GAUSSIAN PROCESS FOR TRAJECTORIES

Kien Nguyen
Computer Science Department
University of Southern California
Los Angeles, CA 90007
kien.nguyen@usc.edu

John Krumm
Microsoft Research
Microsoft Corporation
Redmond, WA 98052
jckrumm@microsoft.com

Cyrus Shahabi
Computer Science Department
University of Southern California
Los Angeles, CA 90007
shahabi@usc.edu

ABSTRACT

The Gaussian process is a powerful and flexible technique for interpolating spatiotemporal data, especially with its ability to capture complex trends and uncertainty from the input signal. This chapter describes Gaussian processes as an interpolation technique for geospatial trajectories. A Gaussian process models measurements of a trajectory as coming from a multidimensional Gaussian, and it produces for each timestamp a Gaussian distribution as a prediction. We discuss elements that need to be considered when applying Gaussian process to trajectories, common choices for those elements, and provide a concrete example of implementing a Gaussian process.

***K*eywords** Gaussian process · trajectory interpolation · spatiotemporal data

1 Introduction

The availability of devices with location tracking capability has helped generate a tremendous amount of trajectory data of humans, animals, vehicles, and drones, which is valuable in various applications [1]. However, trajectories may have missing locations between their measurements and uncertainty in the measurements depending on how they were captured. Interpolating missing locations between measurements, or making predictions about future locations, is often necessary in order to derive high utility from trajectories. This work presents Gaussian Process (GP) as a powerful and flexible technique for trajectory interpolation and prediction.

A trajectory S is defined as a sequence of time-ordered noisy location measurements $S = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{|S|}\}$, where each measurement or data point $\boldsymbol{x}$ (bold symbol) includes $\boldsymbol{x}.lon$ and $\boldsymbol{x}.lat$ as the longitude and latitude, $\boldsymbol{x}.t$ as the timestamp, and $\boldsymbol{x}.\sigma$ as the accuracy or uncertainty. Previous work [2] showed that it is reasonable to assume Gaussian noise for location measurements such as GPS points. Therefore, $\boldsymbol{x}.\sigma$ can be considered as the standard deviation of independent Gaussian noise of longitude and latitude. Figure 1 shows an example of a trajectory with its measurements shown as black crosses, uncertainty as the blue area, and some predictions made by a GP, which is explained later.

GP has been studied and applied extensively [3]. There have been several works that applied GP for different types of trajectories such as in robotics [4, 5, 6, 7, 8], video surveillance [9, 10], batch trajectories processing [11], sensor trajectories in active sensing [12], travel-time prediction [13], and human motion [14, 15]. This work describes how to apply GP for trajectory interpolation and prediction, especially GPS-based trajectories.

2 Gaussian Process

A Gaussian Process (GP) is a generalization of the Gaussian probability distribution. While a probability distribution describes scalar or vector random variables, a stochastic process describes properties of functions, i.e., instead of describing the probability of generating a point as a Gaussian distribution, a GP describes the probability of generating a function. If one can loosely consider a function as a very long vector, where each entry in the vector indicates the function value $f(t)$ at a specific input t (which is a timestamp in our case), then a GP implies that any subset of entries in that long vector is distributed according to a multidimensional Gaussian. We build a separate GP for latitude and another one for longitude, assuming that their values change independently. This assumption is reasonable when objects can move in free space such as animals, drones, or human movements. In some applications with specific constraints

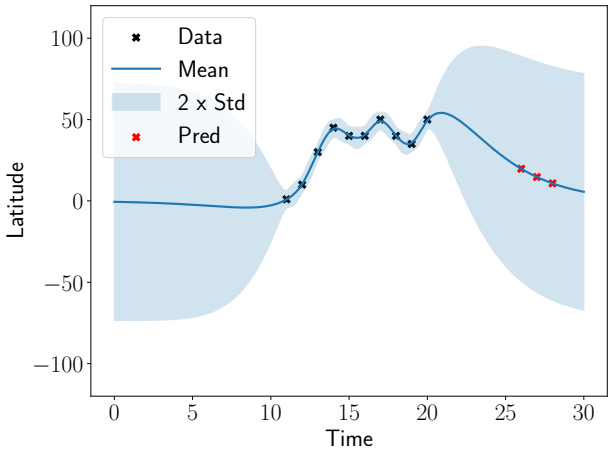


Figure 1: Illustration of a Gaussian process trained from data (black cross) making predictions (red cross).

such as road networks, without that network data and working only with location measurements, we can assume that the movement is in free space. After training and making predictions, latitude and longitude predictions are combined together to derive the final location prediction.

As an example, one wants to estimate locations of a user at a set of timestamps $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$. Thus, the input of the function $f(t)$ that a GP (for either latitude or longitude) describes is a timestamp $t \in \mathcal{T}$, and the output is the mean and variance of the Gaussian distribution that describes the distribution of $f(t)$. A GP implies that the set $\{f(t)|t \in \mathcal{T}\}$ is distributed according to a $|\mathcal{T}|$-dimensional Gaussian. When $\mathcal{T}$ includes only one timestamp t, the longitude (or latitude) of the estimated location is distributed according to a univariate Gaussian distribution. Figure 1 shows an example of a GP trained from data points (represented as black crosses) and making predictions (represented as red crosses). The location axis can either be longitude, latitude, or their transformed values such as converted to another spatial coordinate system.

A GP is specified by its mean function $m(t)$ and covariance function $k(t, t')$. A deterministic *mean function* $m(t)$ is used to specify the mean of the multidimensional Gaussian, defining the mean of the estimated locations. In the literature, GPs are often studied with a zero mean function, which means $m(t) = 0$ for all t, because a deterministic mean function can be incorporated after the kernel function is trained from training data.

The covariance matrix of the multidimensional Gaussian defines how points at different values of the independent variable (time in our case) correlate with each other. In general, timestamps that are closer would correlate more and those that are further apart would correlate less, which means the correlation decreases to zero as $|t - t'|$ becomes larger. A GP models such correlation using a scalar *covariance kernel/function* $k(t, t')$. A GP can be very flexible by using and combining different kernels, which will be discussed later. A kernel often has some parameters that need to be trained from data, such as pertaining to the uncertainty of the input data or how strong a correlation should be given the time difference $|t - t'|$.

When using a GP for trajectory interpolation and prediction, one would need to specify its mean function $m(t)$ and covariance function $k(t, t')$, train its parameters from data, then make predictions. We discuss each of these elements in the next section.

3 Gaussian Process Elements

In this section, we describe the elements needed to implement a GP for trajectories, including data preparation, mean function, covariance function (or kernel), training, and inference. Our implementation uses the GPFlow library [16], which is a package for building Gaussian process models in Python, but the concepts also apply to other libraries.

3.1 Data Preparation

We first describe how to prepare data from a trajectory $S = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{|S|}\}$ for GP training and inference. As mentioned, two separate GPs are created for longitude and latitude predictions. Input data for training each GP includes the sequence $\{(t_1, v_1), \ldots, (t_{|S|}, v_{|S|})\}$ where $t_i = \boldsymbol{x}_i.t$ and coordinate v_i can be either $\boldsymbol{x}_i.lon$ or $\boldsymbol{x}_i.lat$, and

the measurement uncertainty $\sigma_m = \boldsymbol{x}.\sigma$. From the training sequence, we create a feature vector $\boldsymbol{X} = [t_1, \ldots, t_{|S|}]^T$ and label vector $\boldsymbol{y} = [v_1, \ldots, v_{|S|}]^T$. Similarly, for interpolating or predicting locations of the user at timestamps in a timestamp set $\mathcal{T}$, a feature vector $\boldsymbol{X}_* = [t | t \in \mathcal{T}]^T$ is created.

In addition to $\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{X}_*$ and σ_m, one may also use a prior uncertainty σ_0, which represents the prior information of the uncertainty of locations of the users in terms of the standard deviation of a distribution. For example, σ_0 can be the standard deviation of a large Gaussian distribution covering an entire city area, which would indicate a high degree of prior uncertainty.

3.2 Mean Function

As mentioned, the mean function $m(t)$ is a deterministic function defining the mean of the estimated locations. A GP makes predictions based mainly on parameters learned from training data. When a prediction timestamp is far from training timestamps, the prediction tends to return to the mean function.

A GP is often assumed to have zero mean, i.e., $m(t) = 0, \forall t$. However, when there is a clear pattern, it can be beneficial to provide a more realistic mean to the model. For example, a trajectory may have a linear trend or we may believe future locations are likely to follow a linear extrapolation of two most recent measurements.

Using the GPFlow library, common mean functions can be specified using *gpflow.mean_functions*. For example, a constant mean function $m(t) = c$ with *gpflow.mean_functions.Constant*, a linear mean function $m(t) = At + b$ with *gpflow.mean_functions.Linear*, or a custom mean function can also be specified.

3.3 Kernel/Covariance Function

The kernel or covariance function $k(t, t')$ is the core element that helps a GP capture trends in data and gives a GP flexibility and efficiency. Different kernels capture different kinds of trends, and kernels can also be combined together for more complex models. In this section, we describe some common kernels and some common types of combinations. Examples of these kernels are shown in Figure 2, and examples of these combinations are shown in Figure 3. More detail about kernels can be found in [3] and [17].

Note that parameters of a kernel can be set to be trained from data or fixed. For example, if we know the measurement uncertainty σ_m, we can set it to be fixed; otherwise, it can be learned from the data. If using the GPFlow library, this can be set using the *set_trainable* method.

3.3.1 Common Kernel Types

Constant Kernel The constant kernel outputs constant predictions, i.e., $f(t) = c$ with $c \sim \mathcal{N}(0, \sigma^2)$. The formula is

$$k_{Constant}(t, t') = \sigma^2 \tag{1}$$

The constant kernel is often used to modify the mean of a GP or to scale the magnitude of the other factor (kernel) of a kernel. The GPFlow library provides this kernel by *gpflow.kernels.Constant* with parameter *variance*.

White Kernel The white kernel explains/produces the noise-component of a measurement. The formula is

$$k_{White}(t, t') = \sigma^2 \delta(t, t') \tag{2}$$

where $\delta(t, t') = 1$ when $t = t'$; otherwise, $\delta(t, t') = 0$. GPFlow provides this kernel with *gpflow.kernels.White* with parameter *variance*. Since we know that the measurement uncertainty is σ_m, we can set the variance value of this kernel to σ_m^2 (measurement noise) and set this parameter as non-trainable.

Linear Kernel The linear kernel outputs linear predictions, i.e., $f(t) = ct$ with $c \sim \mathcal{N}(0, \sigma^2)$. The formula is

$$k_{Linear}(t, t') = \sigma^2 t t' \tag{3}$$

The linear kernel is often used in combination with other kernels to capture linear trends in data. This kernel is a *non-stationary* kernel, which means it depends on the absolute locations of the two inputs. We will see later some *stationary* kernels, which only depend on the relative positions of its two inputs. The GPFlow library provides this kernel by *gpflow.kernels.Linear* with parameter *variance*.

Squared Exponential (SE)/Radial Basis Function (RBF) Kernel The SE/RBF kernel is often used as the default kernel for a GP. This is a stationary and smooth kernel with the formula

$$k_{SE}(t,t') = \sigma^2 \exp\Big(-\frac{(t-t')^2}{2l^2}\Big) \tag{4}$$

The length scale l indicates how much one data point affects another and is often trained from data. The variance σ^2 is trained from data but one can set this variance to σ_0^2 to provide prior information about the variance of predictions. GPFlow provides this kernel by *gpflow.kernels.SquaredExponential* with parameters *variance* and *length_scale*. Also, instead of providing scalar values, prior distributions can also be used as the prior for the length scale l and variance σ^2.

Rational Quadratic (RQ) Kernel Another common kernel that is often used as the default kernel for a GP is the rational quadratic (RQ) kernel with the formula

$$k_{RQ}(t,t') = \sigma^2\Big(1+\frac{(t-t')^2}{2\alpha l^2}\Big)^{-\alpha} \tag{5}$$

The RQ kernel is equivalent to adding together many SE kernels with different length scales l. It converges to the SE kernel when $\alpha \to \infty$. Parameter α controls the relative weighting of large-scale and small-scale variations. The GPFlow library provides this kernel by *gpflow.kernels.RationalQuadratic* with parameters *variance*, *length_scale*, and *alpha*.

Matérn Kernel The Matérn kernel is another common kernel. It is a stationary kernel and a generalization of the SQ kernel with an additional parameter ν controlling the smoothness. It converges to the SE kernel when $\nu \to \infty$. The formula for the Matérn kernel is rather complicated. However, there are three common values of ν, which are $\frac{1}{2}, \frac{3}{2}, \frac{5}{2}$, that produce three common kernels called *Matern12*, *Matern32*, and *Matern52*. Their formulas are

$$k_{Matern12}(t,t') = \sigma^2 \exp\Big(-\frac{1}{l}|t-t'|\Big) \tag{6}$$

$$k_{Matern32}(t,t') = \sigma^2\Big(1+\frac{\sqrt{3}}{l}|t-t'|\Big)\exp\Big(-\frac{\sqrt{3}}{l}|t-t'|\Big) \tag{7}$$

$$k_{Matern52}(t,t') = \sigma^2\Big(1+\frac{\sqrt{5}}{l}|t-t'|+\frac{5}{3l}(t-t')^2\Big)\exp\Big(-\frac{\sqrt{5}}{l}|t-t'|\Big) \tag{8}$$

Using the GPFlow library, these kernels are provided by *gpflow.kernels.Matern12*, *gpflow.kernels.Matern32* and *gpflow.kernels.Matern52* with parameters *variance* and *length_scale*.

3.3.2 Common Types of Kernel Combination

Kernels can be combined together to create more complex kernels, helping to capture more complex trends in the data. For example, we may want to capture both a wiggling movement with an SE kernel and measurement uncertainty with a white kernel. Two common types of combining kernels are summing and multiplying. Examples of these combinations are shown in Figure 3.

Summing Kernels Summing two kernels k_1 and k_2 means summing their output:

$$k_{Sum}(t,t') = k_1(t,t') + k_2(t,t') \tag{9}$$

Roughly speaking, summing kernels is similar to an OR operation, which means the output of k_{Sum} would be higher if the output of either k_1 or k_2 is higher. An example of summing kernels is to capture both a wiggling movement with a SE kernel and measurement uncertainty with a white kernel:

$$k(t,t') = k_{SE}(t,t') + k_{White}(t,t') = \sigma_{SE}^2 \exp\Big(-\frac{(t-t')^2}{2l^2}\Big) + \sigma_{White}^2\delta(t,t') \tag{10}$$

An illustration of this combination is shown in Figure 3.

Multiplying Kernels Multiplying two kernels k_1 and k_2 means multiplying their output:

$$k_{Mul}(t,t') = k_1(t,t') \times k_2(t,t') \tag{11}$$

Roughly speaking, multiplying kernels is similar to an AND operation, which means the output of k_{Sum} would be higher if the output of both k_1 and k_2 is higher. An example of multiplying kernels is multiplying two linear kernels to create a quadratic kernel or more to create higher-degree kernels. An illustration of this combination is shown in Figure 3.

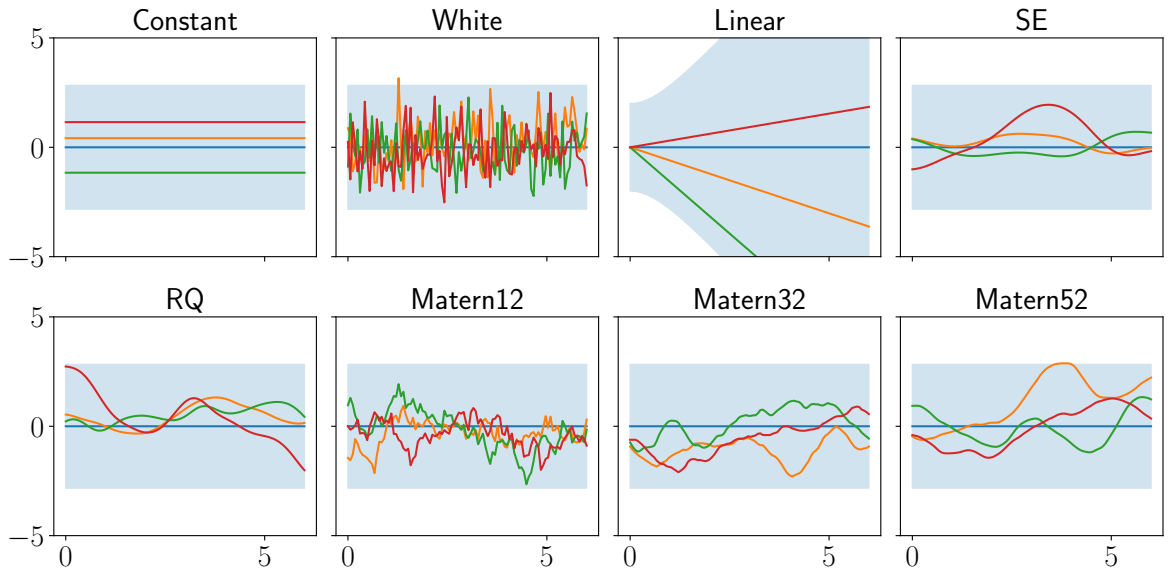


Figure 2: Examples of functions drawn from GPs with different kernels (in red, green, and orange). Each figure shows 3 functions drawing from the same GP with the blue line indicating the mean of the GP and the blue area indicating the variance of the GP. Predictions made from these kernels would have a similar shape.

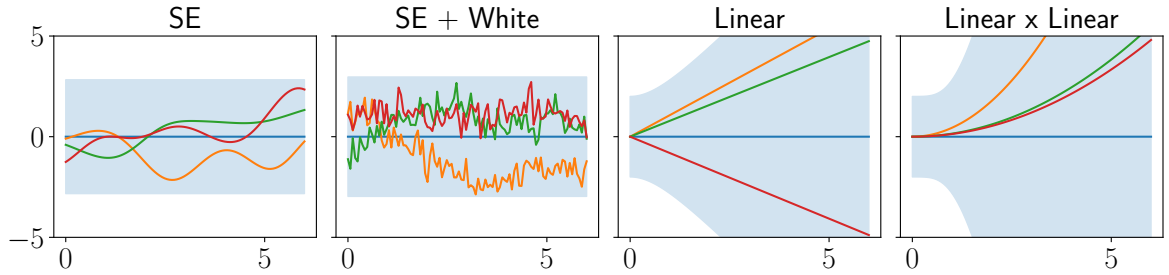


Figure 3: Examples of functions drawn from GPs with different types of kernel combination (in red, green, and orange). Each figure shows 3 functions drawing from the same GP with the blue line indicating the mean of the GP and the blue area indicating the variance of the GP. Predictions made from these kernels would have a similar shape.

3.4 Training and Inference

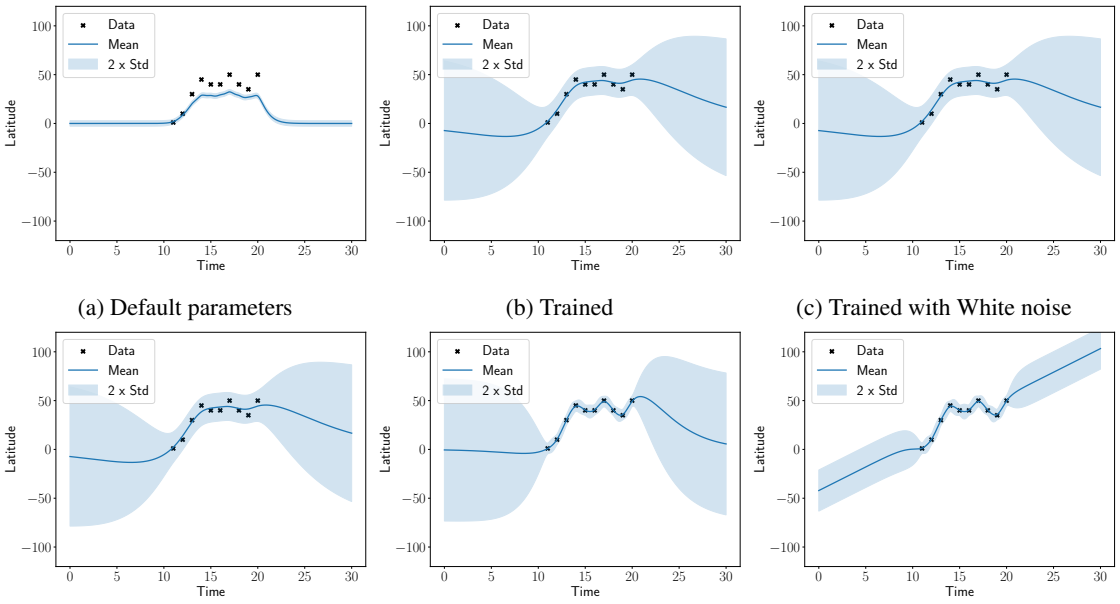
After constructing a mean function and a kernel and setting trainable parameters appropriately, training and inference processes are often straightforward using provided methods in GP libraries. Using GPFlow, for training, one can use existing optimizers in the package *gpflow.optimizers* such as the *Scipy* optimizer with input $\boldsymbol{X}, \boldsymbol{y}$. Once trained, we can use the trained model to make predictions with *predict_y*, which returns the mean and variance of predictions for new data points $\boldsymbol{X}_*$.

Note that the training process minimizes the negative log marginal likelihood. Since likelihood specifies the probability density of the observations/measurements given the parameters, for trajectory data, we can use Gaussian likelihood, which is the default in the GPFlow library. The likelihood variance indicates the uncertainty learned from data. For trajectory data, uncertainty in the movement of measurements can be captured by the model with likelihood variance, and the uncertainty of each measurement can be captured by a white kernel.

4 Gaussian Process Example

In this section, we present an example of using GP for trajectories. In this example, we use given (training) timestamps $\boldsymbol{X} = [11, 12, \ldots, 20]^T$, given locations $\boldsymbol{y} = [1, 10, 30, 45, 40, 40, 50, 40, 35, 50]^T$, and seek computed values at

timestamps $\boldsymbol{X}_* = [21, 22, \ldots, 24]^T$ (illustrated in Figure 1). For reproducibility, we use the GPFlow library and set the TensorFlow random seed to 1. GPFlow also provides a utility method *print_summary* that prints values of the parameters of a model.



(a) Default parameters (b) Trained (c) Trained with White noise

(d) Same as (c), but with fixed White noise (e) Same as (d), but with small likelihood variance (f) Same as (e), but with linear mean function

Figure 4: Example of GPs trained from the same training data (as black cross) with different configuration of kernel and mean function.

The first model m_0 has a zero mean function and a Matern32 kernel with default parameters, which is *lengthscales* $l = 1$ and *variance* $\sigma^2 = 1$. Figure 4a shows the predictions of this model before we trained its parameters with $\boldsymbol{X}$. We can see that the predictions do not appropriately capture measurements and give a quick return to 0.

After we train the model with the data to get a trained model m_1, we have $l = 7.35$, $\sigma^2 = 1346.36$, and likelihood variance (which indicates model variance) $\sigma^2_{likelihood} = 36.4$. Figure 4b shows the predictions of this model after we trained with $\boldsymbol{X}$. It is clear that m_1 can capture the data and trend in the data significantly better than m_0.

Next, we consider model m_2 as a summed combination of a Matern32 kernel and a white kernel, all with default parameters indicating default priors (e.g., prior for measurement uncertainty captured by the white kernel is 1). This also means all parameters can be trained from data. After we train the model with the data, we have $l = 7.35$, $\sigma^2_{Matern32} = 1346.36$, $\sigma^2_{White} = 18.23$, and $\sigma^2_{likelihood} = 18.23$. Figure 4c shows that the model m_2 makes similar predictions compared to model m_1. However, the white kernel captured some of the measurement noise.

Next, we consider model m_3, which similar to m_2, but we fix the variance of the white kernel to *variance* $\sigma^2_{White} = 4$, which means we assume that each measurement has uncertainty $\sigma^2 = 4$. Fixing the variance means the training process will not change this parameter. After we train the model with the data, we have $l = 7.35$, $\sigma^2_{Matern32} = 1346.36$, $\sigma^2_{White} = 4$, and $\sigma^2_{likelihood} = 32.464$. Since we fixed the variance of the white kernel, the likelihood variance was adjusted to capture more of the uncertainty. Figure 4d shows that the model m_3 make similar predictions compared to model m_2.

Next, we consider model m_4 similar to m_3 but in addition to fixing the variance of the white kernel to 4, we also set the likelihood variance to a very small initial value $\sigma^2_{likelihood} = 0.0001$, which indicates that we are confident of our knowledge of measurement uncertainty. After we train the model with the data, we have $l = 4.11$, $\sigma^2_{Matern32} = 1328.03$, $\sigma^2_{White} = 4$, and $\sigma^2_{likelihood} = 0.00015$. Figure 4e shows that, compared to model m_3, the model m_4 make predictions closer to the training measurements. Figure 1 shows predictions made by m_4 for $\boldsymbol{X}_*$.

In the final example, we consider model m_5 similar to m_4 but using a linear mean function starting with a guess $y = x+1$. After we train the model with the data, we have $l = 1.37$, $\sigma^2_{Matern32} = 107.24$, $\sigma^2_{White} = 4$, $\sigma^2_{likelihood} = 0.0001$, and a mean function $y = 4.84x - 42.14$. The training process learned the linear line going through the measurements

and used that as the mean function. The model then made predictions closer to that mean function and with smaller variance, as shown in Figure 4f.

5 Discussion

Gaussian process is a powerful and flexible technique that is capable of capturing trends and uncertainty of location measurements. A GP can use a simple or complex mean function, or even use output of other models, such as a deep neural network model [18], as its mean function. A GP can use standard kernels, combined kernels, or other complex kernels. These capabilities help GP incorporate knowledge from other domains for better interpolation and prediction.

There are some issues that need to be taken into account when one considers using GPs for trajectories. One issue is that the length scale of a kernel, which roughly indicates how far in time a measurement affects other predictions, tends to be determined by the non-smooth region in the data. For trajectories, these regions are often where we have dense measurements with many direction changes. These regions can make the length scale become too small, resulting in a very sharp change in variance. It can be beneficial to create different GPs when the trajectory sampling rate changes significantly. Another issue can be the assumption of independence between longitude and latitude changes. Other issues choosing appropriate kernels, or computational complexity. However, in general, GP is a useful technique for trajectory interpolation and prediction.

References

[1] Yu Zheng. Trajectory data mining: an overview. *TIST*, 6(3):1–41, 2015.

[2] Frank Van Diggelen. Gnss accuracy: Lies, damn lies, and statistics. *GPS world*, 18(1):26–33, 2007.

[3] Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006.

[4] Gregory Cox, George Kachergis, and Richard Shiffrin. Gaussian process regression for trajectory analysis. In *Proceedings of the annual meeting of the cognitive science society*, volume 34, 2012.

[5] Tim D Barfoot, Chi Hay Tong, and Simo Särkkä. Batch continuous-time trajectory estimation as exactly sparse gaussian process regression. In *Robotics: Science and Systems*, volume 10. Citeseer, 2014.

[6] Lukas Hewing, Elena Arcari, Lukas P Fröhlich, and Melanie N Zeilinger. On simulation and trajectory prediction with gaussian process dynamics. In *Learning for Dynamics and Control*, pages 424–434. PMLR, 2020.

[7] Gang Cao, Edmund M-K Lai, and Fakhrul Alam. Gaussian process model predictive control of an unmanned quadrotor. *Journal of Intelligent & Robotic Systems*, 88(1):147–162, 2017.

[8] Elnaz Jahani Heravi and Sohrab Khanmohammadi. Long term trajectory prediction of moving objects using gaussian process. In *2011 First International Conference on Robot, Vision and Signal Processing*, pages 228–232. IEEE, 2011.

[9] Kihwan Kim, Dongryeol Lee, and Irfan Essa. Gaussian process regression flow for analysis of motion trajectories. In *2011 International Conference on Computer Vision*, pages 1164–1171. IEEE, 2011.

[10] David Ellis, Eric Sommerlade, and Ian Reid. Modelling pedestrian trajectory patterns with gaussian processes. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1229–1234. IEEE, 2009.

[11] Mattias Tiger and Fredrik Heintz. Online sparse gaussian process regression for trajectory modeling. In *2015 18th International Conference on Information Fusion (Fusion)*, pages 782–791. IEEE, 2015.

[12] Jerome Le Ny and George J Pappas. On trajectory optimization for active sensing in gaussian process models. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6286–6292. IEEE, 2009.

[13] Tsuyoshi Idé and Sei Kato. Travel-time prediction using gaussian process regression: A trajectory-based approach. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 1185–1196. SIAM, 2009.

[14] Jisoo Hong, Changmook Chun, Seung-Jong Kim, and Frank C Park. Gaussian process trajectory learning and synthesis of individualized gait motions. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(6):1236–1245, 2019.

[15] Jack M Wang, David J Fleet, and Aaron Hertzmann. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298, 2007.

[16] Mark van der Wilk, Vincent Dutordoir, ST John, Artem Artemev, Vincent Adam, and James Hensman. A framework for interdomain and multioutput Gaussian processes. *arXiv:2003.01115*, 2020.

[17] David Duvenaud. The kernel cookbook: Advice on covariance functions. https://www.cs.toronto.edu/ duvenaud/cookbook/. Last accessed: September 22, 2021.

[18] Vincent Fortuin and Gunnar Rätsch. Deep mean functions for meta-learning in gaussian processes. *arXiv preprint arXiv:1901.08098*, 2019.