Description of Non-Stationary Gaussian Process Implementation

Brennon Shanks

August 28, 2025

1 Overview

This document provides a comprehensive description of the Jupyter notebook `nsGP_autocorr.ipynb` and the Python file `original.py`, which implement a non-stationary Gaussian Process (GP) model for analyzing autocorrelation data (`J_corr_z`). The notebook includes data loading, normalization, GP modeling, posterior sampling, and Fast Fourier Transform (FFT) analysis. The `original.py` file defines the `NonStationaryGP` class, which uses a Gibbs kernel and an exponential mean function. We present the mathematical foundations with LaTeX-formatted equations for clarity, describe the code structure, and discuss modifications, including incorporating priors and changing mean and kernel functions.

2 `original.py`: Non-Stationary GP Class

2.1 Global Settings and Utilities

The file sets PyTorch's default data type to double precision (`torch.float64`) to ensure numerical stability for GP computations, which involve matrix inversions and Cholesky decompositions.

The `custom_cdist` function computes squared Euclidean distances between two sets of points:

$$\text{custom\_cdist}(x_1, x_2)_{ij} = \|x_{1i} - x_{2j}\|^2 \tag{1}$$

For inputs $x_1 \in \mathbb{R}^{N\times 1}$, $x_2 \in \mathbb{R}^{M\times 1}$, it returns a matrix $\in \mathbb{R}^{N\times M}$. The implementation uses:

$$\|x_{1i} - x_{2j}\|^2 = x_{1i}^2 + x_{2j}^2 - 2x_{1i} \cdot x_{2j} \tag{2}$$

This is computed efficiently via matrix multiplication, avoiding explicit loops.

2.2 `NonStationaryGP` Class

The `NonStationaryGP` class, a subclass of `torch.nn.Module`, defines a GP with a non-stationary Gibbs kernel and an exponential mean function. The GP models data:

$$y \sim \mathcal{N}(m(t), K(t, t') + \sigma_n^2 I) \tag{3}$$

where $m(t)$ is the mean function, $K(t, t')$ is the covariance kernel, and σ_n^2 is the noise variance.

2.2.1 Choosing a GP Prior

GP prior mean selection defines the types of functions that could possibly explain the data before looking at any observations. To specify a GP prior, you need to choose a mean and covariance function. To enforce a tail behavior, such as sending the autocorrelation to 0 at large times, one can force the kernel to approach 0 and the mean to approach 0 for large t. Below, I show the function choices that I selected for this code, although you can easily alter these and compare the results with different prior choices after the fact. All you need to do to modify these is change the functions in the `NonStationaryGP` class alongside the initialized hyperparameters and rerun the code.

2.2.2 Mean Function

The mean function models the expected autocorrelation:

$$m(t) = A \exp\left(-\frac{t}{\tau}\right) \tag{4}$$

This exponential decay describes the simple ideal gas solution to the Fokker-Planck equation. This mean function has two hyperparameters:

- `A_raw`, `tau_raw`.

2.2.3 Gibbs Kernel

The covariance kernel I selected is known as the *Gibbs kernel*, which for a fixed lengthscale ℓ is given by:

$$K(t_i, t_j) = \sigma(t_i)\sigma(t_j) \exp\left(-\frac{(t_i - t_j)^2}{2\ell^2}\right) \tag{5}$$

The prefactor $\sigma(t_i)\sigma(t_j)$ introduces non-stationarity, and the exponential term resembles a radial basis function or squared-exponential kernel. The `f_kernel` method in the `NonStationaryGP` class uses `custom_cdist` for $(t_i - t_j)^2$.

2.2.4 Width Function

The Gibbs kernel requires specification of a time-varying width function, which I selected to be an inverse sigmoid:

$$\sigma(t) = \frac{\text{max\_val}}{1 + \exp(\text{slope} \cdot (t - \text{loc}))} \tag{6}$$

This sigmoid-like function allows the kernel's amplitude to vary smoothly to zero, transitioning around $t = \text{loc}$. The hyperparamters for the width function are therefore

- `ell_raw`, `max_val_raw`, `slope_raw`, `loc_raw`, `sigma_n_raw`.

These control the kernel's length scale (ℓ), time-varying amplitude (max_val, slope, loc), noise standard deviation (σ_n), and mean function's amplitude (A) and decay time (τ).
Note: The parameters are initiated as exponentials of the raw parameters. This is done to ensure that every sample of these hyperparameters is positive, ensuring that the optimizer explores the better behaved log space.

2.2.5 Negative Log Marginal Likelihood

The negative log marginal likelihood (NLML) for training is:

$$\text{NLML} = \frac{1}{2}(y - m)^\top (K + \sigma_n^2 I)^{-1}(y - m) + \frac{1}{2}\log|K + \sigma_n^2 I| + \frac{n}{2}\log(2\pi) \tag{7}$$

where: - $y \in \mathbb{R}^n$: Observations. - $m \in \mathbb{R}^n$: Mean vector $m(t_i)$. - $K \in \mathbb{R}^{n \times n}$: Kernel matrix. - $\sigma_n^2 I$: Noise covariance. This equation is essentially a loss function for the hyperparameters, and its minimum value is what the stochastic gradient descent optimizer is looking for. The hyperparameters at the NLML likelihood represent the most likely hyperparameters to explain the provided observations of the autocorrelations provided from your simulations, and are subsequently used in the GP posterior evaluation.
Using Cholesky decomposition ($K + \sigma_n^2 I = LL^\top$):

$$(y - m)^\top (K + \sigma_n^2 I)^{-1}(y - m) = (y - m)^\top (LL^\top)^{-1}(y - m) \tag{8}$$

$$\log|K + \sigma_n^2 I| = 2\sum_i \log L_{ii} \tag{9}$$

Finally, the `NEG_LMLH` method symmetrizes K for numerical stability.

2.2.6 Posterior Prediction

The `predict` method computes the posterior for test points X_{test}:

$$\mu_{\text{post}} = m(X_{\text{test}}) + K(X_{\text{test}}, X_{\text{train}})(K(X_{\text{train}}, X_{\text{train}}) + \sigma_n^2 I)^{-1}(y - m(X_{\text{train}})) \tag{10}$$

$$\text{cov}_{\text{post}} = K(X_{\text{test}}, X_{\text{test}}) - K(X_{\text{test}}, X_{\text{train}})(K(X_{\text{train}}, X_{\text{train}}) + \sigma_n^2 I)^{-1} K(X_{\text{train}}, X_{\text{test}}) \tag{11}$$

A jitter (10^{-8}) is added to $K(X_{\text{test}}, X_{\text{test}})$ for stability.

2.3 Modifications

The `NonStationaryGP` class can be modified as follows:

2. **Incorporating Priors**: If you have prior information on hyperparameter values, then you can put priors on those parameters and add them to the NLML calculation for hyperparameter training. Add priors to `NEG_LMLH` by subtracting log prior terms. For example, for a Gamma prior on $\tau \sim \text{Gamma}(\alpha, \beta)$:

$$p(\tau) = \frac{\beta^\alpha}{\Gamma(\alpha)} \tau^{\alpha-1} e^{-\beta\tau} \tag{12}$$

$$-\log p(\tau) = -(\alpha - 1)\log\tau + \beta\tau + \log\Gamma(\alpha) + \alpha\log\beta \tag{13}$$

Modify `NEG_LMLH`:

```
def NEG_LMLH(self, X, Y):
    nll = ...  # existing NLML
    log_prior_tau = (2.0 - 1) * torch.log(self.tau) - 200.0
    * self.tau - torch.lgamma(2.0) - torch.log(1.0 / 200.0)
    return nll - log_prior_tau

```

Similar priors can be included for other parameters (e.g., Normal for ℓ, Gamma for σ_n).

3 Conclusion

The notebook and `original.py` implement a non-stationary GP for autocorrelation analysis. Modifications to the mean, kernel, and priors allow customization for different data characteristics or prior knowledge.