Stan Korea

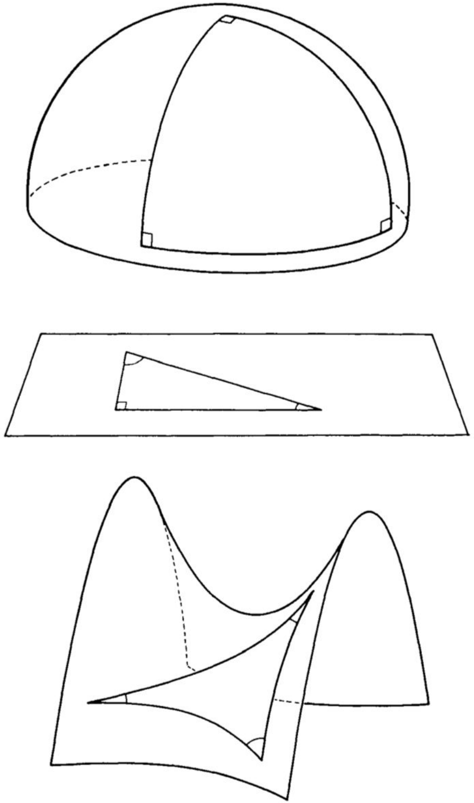
Hyperbolic and Manifold sampling

Hyunji Moon Industrial Engineering, SNU

CONTENTS

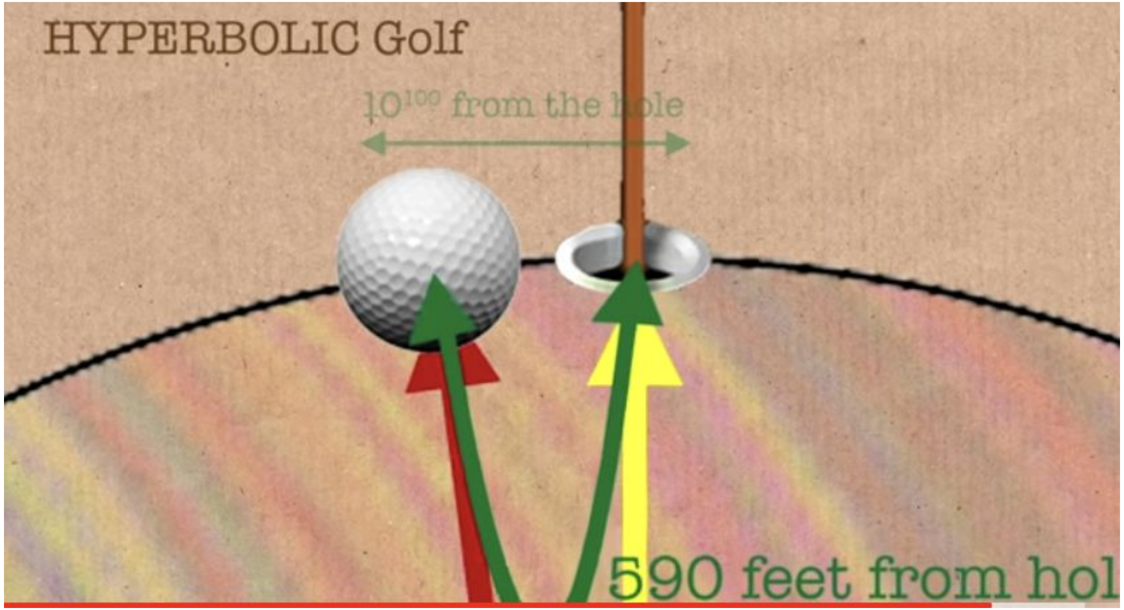
The shape of space (textbook)

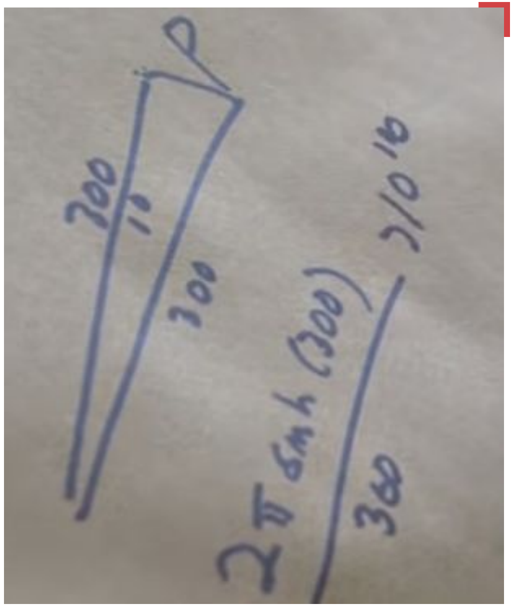
1. Topology vs. geometry
2. Intrinsic vs. extrinsic properties
3. Local vs. global properties
4. Homogeneous vs. nonhomogeneous geometries
5. Closed vs. open manifolds



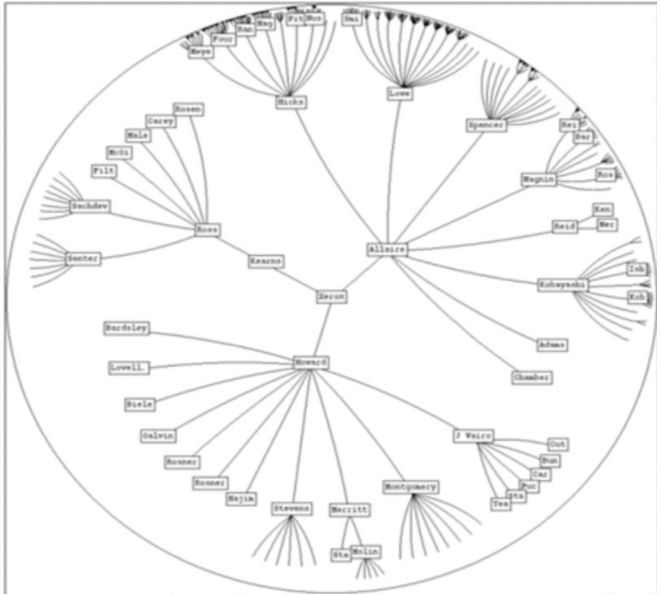
ure 3.7 The hemisphere, the plane, and the saddle surface all have different intrinsic geometries.

1. change X O when the surface is deformed
2. Flatlanders living in the surfaces (topologically) tell one from the other O X (diff. embedding)
3. small region vs mfd as a whole. local geometry global topology. flat torus and doughnut surface have the same global topology, but different local geometries.
4. local geometry is the same O X at all points. parameterization-invariant.
5. given no boundary (= edges?) + cpt vs non cpt component



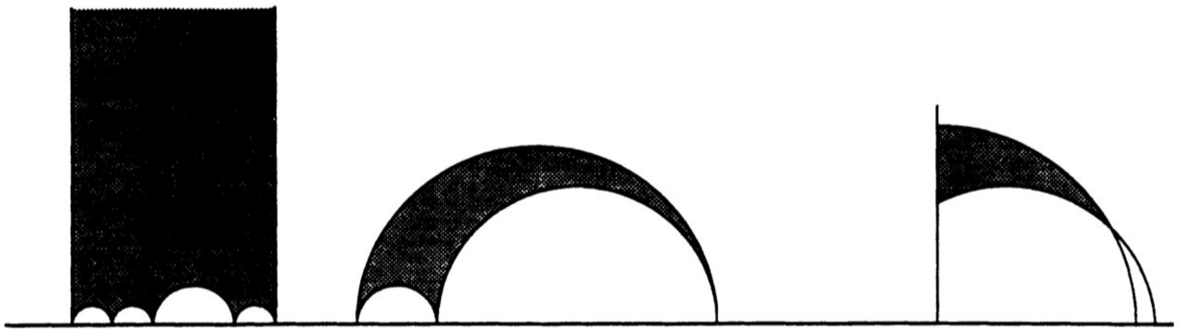


5.23 vs 10^100 (1 degree off in 300 radius)



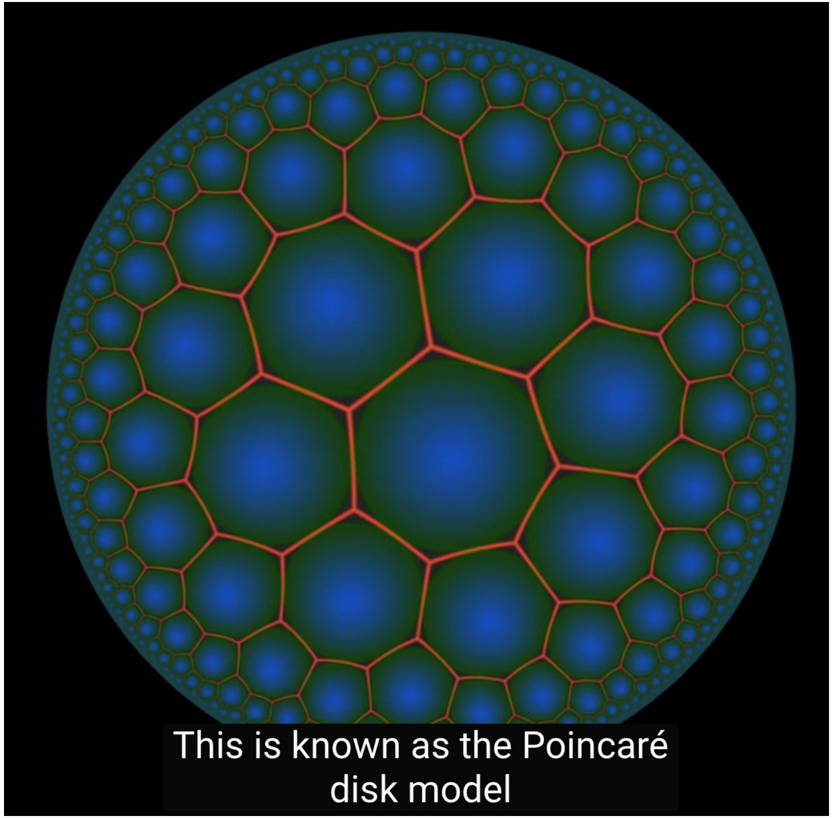
- Hyperbolic tree and geodesic
- Hyperbolic browser in information retrieval

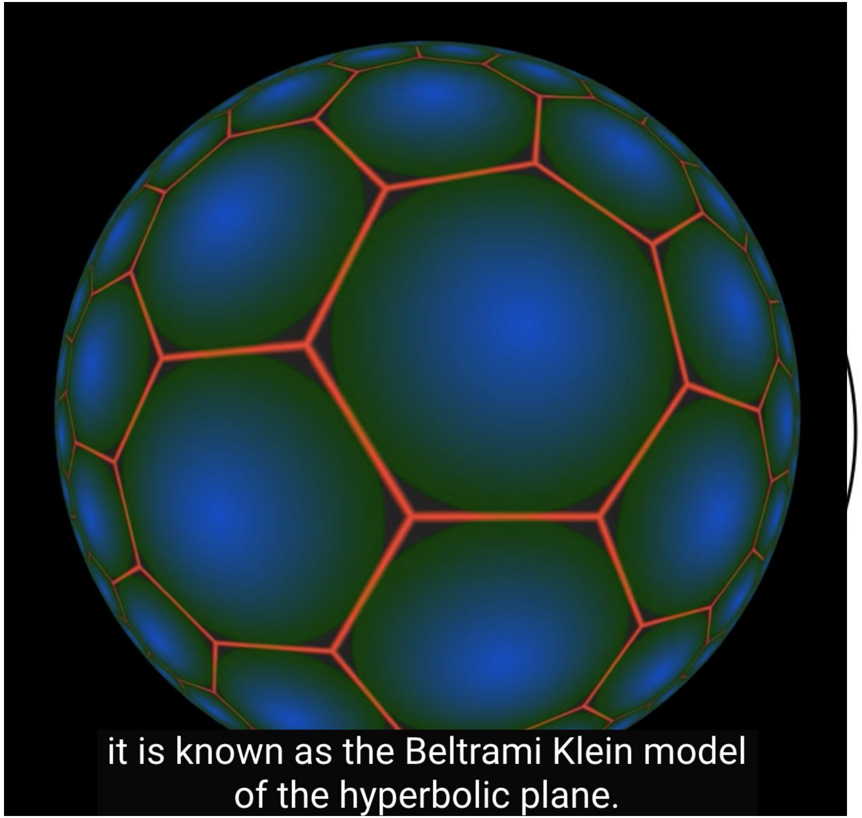
- upper plane: 2D Hyperbolic space in euclidean space
- most area of H space mapped into the region near real axis



an ideal hexagon　　an ideal triangle　　a triangle

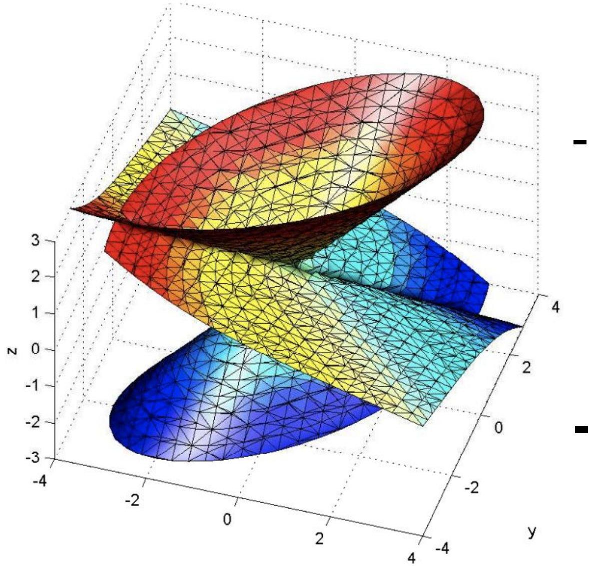
Two hyperbolic models





preserve straight line, not angle & area

Hyperbolic?



4xyz + xz2 + yz2 + 2z3 − x3 − 3zx2 − y3 − 3zy2 hyperbolic (w.r.t (0, 0, 1))

ref: exponential varieties

- **hyperbolic polynomial** p w.r.t e: p(e) > 0, t → p(x − te) has only real roots for all vector x ∈ Rn. can be generalized to all e.
  - t → (x1 − t)(x2 − t)· · ·(xn − t) w.r.t (1, … ,1)
- **hyperbolic cone**
  - from x, slide a $\Lambda_{++} := \{x \in \mathbb{R}^n : p(x - te) = 0 \ \Rightarrow \ t > 0\}$ meet p(x) = 0 vs opposite direction cross it n times
  - convex, convex optimization problems with nonnegative constraints
- **hyperbolic exponential family**: canonical parameters form a convex cone, partition (A: log partition) function is the power of a homogeneous polynomial
  - $p_\theta(x) \ = \ \exp(-\langle \theta, T(x) \rangle - A(\theta))$
  - eg. normal $A(\theta) \ = \ -\frac{1}{2}\log\det(\theta) + \frac{m}{2}\log(2\pi)$
  - construction of exponential families from complete hyperbolic polynomials

From (function, space) to (sampler, manifold)

$$\int_{\partial\Omega} \omega = \int_{\Omega} d\omega$$

Sampler

- MC, MCMC, ParVI, IS

Manifold

- Sphere, Euclidean, Hyperbolic
- metric: wasserstein space

better sampler while remain the space fixed?

Better sampler?

not just $\log\ p(\theta, x)$

Evaluation of the model on the unconstrained scale

$$p_Y(y) = p_X(f^{-1}(y)) \boxed{\left| \det J_{f^{-1}}(y) \right|}$$

need logdet calculation!

Better sampler

Sampling from a distribution supported on a manifold:

How to comply to the manifold geometry while being efficient?

Sampling on Probability Manifolds:

ParVIs have a natural optimization interpretation on a probability space.
When target measure is view as a point, we aim to **converge** to that point (measure) **efficiently**
-> need for **metric (wass.)**and **geodesic (least action among admissible path)**

$$d(x, y) = \sqrt{\inf_{\gamma_t : \gamma_0 = x, \gamma_1 = y} \int_0^1 \langle \dot{\gamma}_t, \dot{\gamma}_t \rangle_{T_{\gamma_t}\mathcal{M}} \, \mathrm{d}t}.$$

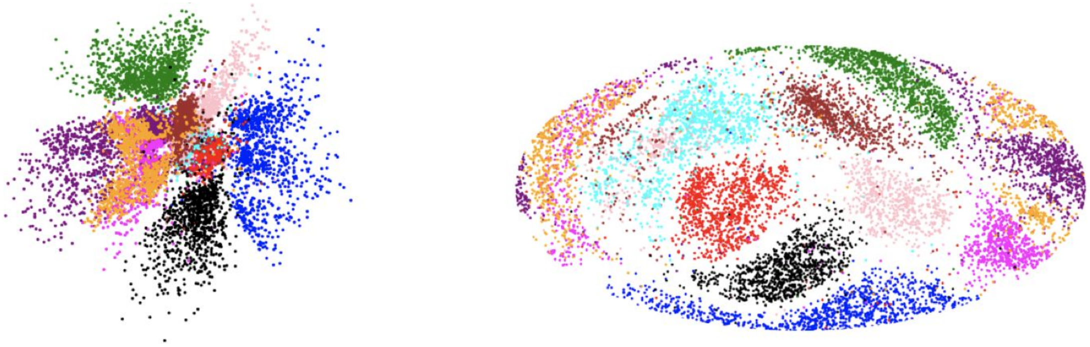
geodesic aka

- auto parallel curves under an affine connection (covariant derivative)
- generalized straight line

Stan
Korea

Hyperbolic Representation

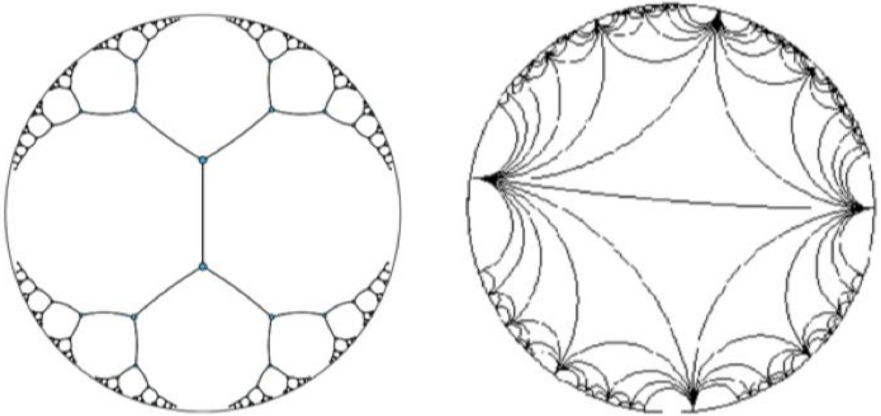
Better representation



(a) $\mathbb{R}^2$ latent space of the $\mathcal{N}$-VAE.

(b) Hammer projection of $\mathcal{S}^2$ latent space of the $\mathcal{S}$-VAE.

[minimum-distortion embedding]



Hyperbolic?
Tree, Hierarchical, Factorization [64, 66, 73] :
factor matrices on Stiefel manifold [68, 33]
ref from [this] lecturenote

$\{ M \in R^{m\times n} \mid M M^{\top} = I_m \}$.

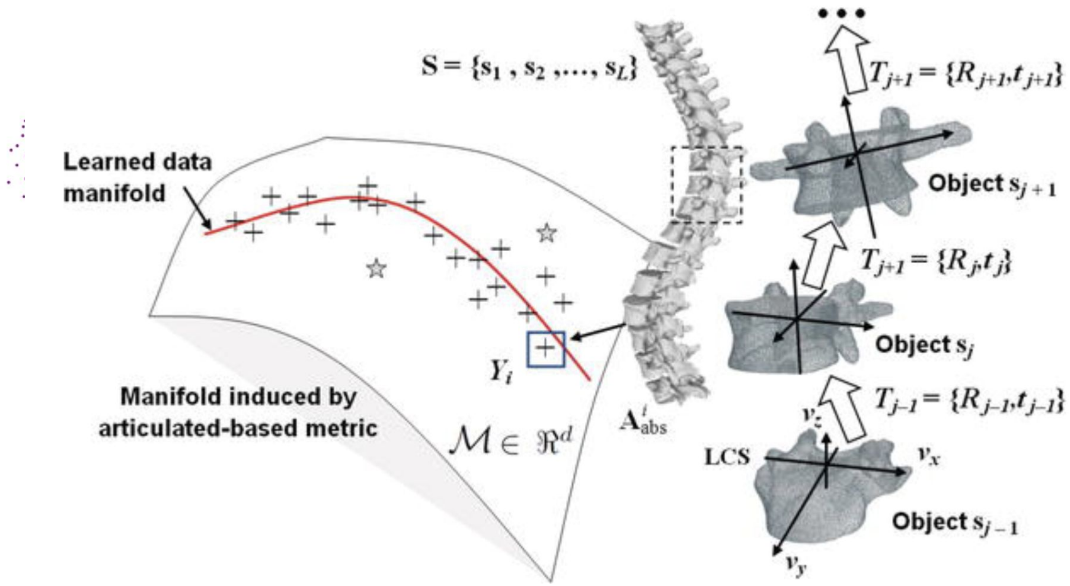
analogous hyperbolic latent space and tree structure

Hyperbolic and hierarchical structure : representation

Hyperbolic space is a geometry that is known to be well-suited for representation learning of data with an underlying hierarchical structure

- cognitive science: use a hierarchy to organise object categories
- biology: living organisms are related in a hierarchical manner given by the evolutionary tree

Better representation



- data manifold
- structured data (MRI, CT, ultrasound)
- linear techniques unsuitable for capturing variations in anatomical structures
- articulated structure metric on {s,R,t} scaling, rotation, translation

$$d_M\left(\mathbf{A}^i_{\text{abs}}, \mathbf{A}^j_{\text{abs}}\right) = \sum_{k=1}^{L} d_M\left(T^i_k, T^j_k\right) = \sum_{k=1}^{L} \| \mathbf{t}^i_k - \mathbf{t}^j_k \| + \sum_{k=1}^{L} d_G\left(\mathbf{R}^i_k, \mathbf{R}^j_k\right)$$

ref: https://www.intechopen.com/books/manifolds-ii-theory-and-applications/manifold-learning-in-medical-imaging

Stan
Korea

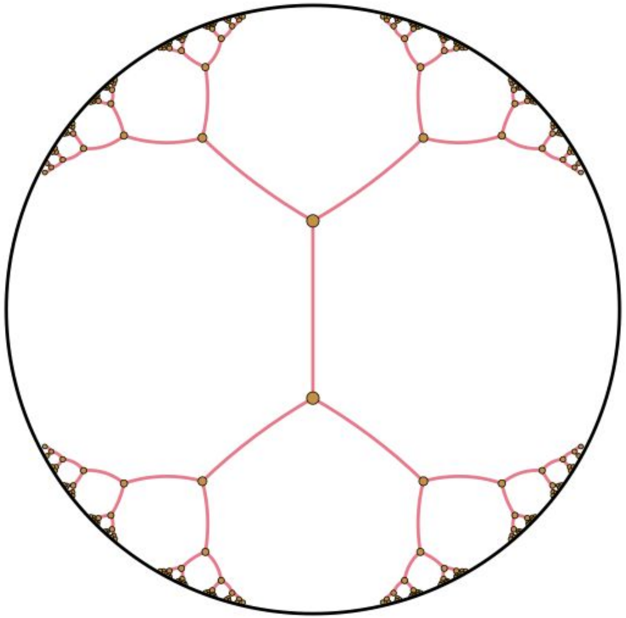
Implemented examples

Continuous Hierarchical Representations with Poincaré VAE

- hyperbolic spaces alternative continuous approach to learn hierarchical representations from textual graph-structured data
- continuous version of tree, smooth and differentiable
- endow VAEs with a Poincaré ball model of hyperbolic geometry as a latent space
- encoder: observation -> encoding (low dim latent space)
- decoder: encoding -> observation
- exponential growth of the Poincaré surface area with respect to its radius ~ exponential growth of the number of leaves in a tree with respect to its depth
- replace VAE’s latent space from Euclidean metric to hyperbolic
- beneficial in terms of model generalisation and can yield more interpretable representations

future work:

- best hyperbolic model for gradient-based learning
- principled way of assessing hierarchical data or not



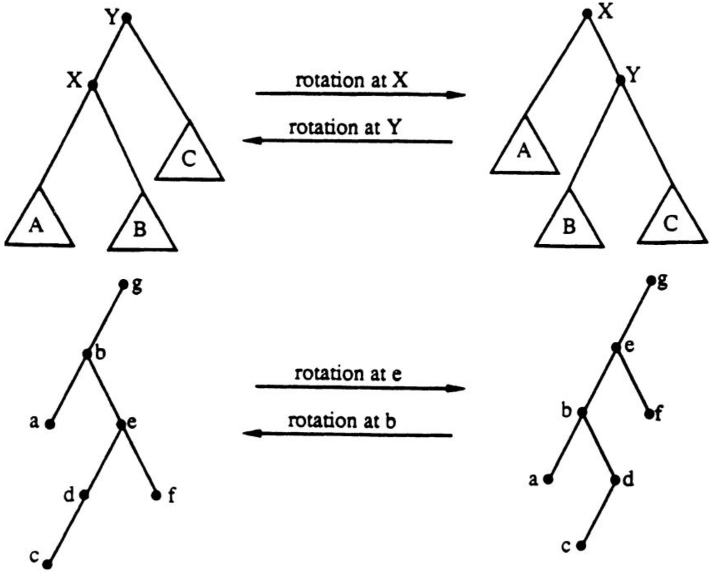
Poincare disc model embedding

Continuous-Time Birth-Death MCMC for Bayesian Regression Tree Models

- Bayesian additive regression trees sampling

- unlikely for a regression tree MCMC algorithm to fully explore the space of nearly equivalent trees that have high posterior probability

- generalization of rotation mechanism found in the binary search tree literature (Sleator88); Gramacy and Lee (2008) improve mixing of a Bayesian treed Gaussian Process model by applying the rotation algorithm from the Binary Search Tree literature

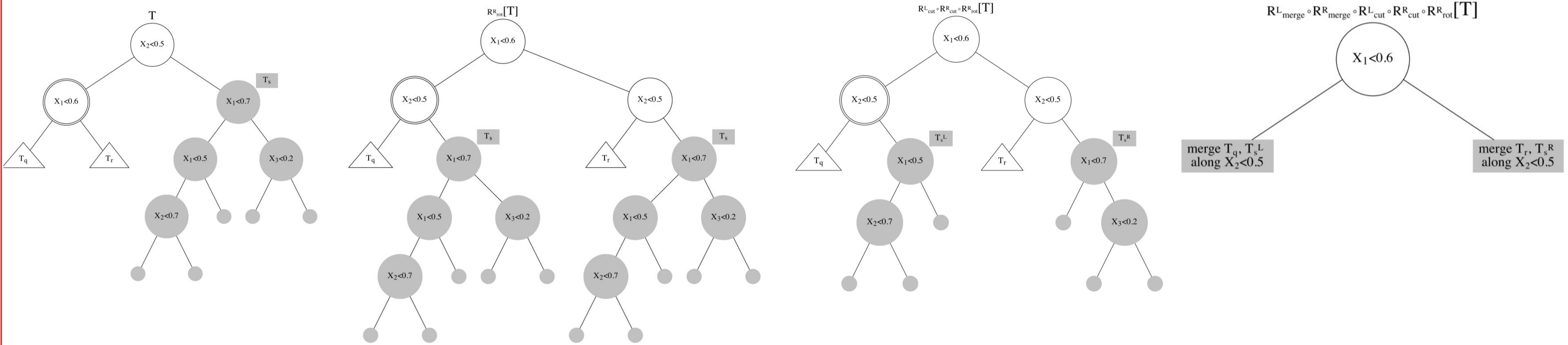


ROTATION DISTANCE, TRIANGULATIONS, AND HYPERBOLIC GEOMETRY

Stan Korea

Efficient Metropolis–Hastings Proposal Mechanisms
: propose rotate operation

$$\mathcal{R}[T] = \mathcal{R}^L_{merge}\mathcal{R}^R_{merge}\mathcal{R}^L_{cut}\mathcal{R}^R_{cut}\mathcal{R}^R_{rot}[T]$$



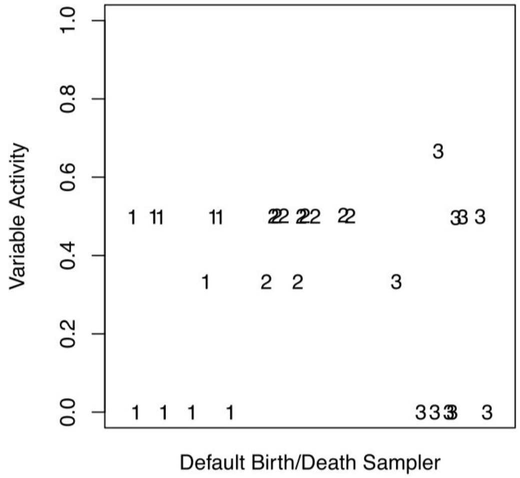
rotation for T to appear on both sides

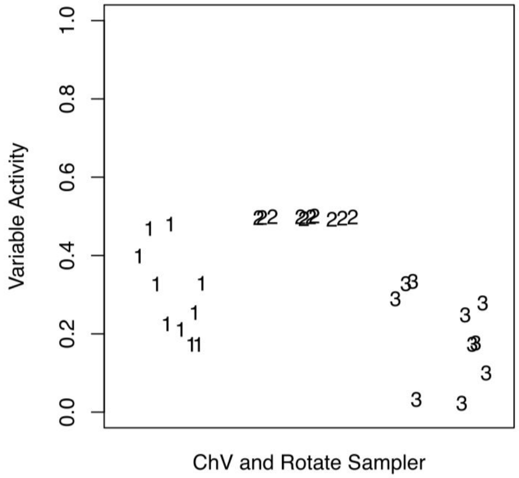
both copies sub-tree cut along the X1 < 0.6 rule -> sub-trees

merged along the rule X2 < 0.5.

Efficient Metropolis–Hastings Proposal Mechanisms

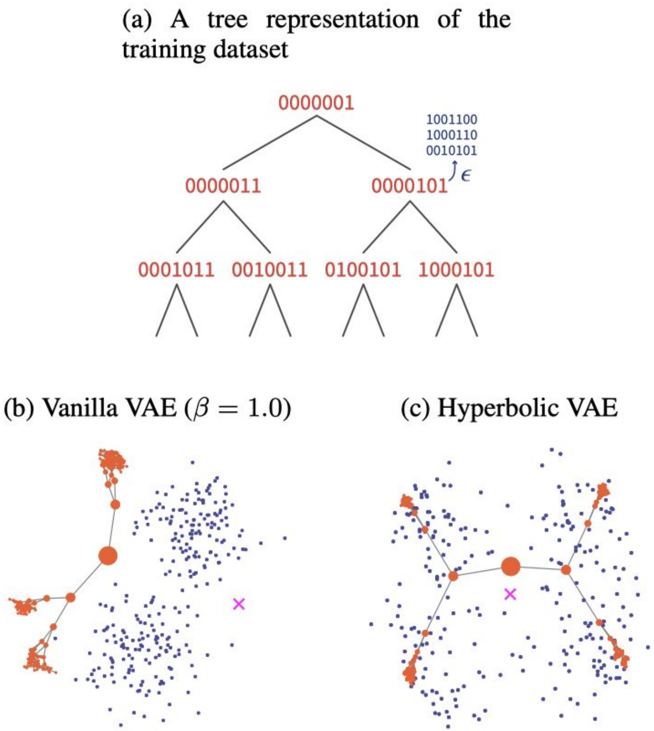
- more continuous and detect important variable (1, 3)
- discretized variable importance for 10 cv sets





Stan Korea

[Wrapped Normal Distribution on Hyperbolic Space for Gradient-Based Learning](#)



(a) A tree representation of the training dataset

(b) Vanilla VAE ($\beta = 1.0$)

(c) Hyperbolic VAE

need:
probability distributions on H that admit parametrization of density function that can be computed analytically and differentiated

how:

1) defining Gaussian distribution on the tangent space at the origin of the hyperbolic space
2) transporting the tangent space to a desired location in the space
3) projecting the distribution onto hyperbolic space

Stan

Wrapped Normal Distribution on Hyperbolic Space for Gradient-Based Learning

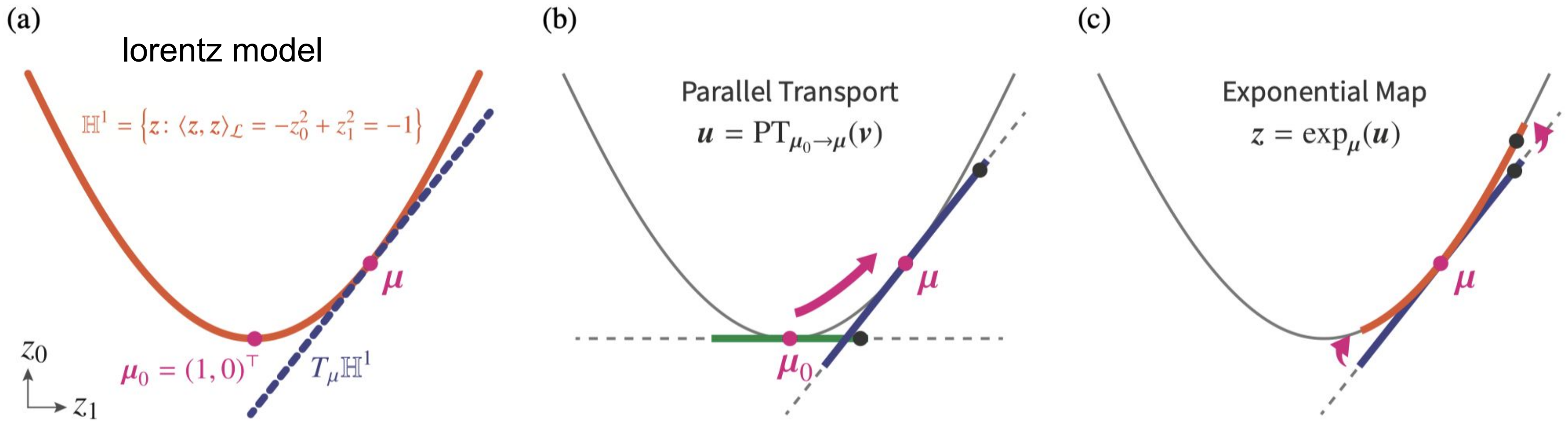


Figure 2: (a) One-dimensional Lorentz model $\mathbb{H}^1$ (red) and its tangent space $T_{\boldsymbol{\mu}}\mathbb{H}^1$ (blue). (b) Parallel transport carries $\boldsymbol{v} \in T_{\boldsymbol{\mu}_0}$ (green) to $\boldsymbol{u} \in T_{\boldsymbol{\mu}}$ (blue) while preserving $\|\cdot\|_{\mathcal{L}}$. (c) Exponential map projects the $\boldsymbol{u} \in T_{\boldsymbol{\mu}}$ (blue) to $\boldsymbol{z} \in \mathbb{H}^n$ (red). The distance between $\boldsymbol{\mu}$ and $\exp_{\boldsymbol{\mu}}(\boldsymbol{u})$ which is measured on the surface of $\mathbb{H}^n$ coincides with $\|\boldsymbol{u}\|_{\mathcal{L}}$.

$$\mathbb{H}^n = \{\boldsymbol{z} \in \mathbb{R}^{n+1} : \langle \boldsymbol{z}, \boldsymbol{z}\rangle_{\mathcal{L}} = -1,\ \ z_0 > 0\}$$

$$\langle \boldsymbol{z}, \boldsymbol{z}'\rangle_{\mathcal{L}} = -z_0 z_0' + \sum_{i=1} z_i z_i',$$

$$\boldsymbol{z} = \exp_{\boldsymbol{\mu}}(\boldsymbol{u}) = \cosh(\|\boldsymbol{u}\|_{\mathcal{L}})\boldsymbol{\mu} + \sinh(\|\boldsymbol{u}\|_{\mathcal{L}})\frac{\boldsymbol{u}}{\|\boldsymbol{u}\|_{\mathcal{L}}}$$

Wrapped Normal Distribution on Hyperbolic Space for Gradient-Based Learning

Algorithm 1 Sampling on hyperbolic space

Input: parameter $\boldsymbol{\mu} \in \mathbb{H}^n$, Σ
Output: $\boldsymbol{z} \in \mathbb{H}^n$
Require: $\boldsymbol{\mu}_0 = (1, 0, \cdots, 0)^\top \in \mathbb{H}^n$
Sample $\tilde{\boldsymbol{v}} \sim \mathcal{N}(\mathbf{0}, \Sigma) \in \mathbb{R}^n$
$\boldsymbol{v} = [0, \tilde{\boldsymbol{v}}] \in T_{\boldsymbol{\mu}_0}\mathbb{H}^n$
Move $\boldsymbol{v}$ to $\boldsymbol{u} = \mathrm{PT}_{\boldsymbol{\mu}_0 \to \boldsymbol{\mu}}(\boldsymbol{v}) \in T_{\boldsymbol{\mu}}\mathbb{H}^n$ by eq. (3)
Map $\boldsymbol{u}$ to $\boldsymbol{z} = \exp_{\boldsymbol{\mu}}(\boldsymbol{u}) \in \mathbb{H}^n$ by eq. (5)

Algorithm 2 Calculate log-pdf

Input: sample $\boldsymbol{z} \in \mathbb{H}^n$, parameter $\boldsymbol{\mu} \in \mathbb{H}^n$, Σ
Output: $\log p(\boldsymbol{z})$
Require: $\boldsymbol{\mu}_0 = (1, 0, \cdots, 0)^\top \in \mathbb{H}^n$
Map $\boldsymbol{z}$ to $\boldsymbol{u} = \exp_{\boldsymbol{\mu}}^{-1}(\boldsymbol{z}) \in T_{\boldsymbol{\mu}}\mathbb{H}^n$ by eq. (6)
Move $\boldsymbol{u}$ to $\boldsymbol{v} = \mathrm{PT}_{\boldsymbol{\mu}_0 \to \boldsymbol{\mu}}^{-1}(\boldsymbol{u}) \in T_{\boldsymbol{\mu}_0}\mathbb{H}^n$ by eq. (4)
Calculate $\log p(\boldsymbol{z})$ by eq. (7)

$$\det\left(\frac{\partial\, \mathrm{proj}_{\boldsymbol{\mu}}(\boldsymbol{v})}{\partial \boldsymbol{v}}\right) = \det\left(\frac{\partial \exp_{\boldsymbol{\mu}}(\boldsymbol{u})}{\partial \boldsymbol{u}}\right) \cdot \det\left(\frac{\partial\, \mathrm{PT}_{\boldsymbol{\mu}_0 \to \boldsymbol{\mu}}(\boldsymbol{v})}{\partial \boldsymbol{v}}\right)$$

$$= \left(\frac{\sinh r}{r}\right)^{n-1}$$

= 1 (norm preserving)

Stan Korea

Code corner

develop · stan / src / stan / mcmc / hmc /

rok-cesnovar fix double include

..	
hamiltonians	fix double include
integrators	cleanup includes of Eigen/*
nuts	
nuts_classic	
static	
static_uniform	
xhmc	
base_hmc.hpp	

develop · stan / src / stan / mcmc / hmc / integrators /

rok-cesnovar cleanup includes of Eigen/*

..	
base_integrator.hpp	Revert
base_leapfrog.hpp	Revert
expl_leapfrog.hpp	fix inclu
impl_leapfrog.hpp	cleanup includes of Eigen/*

develop · stan / src / stan / mcmc / hmc / hamiltonians /

rok-cesnovar fix double include

..	
base_hamiltonian.hpp	cleanup i
dense_e_metric.hpp	cleanup i
dense_e_point.hpp	removed
diag_e_metric.hpp	Revert "R
diag_e_point.hpp	removed
ps_point.hpp	fix double
softabs_metric.hpp	Fix mix in
softabs_point.hpp	Revert "R
unit_e_metric.hpp	Revert "R
unit_e_point.hpp	Revert "R

Stan Korea

Code corner: expl vs impl

$$H(\boldsymbol{\omega}, \mathbf{p}) = -\log\left(f(\boldsymbol{\omega})\right) + \frac{1}{2}\log\left((2\pi)^D|\mathbf{M}|\right) + \frac{1}{2}\mathbf{p}^\top\mathbf{M}^{-1}\mathbf{p}.$$

$$\tilde{H}(\boldsymbol{\omega}, \mathbf{p}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{p}}) = H_1(\boldsymbol{\omega}, \tilde{\mathbf{p}}) + H_2(\tilde{\boldsymbol{\omega}}, \mathbf{p}) + \Omega h(\boldsymbol{\omega}, \mathbf{p}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{p}})$$

Algorithm 1 Implicit Leapfrog Step

Inputs: $\mathbf{p}_0, \boldsymbol{\omega}_0, \epsilon,$
$\mathbf{p} = \mathbf{p}_0$
while $\Delta p > \delta$ **do**
  $\mathbf{p}' = \mathbf{p}_0 + \frac{\epsilon}{2}\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}\tau}(\mathbf{p}, \boldsymbol{\omega}_0)$
  $\Delta p = \max_i\{|p_i - p_i'|\}$
  $\mathbf{p} = \mathbf{p}'$
end while
$\boldsymbol{\omega} = \boldsymbol{\omega}_0$
while $\Delta\omega > \delta$ **do**
  $\boldsymbol{\omega}' = \boldsymbol{\omega}_0 + \frac{\epsilon}{2}\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}\tau}(\mathbf{p}, \boldsymbol{\omega}_0) + \frac{\epsilon}{2}\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}\tau}(\boldsymbol{p}, \boldsymbol{\omega})$
  $\Delta\omega = \max_i\{|\omega_i - \omega_i'|\}$
  $\boldsymbol{\omega} = \boldsymbol{\omega}'$
end while
$\mathbf{p} = \mathbf{p} + \frac{\epsilon}{2}\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}\tau}(\mathbf{p}, \boldsymbol{\omega})$

- computationally expensive
- implicit (fixed point)

first-order Euler integrators
run until convergence

Algorithm 2 Explicit Leapfrog Step

Inputs: $\mathbf{p}, \boldsymbol{\omega}, \tilde{\mathbf{p}}, \tilde{\boldsymbol{\omega}}, \epsilon, \Omega$
$\mathbf{p} = \mathbf{p} - \frac{\epsilon}{2}\partial_{\boldsymbol{\omega}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}})$
$\tilde{\boldsymbol{\omega}} = \tilde{\boldsymbol{\omega}} + \frac{\epsilon}{2}\partial_{\tilde{\mathbf{p}}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}})$
$\tilde{\mathbf{p}} = \tilde{\mathbf{p}} - \frac{\epsilon}{2}\partial_{\tilde{\boldsymbol{\omega}}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p})$
$\boldsymbol{\omega} = \boldsymbol{\omega} + \frac{\epsilon}{2}\partial_{\mathbf{p}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p})$
$c = \cos(2\Omega\epsilon), \quad s = \sin(2\Omega\epsilon)$
$\boldsymbol{\omega} = (\boldsymbol{\omega} + \tilde{\boldsymbol{\omega}} + c(\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}) + s(\mathbf{p} - \tilde{\mathbf{p}}))/2$
$\mathbf{p} = (\mathbf{p} + \tilde{\mathbf{p}} - s(\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}) + c(\mathbf{p} - \tilde{\mathbf{p}}))/2$
$\tilde{\boldsymbol{\omega}} = (\boldsymbol{\omega} + \tilde{\boldsymbol{\omega}} - c(\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}) - s(\mathbf{p} - \tilde{\mathbf{p}}))/2$
$\tilde{\mathbf{p}} = (\mathbf{p} + \tilde{\mathbf{p}} + s(\boldsymbol{\omega} - \tilde{\boldsymbol{\omega}}) - c(\mathbf{p} - \tilde{\mathbf{p}}))/2$
$\tilde{\mathbf{p}} = \tilde{\mathbf{p}} - \frac{\epsilon}{2}\partial_{\tilde{\boldsymbol{\omega}}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p})$
$\boldsymbol{\omega} = \boldsymbol{\omega} + \frac{\epsilon}{2}\partial_{\mathbf{p}} H(\tilde{\boldsymbol{\omega}}, \mathbf{p})$
$\mathbf{p} = \mathbf{p} - \frac{\epsilon}{2}\partial_{\boldsymbol{\omega}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}})$
$\tilde{\boldsymbol{\omega}} = \tilde{\boldsymbol{\omega}} + \frac{\epsilon}{2}\partial_{\tilde{\mathbf{p}}} H(\boldsymbol{\omega}, \tilde{\mathbf{p}})$

Code corner: expl vs impl

```cpp
class expl_leapfrog : public base_leapfrog<Hamiltonian> {
 public:
  expl_leapfrog() : base_leapfrog<Hamiltonian>() {}

  void begin_update_p(typename Hamiltonian::PointType& z,
                      Hamiltonian& hamiltonian, double epsilo
                      callbacks::logger& logger) {
    z.p -= epsilon * hamiltonian.dphi_dq(z, logger);
  }

  void update_q(typename Hamiltonian::PointType& z, Hamiltoni
                double epsilon, callbacks::logger& logger) {
    z.q += epsilon * hamiltonian.dtau_dp(z);
    hamiltonian.update_potential_gradient(z, logger);
  }

  void end_update_p(typename Hamiltonian::PointType& z,
                    Hamiltonian& hamiltonian, double epsilon,
                    callbacks::logger& logger) {
    z.p -= epsilon * hamiltonian.dphi_dq(z, logger);
  }
};
```

implicit: computationally expensive, first-order implicit Euler integrators run until fixed-point iterations run until convergence

```cpp
class impl_leapfrog : public base_leapfrog<Hamiltonian> {
 public:
  impl_leapfrog()
      : base_leapfrog<Hamiltonian>(),
        max_num_fixed_point_(10),
        fixed_point_threshold_(1e-8) {}

  void begin_update_p(typename Hamiltonian::PointType& z,
                      Hamiltonian& hamiltonian, double epsilon,
                      callbacks::logger& logger) {
    hat_phi(z, hamiltonian, epsilon, logger);
    hat_tau(z, hamiltonian, epsilon, this->max_num_fixed_point_, logger);
  }

  void update_q(typename Hamiltonian::PointType& z, Hamiltonian& hamiltonian
                double epsilon, callbacks::logger& logger) {
    // hat{T} = dT/dp * d/dq
    Eigen::VectorXd q_init = z.q + 0.5 * epsilon * hamiltonian.dtau_dp(z);
    Eigen::VectorXd delta_q(z.q.size());

    for (int n = 0; n < this->max_num_fixed_point_; ++n) {
      delta_q = z.q;
      z.q.noalias() = q_init + 0.5 * epsilon * hamiltonian.dtau_dp(z);
      hamiltonian.update_metric(z, logger);

      delta_q -= z.q;
      if (delta_q.cwiseAbs().maxCoeff() < this->fixed_point_threshold_)
        break;
    }
    hamiltonian.update_gradients(z, logger);
  }

  void end_update_p(typename Hamiltonian::PointType& z,
                    Hamiltonian& hamiltonian, double epsilon,
                    callbacks::logger& logger) {
    hat_tau(z, hamiltonian, epsilon, 1, logger);
    hat_phi(z, hamiltonian, epsilon, logger);
  }
```

```cpp
// hat{phi} = dphi/dq * d/dp
void hat_phi(typename Hamiltonian::PointType& z, Hamiltonian& hamiltonian,
             double epsilon, callbacks::logger& logger) {
  z.p -= epsilon * hamiltonian.dphi_dq(z, logger);
}

// hat{tau} = dtau/dq * d/dp
void hat_tau(typename Hamiltonian::PointType& z, Hamiltonian& hamiltonian,
             double epsilon, int num_fixed_point, callbacks::logger& logger) {
  Eigen::VectorXd p_init = z.p;
  Eigen::VectorXd delta_p(z.p.size());

  for (int n = 0; n < num_fixed_point; ++n) {
    delta_p = z.p;
    z.p.noalias() = p_init - epsilon * hamiltonian.dtau_dq(z, logger);
    delta_p -= z.p;
    if (delta_p.cwiseAbs().maxCoeff() < this->fixed_point_threshold_)
      break;
  }
}

int max_num_fixed_point() { return this->max_num_fixed_point_; }

void set_max_num_fixed_point(int n) {
  if (n > 0)
    this->max_num_fixed_point_ = n;
}

double fixed_point_threshold() { return this->fixed_point_threshold_; }

void set_fixed_point_threshold(double t) {
  if (t > 0)
    this->fixed_point_threshold_ = t;
}
```

Thank You.

Stan
Korea