteps, which is also present in classical numerical solvers ([26]). It is remarkable that the same can be achieved while additionally modeling the full posterior distribution.
We mentioned that the measurement function has to be affine, and therefore kept the examples of the information operator affine as well. To break enter the world of nonlinear vector fields, we will make use of the Extended Kalman Filter forward pass.

Definition: Forward Pass, Extended Kalman Filter

The Extended Kalman Filter is used extensively in robotics and control [26], but is also applicable for nonlinear PDEs ([36], [8]).
As mentioned, the observation model is of the form

$$O_i \mid \vec{u}_i \sim \mathcal{N}(m(\vec{u}_i), \Sigma_m)$$

and requires an affine m. For a nonlinear measurement function n, we will adaptively update n to be affine during the forward pass by Taylor-expanding it around the most recent believed mean of state $\vec{U}_i$, $\bar{u_i} = \mathbb{E}[\vec{U}_i \mid o_{1:i-1}]$[a]. The Taylor approximation of n at time i becomes

$$\bar{n_i}(y) = n(\bar{u_i}) + J(\bar{u_i})(y) - J(n)(\bar{u})$$

with J the Jacobian matrix of n. It will be the most accurate around the prior mean. With this adaptive measurement function defined, one can use the classic forward Kalman filter forward pass with the conditional distribution

$$O_i \mid \vec{u}_i \sim \mathcal{N}((\bar{n_i}(\vec{u}_i)), \Sigma_m)$$

Because we can linearize the (assumed to be differentiable) Information operator, we can more generally define the residual measurement $\mathcal{R}\vec{U}(t)$ for a nonlinear PDEs

$$\frac{\partial^q}{\partial t^q} u = f(u)$$

as the nonlinear measurement function

$$\mathcal{R}\vec{U}(t) = E_q \vec{U}(t) - f\left(\vec{U}(t)\right)$$

The Extended Kalman filter then takes care of interpreting this as an affine function of the state using automatic differentiation.

[a] This conditional distribution is obtained by applying eq. 16 with eq. 21 and marginalizing.

More Implementation Details

We implement our solver in JAX [9], which is a Python library for automatic differentiation and high performance numerical linear algebra. It further allows just-in-time compilation of functional programs, which greatly speeds up all procedures by vectorizing and parallelizing where possible. With 64-bit floating point precision, one might hope that the numerical stability of the solver would not an issue. However, the covariance matrices that are formed in the Kalman Filter and RTS Smoother can have very poor condition numbers, which can lead to numerical instability. The covariance matrix generally stores entries that are the square of the length-scale of the state. For small timesteps, the standard deviation in the conditional distribution between states will be small, and the covariance matrix in 18 will contain the square of this small number. This can lead to vanishingly small entries in the covariance matrix that worsen the condition number.
"Roughly speaking, the condition number states how much precision is lost when inverting a matrix and the larger the number, the more precision is lost. Huge condition numbers render the solution of linear systems practically infeasible." - from [36]

Definition: Improving Numerical Stability, generalized Cholesky factors

In [36] they suggest using generalized Cholesky factors [21] of the covariance matrix to improve numerical stability. Generalized Cholesky Factors always exist for symmetric, positive semidefinite matrices[a].
A covariance matrix $\Sigma \in \mathbb{R}^{n\times n}$ can be factored as $\Sigma = CC^\top$ where $C = \Sigma^{(1/2)} \in \mathbb{R}^{n\times n}$ is a lower triangular matrix. A multivariate Gaussian distribution is then parameterized by a mean vector and the generalized Cholesky factor of the covariance matrix.
[36] give the full description of how, given a Cholesky parameterization of $X \sim \mathcal{N}(x; Ax + b, \Sigma)$ and $Y \mid X = x \sim \mathcal{N}(y; Cx + d, \Sigma)$, one can compute the posterior $P(X|Y)$, marginal $P(Y)$ and the likelihood $P(Y|X)$ in closed form, all without having to form the covariance matrix. This sidesteps the numerical instability of the covariance matrix.

[a] The covariance matrices are usually positive definite, but it will be semidefinite when some part of state is deterministic.

As a guesstimate, using the generalized Cholesky factors representation allowed us to take $\times 1.5$ more steps with the solver before numerical instability invalidated the results. It however turns out that the covariance matrix is still ill-conditioned due to an orthogonal issue. The matrix Q depends still on the step-size h in 18. In the case of the integrated Wiener process, for q derivatives, the condition number grows at least as 2^q an in practice is worse for low h, yielding entries down to 10^{-40} for $q = 3, h = 0.0001$ ([36]).

Definition: Improving Numerical Stability, Preconditioning

As argued and solved in [36], Q should not depend on the length of the time step. They give an invertible linear coordinate transformation T to the SDE, which will remove this dependency. Because we are working with LTI SDEs, this transformation can be applied at the SDE level to the matrices as TFT^{-1} and $TLL^\top$. It can then be undone at the end of the backward pass to get the posterior in the original state-space.
The transformation is derived for the integrated Wiener process, but when implementing it in the thesis and applying it to other LTI SDEs we were still able to model up to 10 derivatives in all processes instead of the previous 3. This is a significant improvement and unlocks the possibility of using higher order integrated processes.

The first state and derivatives can be informed by the vector field f - if the DE is of the form $\frac{d}{dt}u = f(u)$, we immediately get the expression for the first derivative as $f(u(0))$. If however the DE is of the form $\frac{d^2}{dt^2}u = f(u)$, we cannot extract the initial derivative, and it is considered in input value, like the initial value $u(0)$. As will also be explored further in chapter 7, one can repeatedly differentiate the vector field to get further derivatives of the initial state.

Definition: Initialization, Taylor-mode automatic differentiation

If using automatic differentiation, one first has to compute the Jacobian, then the Hessian and so on for higher derivatives. The complexity of this grows exponentially with the order of the derivative as $O(2^q)$ [36]. Instead they propose using Taylor-mode automatic differentiation [22], [4] to get the higher derivatives w.r.t. time, which grows as $O(\exp(\sqrt{q}))$. Our implementation uses Taylor-mode automatic differentiation function from Python library `probdiffeq` [36] to initialize the higher derivatives for the experiments to come.

Intuition: Solving the Damped Harmonic Oscillator

Finally, we want to give an example of a solving the damped harmonic oscillator with our probabilistic numerical solver. We solve the ODE

$$\frac{d^2}{dt^2}u = -0.3\frac{d}{dt}u - 1.5u$$

with initial conditions $u(0) = 1$ and $\frac{d}{dt}u(0) = 0$.
We show the posterior distribution of the 4-times integrated Wiener process after being initialized using Taylor-mode automatic differentiation and conditioning on the PDE residual

$$\mathcal{R}(\vec{U}) = \vec{U}_2 + 0.3\vec{U}_1 + 1.5\vec{U}_0$$

of the damped harmonic oscillator being zero. To show the effect of using a higher order integrated prior, we condition only at five timesteps, shown as blue dots[a]. The true solution is shown in red and is computed using the matrix exponential (eq. 9).

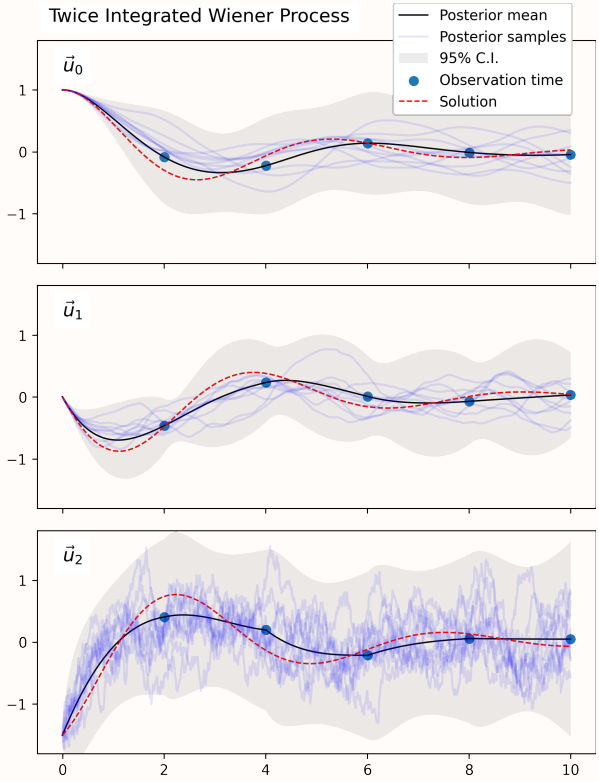


Figure 26: Probabilistic numerical solution of the damped harmonic oscillator ODE. We are using a 2-times integrated Wiener process as prior. Note the increasing irregularity of the state components as we inspect higher derivates which have been integrated less frequently. The highest derivative here, $\vec{u}_2$, is no longer differentiable.

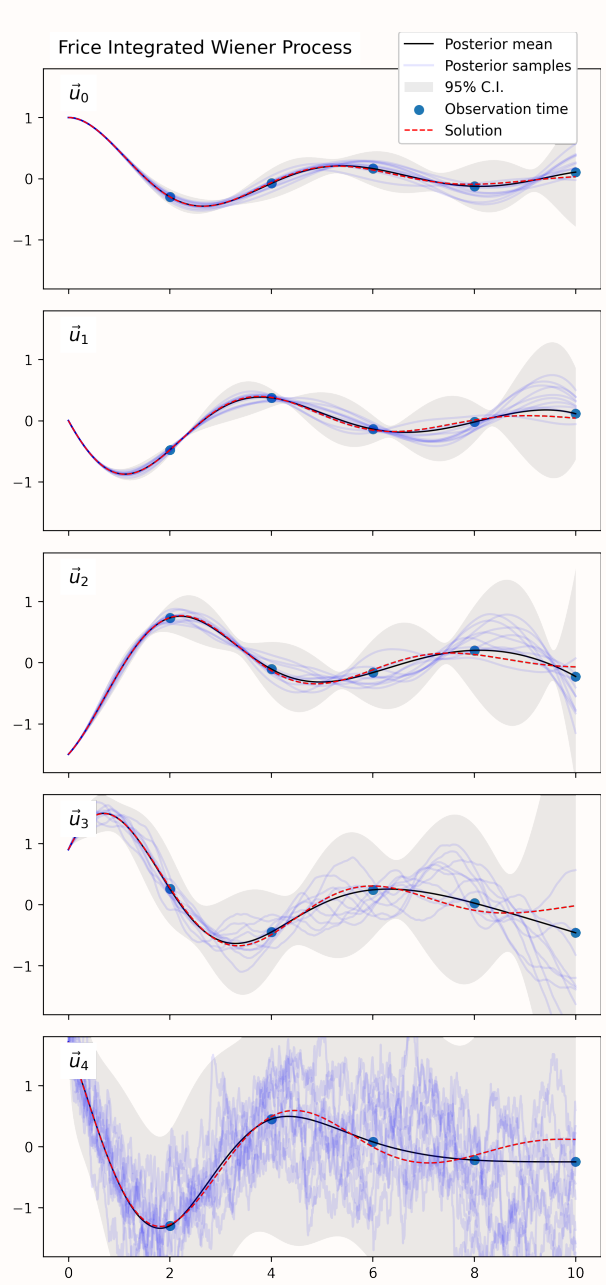


Figure 27: The same type of figure as above, but we are using a 4-times integrated Wiener process as the prior. We have conditioned on the same observations as in the previous figure, but there is much less uncertainty about the solution $\vec{u}_0$.

[a]The vertical position of the blue dots has no meaning and is arbitrarily placed on the mean of the process. We are *not* conditioning the process to pass through these points, unlike in figures 24

The example empirically shows how increasing the order of the prior leads to less uncertainty and faster convergence with fewer steps. This however comes at a cost, as each step is more costly because of the cubic dependence on the size of the state.

7 Applying the Solver and Laplacian Matrix

In this second-to-last chapter, we give demonstrations of the Laplacian matrices we have built, solved with the built PN solver from chapter 6. We will use the colormap in figure 28 for all figures in this section.

Figure 28: Colorbar for the figures in this section, using the perceptually uniform `viridis` [58] colormap. The associated scalar values will generally not be given as they are not important for the interpretation of the figures. Colors will have been renormalized for visualization purposes. However, generally, dark value are negative, light values are positive, and the midpoint is zero.

Solving the Poisson Problem

Here are solutions of the Poisson problem on the 2D surface of various shapes - since they are not time-varying, they will not be solved using our PN solver, but can be solved by a linear solver. The Poisson problem is given by:

$$-\Delta u = f \tag{22}$$

On a mesh with the discrete Laplacian L, this reduces to finding the vector u such that $Lu = f$, which is straightforward to solve with numerical linear algebra tools. For this example, we solve the Poisson problem on a model of a twisty section of kelp. The curvature of the kelp is encoded in the Laplacian matrix. Since this two-dimensional surface has a boundary, we must specify boundary conditions: We set both opposing pairs of edges to the same color, some positive and negative value, respectively (Dirichlet boundary conditions). The solution is illustrated in figure 29. The topmost figure shows the boundary conditions (the interior is set to zero for this visualization). The middle figure shows the force f, which is a positive stretched Gaussian bell on the interior. The bottommost figure shows the computed solution, which satisfies the hot/-cold boundary conditions, while the interior satisfies the curvature dictated by f.

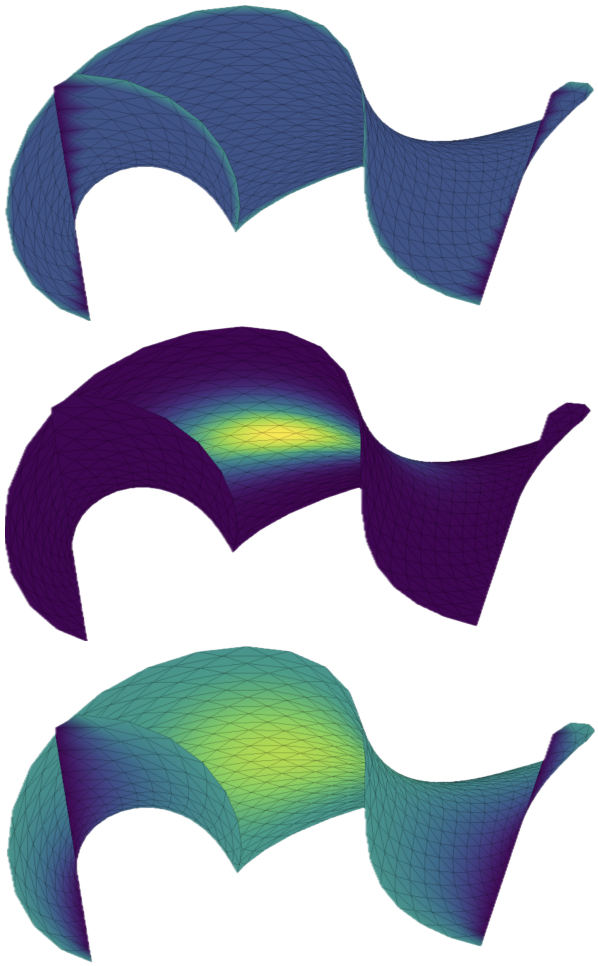


Figure 29

Solving the Heat Equation

We solve the heat equation on the sphere, using the twice-integrated Wiener process, see figure 30 for the mean of the process at selected timesteps. In figure 31 we show the marginal uncertainties of the solution. We interpret the nonuniformity of the figure in the following way: The sphere mesh used has twelve[†††] patches of vertices with incidence number 5 - in these, the solution is more certain than in the remaining ones. The Laplacian matrix will have five nonzero entries in the corresponding rows instead of six, and thus the solution here is less sensitive to the values of the neighbors. Additionally, the neighboring five vertices are spatially closer (the edge lengths are shorter), which introduces higher correlation between them, reducing overall uncertainty.

[†††]This mesh is created by using loop subdivision on the 12-tipped icosahedron with the `icosphere` Python package, which is a coarse approximation to the sphere geometry.

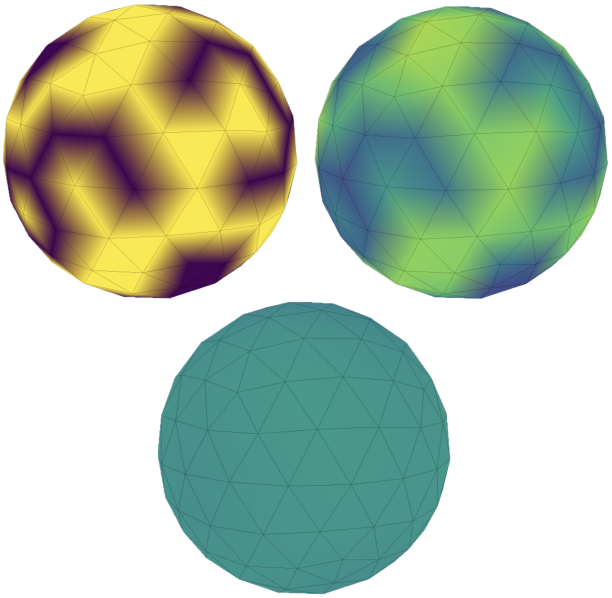


Figure 30: The heat equation solved on the sphere. Upper left image is the initial temperature distribution, where each vertex has been assigned to $\{-1, 1\}$ uniformly at random. Upper right and lower image are spaced equally apart in time, illustrating the heat diffusion. Yellows are positive temperatures, blues are negative temperatures.

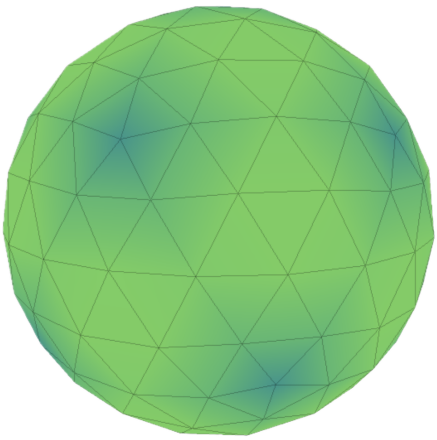


Figure 31: The marginal standard deviations of the solution at the second step in figure 30. Darker colors mean lower uncertainty. The uncertainty is miniscule and has been scaled up for illustration.

Intuition: The Laplacian captures the length-scale

The speed of diffusion is linked to the Laplacian matrix L and depends on the scale of the geometry. We visualize this dependence on the Utah teapot [41], scaled so the body of the teapot is approximately a unit sphere. Figure 32 shows the heat equation initial step and a short time later. The teapot has non-uniform dual face areas, especially around the rim at the lid. That the diffusion process is not scale-invariant is illustrated by the fact that the temperature here will smooth out much quicker than in the remaining, bigger patches.

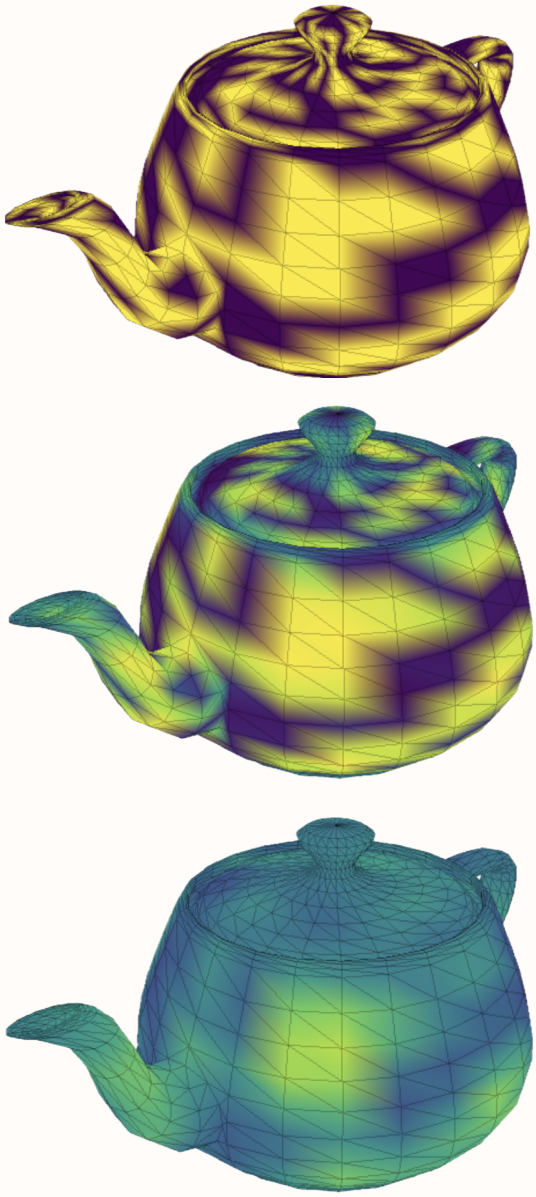


Figure 32: The heat equation being solved on the Utah Teapot. Top: The initial conditions are akin to the ones in fig. 30 and shown in the topmost figure. Middle: Heat diffusion after a short time. Bottom: Heat diffusion after a longer time.

Solving the Wave Equation On an Intrinsically Triangulated Surface

Using our algorithm from chapter 4, we use triangulate the bell-curve geometry, compute the Laplacian, and solve the wave equation on it. To motivate the expected behaviour, one can think of a wave as simply a shockwave traveling at constant speed from a source, which we visualize an approximation of in figure 33. There are multiple ways to compute the location of the front of the shockwave, such as the ones proposed

and compared in [13] and [18]. For the sake of ease of implementation, we use the simplest one, Dijkstra's single-source-all-paths algorithm. The figure shows level-sets of the distance function to the source (on the left, at coordinates $(-0.5, 0.0)$). This figure suggests that one should observe the wave traveling around the bump faster than across it, which is reasonable, given the detour the wave would take to travel across the bump. This simplification ignores the reflections and interference that might occur due to our Dirichelet BCs (which are fixed at a wave-height of zero).
The four stacked planes in figure 34, show the solution of the wave equation on the intrinsic bell-curve mesh.

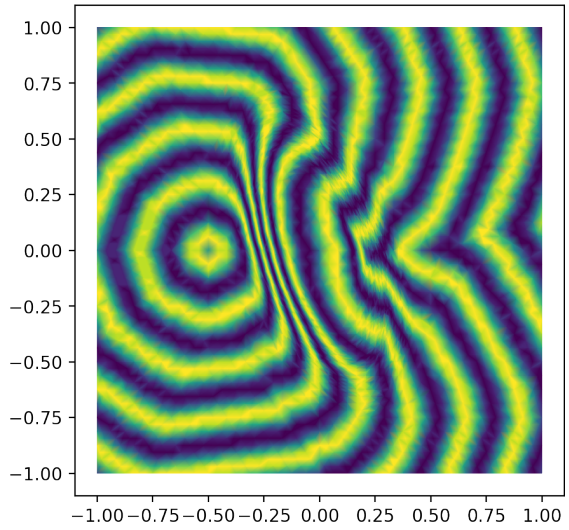


Figure 33: Level sets of the distance function from a source at coordinates $(-0.5, 0.0)$ on the intrinsic triangulation of the Gaussian bump mesh. The wave equation is solved on this mesh in the following figure.

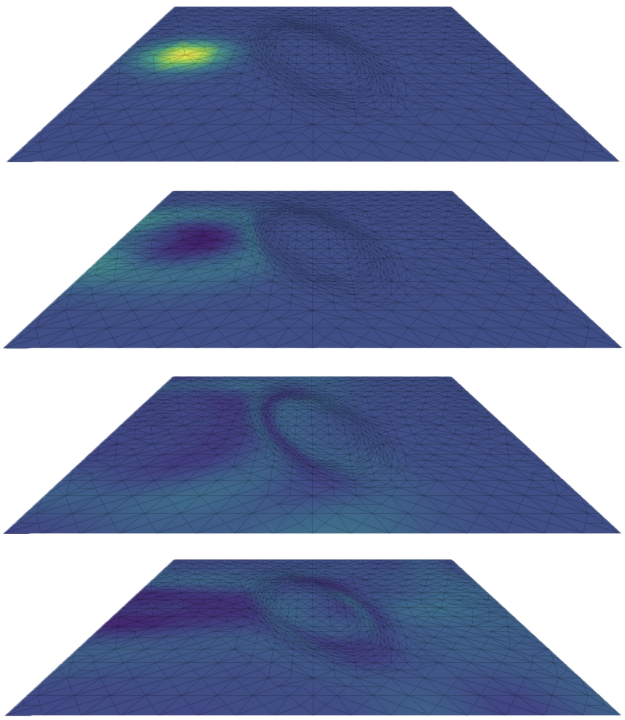


Figure 34: From top to bottom, four consecutive equally spaced instants of the wave equation on the intrinsic triangulation of the Gaussian bump mesh. The initial condition is given in the topmost picture. In the third picture from the top, there are clear similarities with the shapes shown in 33 when comparing the shape of the wave around the bump. We emphasize that the mesh is completely intrinsic and "flat" - there is no extrusion of the bump on this mesh.

8 Experiments with Physically Informed Priors

In this final short chapter, we will demonstrate the efficiency of the probabilistic solver when using physically informed priors. As problems to solve, we choose a nonlinear version of the heat (23) and wave (24) equation, with varying degrees of nonlinearity.

$$\frac{\partial}{\partial t}u = -\Delta u - \alpha u^2 \tag{23}$$

$$\frac{\partial^2}{\partial t^2}u = -\Delta u - \alpha \tan u \tag{24}$$

The hypothesis is that, for small nonlinearities, a prior that encodes the linear heat/wave dynamics will outperform the integrated Wiener process.

Encoding the Linear part of a PDE into a Prior

This idea is adapted from [7] in which the authors show that using integrated Ornstein-Uhlenbeck processes (chapter 6) give an inductive bias when solving ODEs with partially linear dynamics. Using such a prior leads to faster convergence than the usual integrated Wiener process, because it exactly solves the linear part of the ODE. We show that the same idea can be applied to PDEs and give appropriate prior process, but will start by explaining the principle.

Intuition: Mean of Prior solves Linear ODE exactly

We can take any linear ODE and make a series of transformations to the ODE that preserve the modeled system, but will explicitly model more derivatives in the state-space.
We will show the procedure with the damped harmonic oscillator

$$\frac{d}{dt}\vec{u}_0 = \vec{u}_1 \tag{25}$$
$$\frac{d}{dt}\vec{u}_1 = -a\vec{u}_0 - b\vec{u}_1 \tag{26}$$

for some fixed choice of a, b.
Assume we are given initial condition $\vec{u}_0(0) = s_0(0)$, $\vec{u}_1(0) = s_1$. We can feed the initial known values into eq. 26 to get $s_2 = \vec{u}_2(0) = -a\vec{u}_0(0) - b\vec{u}_1(0)$. We now impose regularity on $\vec{u}_2$ by taking the derivative w.r.t. time on all terms in the equations 25 and 26:

$$\begin{aligned}\frac{d^2}{dt^2}\vec{u}_0 &= \frac{d}{dt}\vec{u}_1\\ \frac{d^2}{dt^2}\vec{u}_1 &= \frac{d}{dt}\left(-a\vec{u}_0 - b\vec{u}_1\right)\\ &= -a\frac{d}{dt}\vec{u}_0 - b\frac{d}{dt}\vec{u}_1\end{aligned}$$

By the derivative/integral relationship in the state-space representation, this simplifies to

$$\frac{d}{dt}\vec{u}_0 = \vec{u}_1 \tag{27}$$
$$\frac{d}{dt}\vec{u}_1 = \vec{u}_2$$
$$\frac{d}{dt}\vec{u}_2 = -a\vec{u}_1 - b\vec{u}_2 \tag{28}$$

We have increased the state-space to encompass another derivative. Since we previously determined $\vec{u}_2(0) = s_2$, we have all the required information to solve this ODE.
The steps in this transformation can be repeated indefinitely; one needs to compute the induced initial value of the derivative from the set of equations and then augment the state-space. We start off with system matrix F', and after $q-1$ repetitions we end up with a system matrix $F \in \mathbb{R}^{q+1\times q+1}$

$$F' = \begin{bmatrix}0 & 1\\ -a & -b\end{bmatrix} \to F = \begin{bmatrix}0 & 1 & 0 & \dots & 0 & 0 & 0\\ 0 & 0 & 1 & \dots & 0 & 0 & 0\\ 0 & 0 & 0 & \dots & 0 & 0 & 0\\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\ 0 & 0 & 0 & \dots & 0 & 1 & 0\\ 0 & 0 & 0 & \dots & 0 & 0 & 1\\ 0 & 0 & 0 & \dots & 0 & -a & -b\end{bmatrix}$$

The original system matrix F' is nested inside F.
We conclude by transforming this into a LTI SDE by adding zero mean Brownian motion increments into the highest derivative. When integrated, it will still have zero mean and therefore will not affect the mean of the process. This is similar in spirit to the spring example in chapter 6, and yields the following LTI SDE which can directly be applied as a prior.

$$\frac{d}{dt}\vec{U}_i = \vec{U}_{i+1} \qquad i \in \{0,\dots,q-1\}$$
$$d\,\vec{U}_q = -b\vec{U}_{q-2}dt - a\vec{U}_{q-1}dt + \sigma dW(t)$$

The previous example was for scalar-valued ODEs, but the same principle can be applied to the state-space representation of vector-valued ODEs like the ones given at the end of chapter 6, which can be obtained after applying the Method of Lines to a PDE. The same principle is then simply applied component-wise. For a dimension n vector-valued linear ODE

$$\frac{d^2}{dt^2}u = Au + B\frac{d}{dt}u$$

with state-space system matrix F', the corresponding

q-times integrated prior will be constructed as

$$F' = \begin{bmatrix} \mathbf{0} & \mathbf{I}_n \\ A & B \end{bmatrix} \to F = \begin{bmatrix} \mathbf{0} & \mathbf{I}_n & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_n & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I}_n & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{I}_n \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & A & B \end{bmatrix}$$

We then build the LTI SDE with the diffusion matrix $LL^\top = E_q E_q^\top$, which ensures that the noise is only fed into the highest derivative.
This process of repeated integration yields a LTI SDE with q continuous derivatives whose mean coincides with the solution of the original ODE. We visualize this property for the scalar wave equation and then give the PDE-wave equation counterpart.

Definition: Wave Process

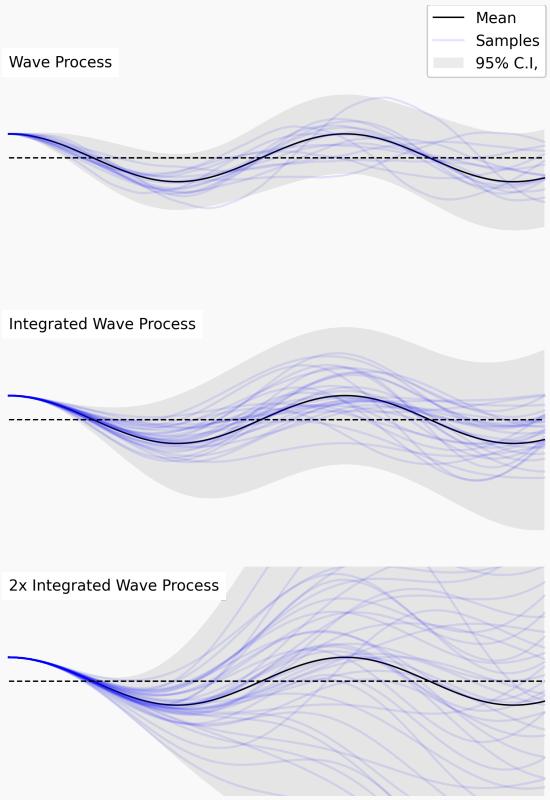


Figure 35: Iterated integrations of the Wave Process. The mean of the scalar Wave process solves the scalar wave equation $\frac{d^2}{dt^2}u = -au$ exactly. The simplest scalar Wave process has system matrix F

$$F = \begin{bmatrix} 0 & 1 \\ -a & 0 \end{bmatrix} \qquad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

(shown at the top), while the once-integrated (middle) Wave process has system matrix F

$$F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -a & 0 \end{bmatrix} \qquad L = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

We initialize the process with the initial conditions calculated from the ODE $(s_0, s_1, \dots)$, and the mean will then satisfy the ODE exactly (fig. 35). The exact solution for $a = 1$ is $\cos(t)$.

The corresponding process for the wave equation $\frac{d^2}{dt^2}u = -\Delta u$ can be built by discretizing the Laplacian matrix L and using the Wave process as the prior,

$$F = \begin{bmatrix} \mathbf{0} & \mathbf{I}_n \\ -L & \mathbf{0} \end{bmatrix} \qquad LL^\top = E_1 E_1^\top = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_n \end{bmatrix}$$

We will denote this SDE as the Wave Process.

Intuition: Wave Process in Action

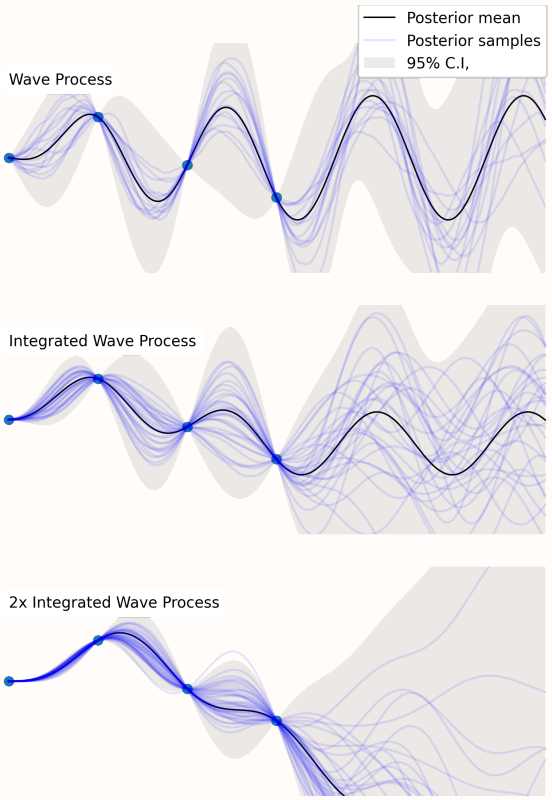


Figure 36: Draws from the Integrated Wave Process, Conditioned on the same four points that were also used in chapter 6.

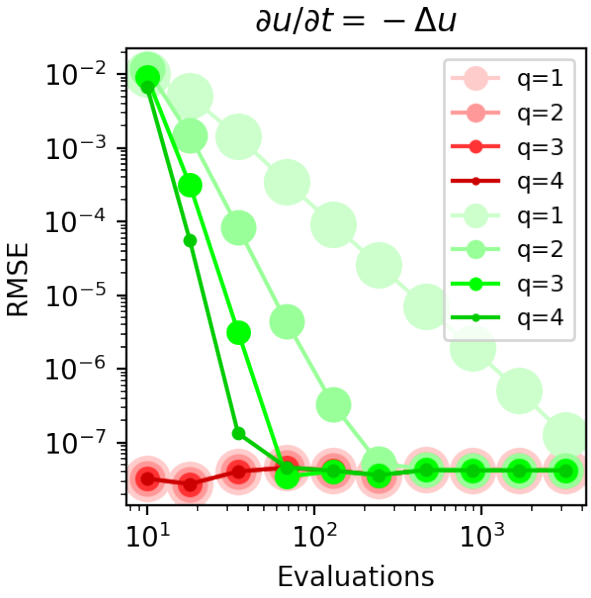
One is tempted to define a Heat process analogously by placing the Laplacian in the bottom right corner of the dynamics matrix F (which is the correct way of doing it), but it turns out that this reduces to an instance of the Ornstein-Uhlenbeck process with a specific setting of the rate-parameter. We will still refer to it as the Heat process / prior, which captures the spatial diffusion encoded in L.

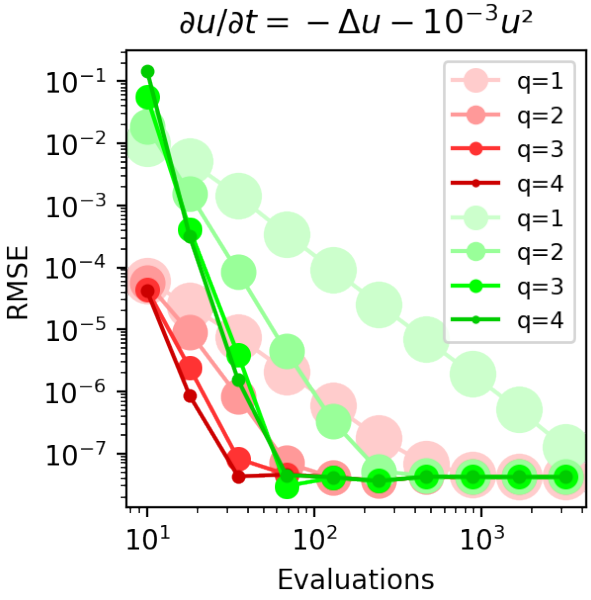
Experiments

We empirically investigate the efficiency of the probabilistic solver when using the integrated Wiener process, the integrated Wave process, and the integrated heat process as priors for the nonlinear heat and wave equations (eq. 23 & 24). As the

nonlinearities become larger and we move away from the linear dynamics, the prior will not as related and we should expect a decrease in efficiency. We run both problems for $\alpha \in \{0, 10^{-3}, 1\}$.
For eq. 23 we will use the Heat processes, and for eq. 24, we will use integrated Wave processes as our prior. We will compare against repeatedly integrated Wiener processes as priors.

The problems will be solved on the 2D surface of the sphere, approximated by a 12-vertex icosphere. The vector fields are integrated from $t_0 = 0$ to $t_{\mathbb{T}} = 10$. We will grade the PN solutions by the root-mean-square-error at all computed timesteps against a high-accuracy reference solution implemented in `diffrax` [32], using the Kvaerno5/4 method [37], suitable for stiff problems like the heat and wave equation. The reference solution has both relative and absolute tolerances of 10^{-8}, which then also serves as a lower bound for the error of the PN solution.
The PN solution will be computed for varying amount of timesteps and for different choices of q in the prior process. We will then plot the work/precision diagrams for the different choices of q and a.





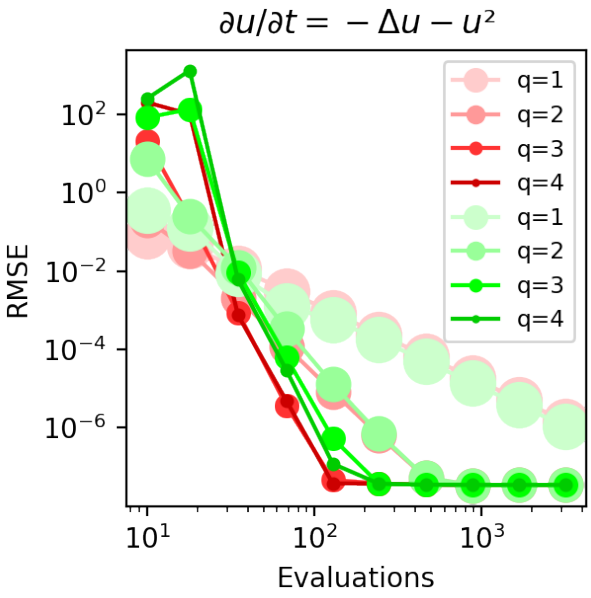


Figure 37

Results

See figures 37 through 38 for all work/precision diagrams. In the figures, the integrated Wiener process is depicted in green, the integrated Wave process in blue, and the integrated heat process in red. The x-axis shows the number of timesteps in the discretized prior ($|\mathbb{T}|$), and the y-axis shows the root-mean-square-error. As the number of steps increase, the error generally decreases. For increasing q, the error drops faster.
The integrated Wiener processes are generally being outperformed, however less decisively so for significant nonlinearities ($a = 1$). For $a = 0$ the expected exact solution of the priors is observed, even for 10 timesteps, massively outperforming the integrated Wiener process (but this was not a fair competition).
The result confirm that an appropriate prior process gives the solver an inductive bias that helps solve the ODEs more exactly and efficiently.

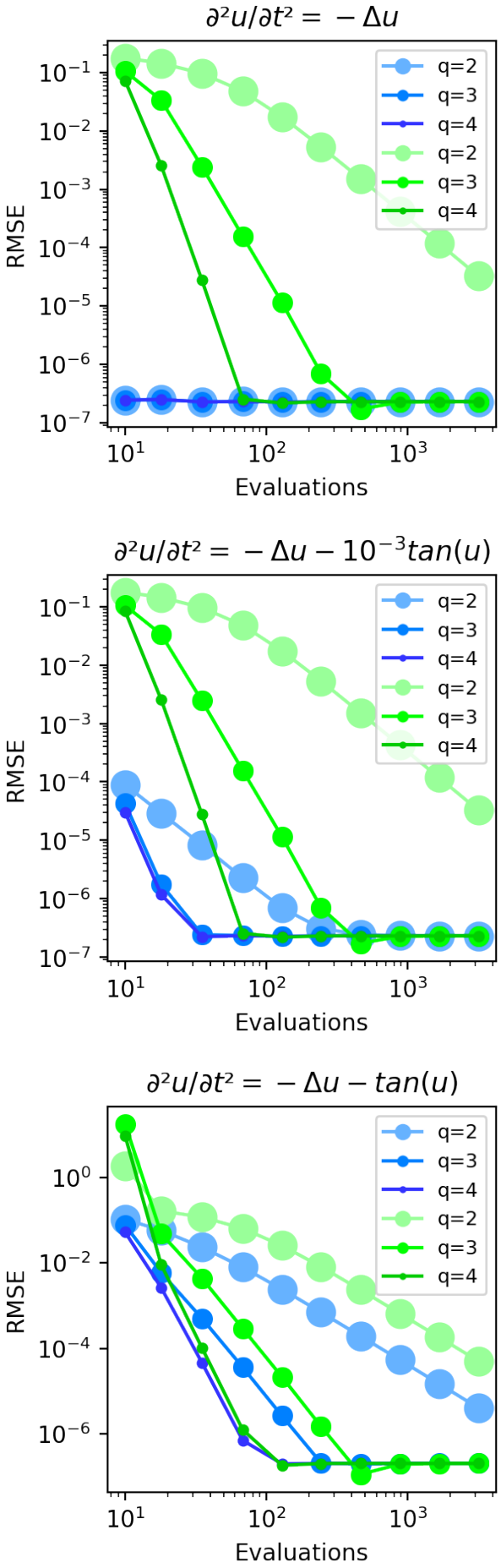


Figure 38

9 Conclusion

We have explored only few aspects of probabilistic numerical solvers on manifolds. The thesis explored only the Laplacian, but we also gave formulas for the divergence and curl, differential operators that appear, for example, in the Navier Stokes equation. Since these are linear operators too, the methods presented here should still be applicable. Probabilistic numerical fluid flow solvers on manifolds could be an interesting future research direction, especially relevant in the context of climate models. The steep cubic cost in the state-space dimension is however a drawback. This can be sidestepped by assumining spatial independence or Kronecker structure in the prior model, as in [35]. The methods of discrete exterior calculus also has many promising avenues for future research. The thesis only touched upon the basics, but the field is rich with applications in computer graphics and physics simulations.

References

[1] Georgios Arvanitidis, Lars Kai Hansen, and Søren Hauberg. "Latent Space Oddity: On the Curvature of Deep Generative Models". In: (2017). URL: https://arxiv.org/abs/1710.11379.

[2] Eberhard Bänsch. "Local mesh refinement in 2 and 3 dimensions". In: *IMPACT of Computing in Science and Engineering* 3.3 (1991), pp. 181–191. ISSN: 0899-8248. DOI: https://doi.org/10.1016/0899-8248(91)90006-G. URL: https://www.sciencedirect.com/science/article/pii/089982489190006G.

[3] Felix Berkenkamp et al. "Safe Model-Based Reinforcement Learning with Stability Guarantees". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017. URL: https://neurips.cc/Conferences/2017/Paper/6623-safe-model-based-reinforcement-learning-with-stability-guarantees.

[4] Jesse Bettencourt, Matthew J. Johnson, and David Duvenaud. "Taylor-Mode Automatic Differentiation for Higher-Order Derivatives in JAX". In: *arXiv preprint* (2023). Available at https://arxiv.org/.

[5] Alexander I. Bobenko and Boris A. Springborn. "A Discrete Laplace–Beltrami Operator for Simplicial Surfaces". In: *Discrete & Computational Geometry* (2007). DOI: 10.1007/s00454-007-9006-1. URL: https://page.math.tu-berlin.de/~bobenko/papers/2007_Bob_Spr.pdf.

[6] Nathanael Bosch, Philipp Hennig, and Filip Tronarp. "Calibrated Adaptive Probabilistic ODE Solvers". In: *arXiv preprint arXiv:2012.08202* (2020). URL: https://arxiv.org/pdf/2012.08202.

[7] Nathanael Bosch, Philipp Hennig, and Filip Tronarp. "Probabilistic Exponential Integrators". In: *arXiv preprint arXiv:2012.08202* (2020). URL: https://arxiv.org/pdf/2012.08202.

[8] Nathanael Bosch, Filip Tronarp, and Philipp Hennig. "Pick-and-Mix Information Operators for Probabilistic ODE Solvers". In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 2022, pp. 10015–10027.

[9] James Bradbury et al. *JAX: Composable Transformations of Python+NumPy Programs*. http://github.com/google/jax. 2018.

[10] R. L. Bryant et al. *Exterior Differential Systems*. Springer-Verlag, 1991.

[11] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd, 2003. URL: https://doi.org/10.1002/0470868279.

[12] Sylvain Calinon. "Gaussians on Riemannian Manifolds: Applications for Robot Learning and Adaptive Control". In: (2019). URL: https://arxiv.org/abs/1909.05946.

[13] Marcel Campen, Martin Heistermann, and Leif Kobbelt. "Practical Anisotropic Geodesy". In: *Eurographics Symposium on Geometry Processing* 32.5 (2013).

[14] Ricky T. Q. Chen et al. "Neural Ordinary Differential Equations". In: (2018). URL: https://arxiv.org/abs/1806.07366.

[15] Albert Chern et al. "Shape from Metric". In: *ACM Transactions on Graphics* (2018). DOI: 10.1145/3197517.3201276. URL: https://doi.org/10.1145/3197517.3201276.

[16] Keenan Crane. *Discrete Differential Geometry: An Applied Introduction*. Tech. rep. Carnegie Mellon University, 2023. URL: https://www.cs.cmu.edu/~kmcrane/Projects/DDG/paper.pdf.

[17] Keenan Crane. *The n-Dimensional Cotangent Formula*. Tech. rep. Carnegie Mellon University, 2019. URL: https://www.cs.cmu.edu/~kmcrane/Projects/Other/nDCotanFormula.pdf.

[18] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. "The Heat Method for Distance Computation". In: *Commun. ACM* 60.11 (Oct. 2017), pp. 90–99. ISSN: 0001-0782. DOI: 10.1145/3131280. URL: http://doi.acm.org/10.1145/3131280.

[19] Mathieu Desbrun et al. "Discrete Exterior Calculus". In: *arXiv preprint math/0508341* (2005). URL: https://arxiv.org/pdf/math/0508341.

[20] Nira Dyn, David Levine, and John A. Gregory. "A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control". In: *ACM Transactions on Graphics (TOG)* (1990). DOI: 10.1145/78956.78958. URL: https://doi.org/10.1145/78956.78958.

[21] Stuart Gibson and Brett Ninness. "Robust Maximum-Likelihood Estimation of Multivariable Dynamic Systems". In: *Automatica* 41 (2005).

[22] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd. SIAM, 2008. ISBN: 978-0-89871-659-7.

[23] Søren Hauberg, Oren Freifeld, and Michael J. Black. "A Geometric Take on Metric Learning". In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 25. 2012.

[24] Philipp Hennig. "Probabilistic Interpretation of Linear Solvers". In: *SIAM Journal on Optimization* (2015). URL: https://arxiv.org/abs/.

[25] Philipp Hennig and Søren Hauberg. "Probabilistic Solutions to Differential Equations and Their Application to Riemannian Statistics". In: *Proceedings of Artificial Intelligence and Statistics (AISTATS)*. 2014, pp. 347–355.

[26] Philipp Hennig, Michael A. Osborne, and Hans P. Kersting. *Probabilistic Numerics: Computation as Machine Learning*. Cambridge, UK: Cambridge University Press, 2022.

[27] Anil N. Hirani. "Discrete Exterior Calculus". Ph.D. Thesis. California Institute of Technology, 2003. URL: https://www.cs.jhu.edu/~misha/Fall09/Hirani03.pdf.

[28] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020. URL: https://arxiv.org/abs/2006.11239.

[29] *Hyperbolic Tiling*. URL: https://commons.wikimedia.org/wiki/File:Hyperbolic_tiling_6_5_2_8.svg.

[30] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82 (1960). DOI: 10.1115/1.3662552.

[31] Guido Kanschat and Robert Scheichl. *Numerical Analysis of Ordinary Differential Equations*. Tech. rep. Universität Heidelberg, July 2019.

[32] Patrick Kidger. "On Neural Differential Equations". PhD thesis. University of Oxford, 2021.

[33] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009. ISBN: 978-0262013192.

[34] Nicholas Krämer, Jonathan Schmidt, and Philipp Hennig. "Probabilistic Numerical Method of Lines for Time-Dependent Partial Differential Equations". In: *arXiv preprint arXiv:2110.11847* (2021). URL: https://arxiv.org/abs/2110.11847.

[35] Nicholas Krämer et al. "Probabilistic ODE Solutions in Millions of Dimensions". In: *Proceedings of the 39th International Conference on Machine Learning (ICML)*. 2022.

[36] Peter Nicholas Krämer. "Implementing Probabilistic Numerical Solvers for Differential Equations". Ph.D. Thesis. Eberhard Karls Universität Tübingen, 2023. URL: http://dx.doi.org/10.15496/publikation-94093.

[37] Anne Kværnø. "Singly diagonally implicit Runge–Kutta methods with an explicit first stage". In: *BIT Numerical Mathematics* 44 (2004).

[38] Dongyuan Li et al. "A Survey on Deep Active Learning: Recent Advances and New Frontiers". In: (2024). URL: https://arxiv.org/abs/2405.00334.

[39] Charles Loop. "Smooth Subdivision Surfaces Based on Triangles". M.S. Mathematics Thesis. MA thesis. University of Utah, 1987.

[40] Maren Mahsereci and Philipp Hennig. "Probabilistic Line Searches for Stochastic Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015.

[41] Martin Edward Newell. "The Utilization of Procedure Models in Digital Image Synthesis". ProQuest Dissertations & Theses, 7529894. PhD thesis. The University of Utah, 1975.

[42] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988. ISBN: 978-1558604797.

[43] Enrico Puppo and Daniele Panozzo. "RGB Subdivision". In: *IEEE Transactions on Visualization and Computer Graphics* (2008). URL: https://cims.nyu.edu/gcl/papers/TVCG08-PuppoPanozzo.pdf.

[44] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part I): Data-Driven Solutions of Nonlinear Partial Differential Equations". In: (2017). URL: https://arxiv.org/abs/1711.10561.

[45] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006. ISBN: 026218253X. URL: http://www.GaussianProcess.org/gpml.

[46] H. E. Rauch, F. Tung, and C. T. Striebel. "Maximum Likelihood Estimates of Linear Dynamic Systems". In: *AIAA Journal* 3 (1965). DOI: 10.2514/3.3166.

[47] Danilo Jimenez Rezende and Shakir Mohamed. "Variational Inference with Normalizing Flows". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. PMLR, 2015.

[48] Carl David Tolmé Runge. "Über die numerische Auflösung von Differentialgleichungen". In: *Mathematische Annalen* (1895). DOI: 10.1007/BF01446807.

[49] Edmond Rusjan. *The Discrete Hodge Star Operator*. 2013. URL: https://web.williams.edu/Mathematics/sjmiller/public_html/hudson/EdmondRusjan-DiscreteHodgeStar.pdf.

[50] Simo Särkkä and Arno Solin. *Applied Stochastic Differential Equations*. Cambridge University Press, 2019.

[51] Rohan Sawhney et al. "Walk on Stars: A Grid-Free Monte Carlo Method for PDEs with Neumann Boundary Conditions". In: *ACM Trans. Graph.* 42.4 (2023).

[52] W. E. Schiesser. *The Numerical Method of Lines: Integration of Partial Differential Equations*. Elsevier, 2012.

[53] Robin M. Schmidt. "Recurrent Neural Networks (RNNs): A Gentle Introduction and Overview". In: (2019). URL: https://arxiv.org/abs/1912.05911.

[54] M. Schober, S. Särkkä, and P. Hennig. "A Probabilistic Model for the Numerical Solution of Initial Value Problems". In: *Statistics and Computing* (Jan. 2019). DOI: 10.1007/s11222-018-9822-z.

[55] Nicholas Sharp. *Intrinsic Triangulations in Geometry Processing*. Ph.D. Thesis. Computer Science Department, School of Computer Science, Carnegie Mellon University, 2021. URL: http://reports-archive.adm.cs.cmu.edu/anon/2021/CMU-CS-21-132.pdf.

[56] Nicholas Sharp and Keenan Crane. *A Laplacian for Nonmanifold Triangle Meshes*. Tech. rep. 5. Carnegie Mellon University, 2020. URL: https://www.cs.cmu.edu/~kmcrane/Projects/NonmanifoldLaplace/NonmanifoldLaplace.pdf.

[57] Nicholas Sharp et al. "DiffusionNet: Discretization Agnostic Learning on Surfaces". In: *ACM Trans. Graph.* XX.X (2022).

[58] Nathaniel Smith, Stefan van der Walt, et al. "The viridis Color Map". In: *Matplotlib documentation* (2015). URL: https://bids.github.io/colormap/.

[59] K. F. Tchon, J. Dompierre, M. G. Vallet, et al. "Two-Dimensional Metric Tensor Visualization Using Pseudo-Meshes". In: *Engineering with Computers* 22 (2006), pp. 121–131. DOI: 10.1007/s00366-006-0012-3. URL: https://doi.org/10.1007/s00366-006-0012-3.

[60] James Townsend, Niklas Koep, and Sebastian Weichwald. "Pymanopt: A Python Toolbox for Optimization on Manifolds using Automatic Differentiation". In: *Journal of Machine Learning Research* 17.137 (2016), pp. 1–5. URL: http://jmlr.org/papers/v17/16-177.html.

[61] Jonathan Wenger et al. "Posterior and Computational Uncertainty in Gaussian processes". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022. URL: https://arxiv.org/abs/2205.15449.

[62] Chris Yu, Henrik Schumacher, and Keenan Crane. "Repulsive Curves". In: *ACM Trans. Graph.* 40.2 (2021).