

# Other Models

Muchang Bahng

Spring 2025

## Contents

<b>1</b>	<b>Topological Data Analysis</b>	<b>3</b>
<b>2</b>	<b>Geodesic Regression</b>	<b>4</b>
2.1	Multiple Geodesic Regression . . . . .	8
2.2	Robust Geodesic Regression . . . . .	8
<b>3</b>	<b>Frechet Regression</b>	<b>9</b>
	<b>References</b>	<b>10</b>

Just some other models I've learned that don't fit in nicely to any of existing categories yet.

# 1 Topological Data Analysis

## 2 Geodesic Regression

In regression, note that we are finding a function  $f : \mathcal{X} \rightarrow \mathcal{Y}$ . In usual linear regression, both  $\mathcal{X}, \mathcal{Y}$  are Euclidean space. However, there are cases where it may not be realistic that one or more of them should be modeled as a vector space. Rather, they may be part of a lower-dimensional manifold. For instance, if we want to use linear regression to predict the top  $k$  principal components of a dataset, they must be orthogonal, i.e. must be in a *Stiefl manifold*.

There are way to model this. For instance, we could have a projection operator that maps from  $\mathbb{R}^m \rightarrow \mathcal{Y}$ . This has its issues too, for one not being very efficient since perhaps the dimension of  $\mathcal{Y}$  may be much less than  $m$ . Therefore, it might be better to directly regress onto a manifold. There were many attempts at this, but the first model to generalize OLS to manifolds was created by Fletcher in 2011 [Fle11] and expanded shortly in [TF13].

We start with the case where there is one covariate (i.e.  $\mathcal{X} = \mathbb{R}$ ) and  $\mathcal{Y} = (M, d)$  is a smooth Riemannian manifold with a metric. Recall that for a smooth manifold  $M$ , for any  $p \in M$  and  $v \in T_p M$ , the tangent space at  $p$ , there is a unique geodesic curve  $\gamma : [0, 1] \rightarrow M$  satisfying  $\gamma(0) = p$ ,  $\gamma'(0) = v$ . This geodesic is guaranteed to exist locally, and with this, we can define the exponential map at  $p$  in the direction of  $v$  as

$$\exp_p(v) = \exp(p, v) = \gamma(1) \quad (1)$$

In other words, the exponential map takes a position and velocity as input and returns the point at time 1 along the geodesic with these initial conditions. With this motivation, we use slightly different notation than regular linear regression, referring  $p$  as the bias and  $v$  as the coefficient.

### Definition 2.1 (Geodesic Regression)

The **geodesic regression** model is a probabilistic model that predicts the conditional distribution of  $y \in (M, d)$  given  $x \in \mathbb{R}$  as

$$y = \exp(\exp(p, vx), \epsilon) \quad (2)$$

where the parameters are  $\theta = \{p, v\}$ , and  $\epsilon$  is a random variable defined over the tangent space at  $\exp(p, vx)$ .

Note that if we set  $\mathcal{Y} = \mathbb{R}^m$ , then we get the ordinary linear regression model back.

### Definition 2.2 (Least Squares Geodesic Regression)

The least squares geodesic regression aims to minimize the MSE loss

$$L(\theta, (x, y)) = L(p, v, x, y) = d(\exp(p, vx), y)^2 \quad (3)$$

### Lemma 2.1 (Risk)

The risk is

$$R(f) = \mathbb{E}_{x,y} [d(\exp(p, vx), y)^2] \quad (4)$$

and the empirical risk for a dataset  $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$  is

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^n d(\exp(p, vx^{(i)}), y^{(i)})^2 \quad (5)$$

Unfortunately, minimizing this does not yield an analytic solution.

**Example 2.1 (Code Walkthrough)**

Let us fit a line onto this. We first define our manifold class with the matrix exponential and logarithm methods.

```

1  class S2:
2      @staticmethod
3      def exp(p, v):
4          v_norm = np.linalg.norm(v)
5          if v_norm < 1e-8:
6              return p
7          return np.cos(v_norm) * p + np.sin(v_norm) * v / v_norm
8
9      @staticmethod
10     def log(p, q):
11         cos_dist = np.clip(np.dot(p, q), -1, 1)
12         if np.abs(cos_dist - 1) < 1e-8:
13             return np.zeros_like(p)
14
15         theta = np.arccos(cos_dist)
16         sin_theta = np.sin(theta)
17
18         if sin_theta < 1e-8:
19             return np.zeros_like(p)
20
21         return theta * (q - cos_dist * p) / sin_theta
22
23     @staticmethod
24     def distance(p, q):
25         cos_dist = np.clip(np.dot(p, q), -1, 1)
26         return np.arccos(cos_dist)
27
28     @staticmethod
29     def project_to_tangent(p, v):
30         return v - np.dot(v, p) * p
31
32     @staticmethod
33     def normalize(x):
34         return x / np.linalg.norm(x)

```

Next, we define our data generation process.

```

1  def generate_sample_data(n_samples=50, noise_level=0.1):
2      X = np.random.uniform(-2, 2, n_samples)
3
4      p_true = S2.normalize(np.array([1, 0, 0]))
5      v_true = S2.normalize(np.array([0, 1, 0.2]))
6      v_true = S2.project_to_tangent(p_true, v_true) * 0.5
7
8      Y = []
9      for x in X:
10         y_clean = S2.exp(p_true, v_true * x)
11
12         noise = np.random.normal(0, noise_level, 3)
13         noise = S2.project_to_tangent(y_clean, noise)
14         y_noisy = S2.exp(y_clean, noise)
15         y_noisy = S2.normalize(y_noisy)

```

```

16
17     Y.append(y_noisy)
18
19     return X, np.array(Y), p_true, v_true

```

Finally, we define our regression model and optimize the loss with BFGS (SGD doesn't work very well here).

```

1  class GeodesicRegression:
2      def __init__(self):
3          self.p = None
4          self.v = None
5
6      def _geodesic_point(self, p, v, x):
7          return S2.exp(p, v * x)
8
9      def _objective(self, params, X, Y):
10         p_flat = params[:3]
11         v_flat = params[3:6]
12
13         p_flat = S2.normalize(p_flat)
14         v_flat = S2.project_to_tangent(p_flat, v_flat)
15
16         total_loss = 0.0
17         for i in range(len(X)):
18             pred = self._geodesic_point(p_flat, v_flat, X[i])
19             loss = S2.distance(pred, Y[i])**2
20             total_loss += loss
21
22         return total_loss / len(X)
23
24     def fit(self, X, Y, p_init=None, v_init=None, method='BFGS'):
25         X = np.array(X)
26         Y = np.array(Y)
27
28         if p_init is None:
29             p_init = S2.normalize(np.array([1, 0, 0]))
30         if v_init is None:
31             v_init = np.array([0, 0.1, 0])
32
33         v_init = S2.project_to_tangent(p_init, v_init)
34
35         initial_params = np.concatenate([p_init, v_init])
36
37         result = minimize(
38             self._objective,
39             initial_params,
40             args=(X, Y),
41             method=method,
42             options={'disp': False}
43         )
44
45         if result.success:
46             self.p = S2.normalize(result.x[:3])
47             self.v = S2.project_to_tangent(self.p, result.x[3:6])
48             return result
49         else:

```

```

50         raise RuntimeError(f"Optimization failed: {result.message}")
51
52     def predict(self, X):
53         X = np.array(X)
54         predictions = []
55
56         for x in X:
57             pred = self._geodesic_point(self.p, self.v, x)
58             predictions.append(pred)
59
60         return np.array(predictions)
61
62     def score(self, X, Y):
63         predictions = self.predict(X)
64         total_loss = 0.0
65
66         for i in range(len(Y)):
67             loss = S2.distance(predictions[i], Y[i])**2
68             total_loss += loss
69
70         return total_loss / len(Y)
71
72 X_train, Y_train, p_true, v_true = generate_sample_data(n_samples=30)
73 model = GeodesicRegression()
74 result = model.fit(X_train, Y_train)

```

This gives the following, which is a good estimate of the original parameters.

$$\hat{p} = \begin{pmatrix} 0.99993911 \\ -0.01038634 \\ -0.00372747 \end{pmatrix} \approx \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = p_{\text{true}}, \quad \hat{v} = \begin{pmatrix} 0.00530609 \\ 0.48332324 \\ 0.07667702 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0.49029034 \\ 0.09805807 \end{pmatrix} = v_{\text{true}} \quad (6)$$

The following figure also visualizes this.

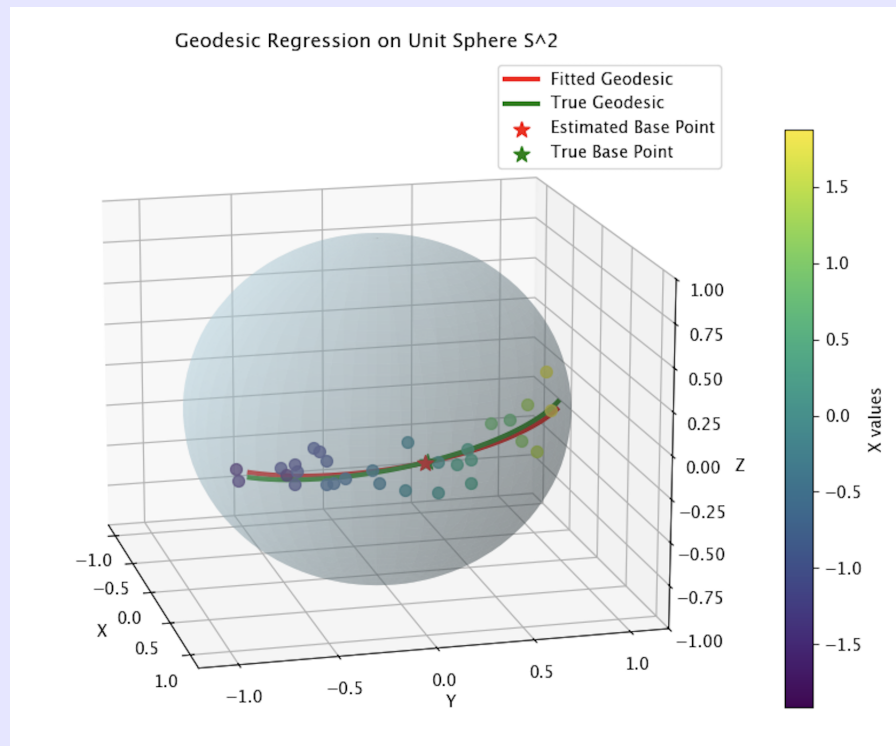


Figure 1

## 2.1 Multiple Geodesic Regression

Note that this was a model for a path in some manifold, and naturally we would like to extend this to have multiple covariates. Kim in 2014 did exactly that, and provided a framework for multivariate general linear models in [KAC<sup>+</sup>14].

## 2.2 Robust Geodesic Regression



### 3 Frechet Regression

## References

- [Fle11] Thomas Fletcher. Geodesic Regression on Riemannian Manifolds. In Pennec, Xavier, Joshi, Sarang, Nielsen, and Mads, editors, *Proceedings of the Third International Workshop on Mathematical Foundations of Computational Anatomy - Geometrical and Statistical Methods for Modelling Biological Shape Variability*, pages 75–86, Toronto, Canada, September 2011.
- [KAC<sup>+</sup>14] Hyunwoo J. Kim, Nagesh Adluru, Maxwell D. Collins, Moo K. Chung, Barbara B. Bendin, Sterling C. Johnson, Richard J. Davidson, and Vikas Singh. Multivariate general linear models (mgm) on riemannian manifolds with applications to statistical analysis of diffusion weighted images. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2705–2712, 2014.
- [TF13] P. Thomas Fletcher. Geodesic regression and the theory of least squares on riemannian manifolds. *Int. J. Comput. Vision*, 105(2):171–185, November 2013.