Vignesh Ravindranath
vgr325

Learning Hyperparameters for 3D x-y-z Position Data using Gaussian Process

1 Overview

The purpose of this assignment is to fine-tune Gaussian Processes hyperparameters – scale, length scale, and noise – on 3D x-y-z positions using both a global window and sliding window technique. The data provided contains 3D coordinates over time of 50 sensors attached all over a subject while they are tracing a 3D curve. The Gaussian Process regression is a non-parametric Bayesian approach is a useful model when trying to understand unknown functions, such as the 3D curve tracing data, where trajectories of all 50 sensors are difficult to know beforehand.

Before delving into the algorithm behind GPs, we can first look at a Gaussian distribution f:

$$p(f) = (2\pi)^{-\frac{n}{2}} |K|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(f-m)^T K^{-1}(f-m)\right), \text{or}$$

$$p(f) \sim N(m, K),$$

where, m is the mean vector and K is the positive semidefinite covariance matrix. The function f is considered a GP if $f(t) = [f(t_1), \dots, f(t_n)]$ has a multivariate normal distribution for all $t = [t_1, \dots, t_n]$, or

$$p(f|t) \sim N\big(f|m(t), K(t,t)\big).$$

In other words, any point t has a unique Gaussian distribution to describe its mean and variance. The covariance function, or the Gaussian kernel (RBF kernel), is described as

$$K(t_i, t_j) = \sigma_f^2 \exp\left(-\frac{1}{2\sigma_l^2}(t_i - t_j)^2\right),$$

where, σ_f and σ_l are hyperparameters for the GP. The kernel function is a measure for how similar any two points t_i and t_j are. If noise $n \sim GP(0, \sigma_n^2 I)$ were to be added to the prior distribution $f \sim GP(m_f, K_{ff})$, then, by definition, the resulting joint distribution of observed values f and $y = f + n$ is

$$p(y(t), f(t)) = N\left(\begin{bmatrix} f \\ y \end{bmatrix} \middle| \begin{bmatrix} m_f \\ m_y \end{bmatrix}, \begin{bmatrix} K_{ff} & K_{fy} \\ {K_{yf}}^T & K_{yy} \end{bmatrix}\right).$$

Where, σ_n is the third hyperparameter for the GP, $K_{ff} = K(t,t) + \sigma_n^2 I$, $K_{fy} = K(t, t_s)$, and $K_{yy} = K(t_s, t_s)$. If the observed data y and predicted data y_s are jointly distributed, then

$$p(y_s|y) = N\big(K_{y_s y} K_{yy}^{-1}(y - m_y) + m_{y_s}, \qquad K_{y_s y_s} - K_{y_s y} K_{y_s y_s}^{-1} K_{y_s y}^T\big).$$

2 Algorithm and Methods

GPs have three hyperparameters – σ_f, σ_l, and σ_n – that can be optimized by minimizing the log-marginal likelihood of the prior. The log likelihood of a GP distribution $p(t) \sim N(0, K + \sigma_n^2 I)$ is

$$\log p(t) = -\frac{1}{2} t^T (K + \sigma_n^2 I)^{-1} t - \frac{1}{2} \log(K + \sigma_n^2 I) - \frac{\mathrm{n}}{2} \log 2\pi.$$

Then, to minimize the log-likelihood, we need to find the derivative of the equation above:

$$\frac{\partial \log(K + \sigma_n^2 I)}{\partial \boldsymbol{\theta}} = trace\left((K + \sigma_n^2 I)^{-1} \frac{\partial \log(K + \sigma_n^2 I)}{\partial \boldsymbol{\theta}} \right),$$

where $\boldsymbol{\theta}$ represents the hyperparameters.

2.1 Overview, Data, and Libraries

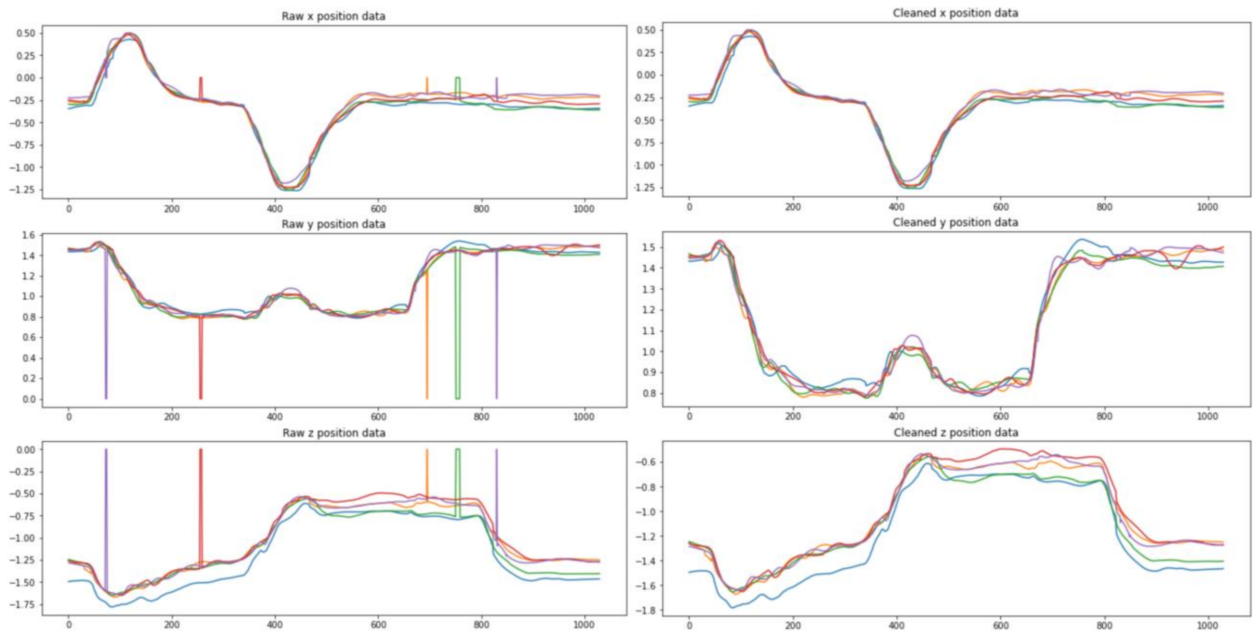
For this assignment, I used data from **sensor 15** (back of right hand), **block 1** (trial 1) of subject **AG**. I implemented a GP process using NumPy and Sklearn with slightly different methods (described below). For the NumPy implementation, the kernel and posterior distributions were written manually (no additional pre-built libraries). However, I was not able to figure out how to optimize the hyperparameters manually, so I implemented the same algorithm using the pre-build GP libraries found in Sklearn. The NumPy implementation will be used for illustrative purposes.

All the libraries I used are listed below:

a)	numpy	– linalg.inv
b)	matplotlib	– pyplot, mlp_toolkits.mplot3d.Axes3D
c)	sklearn.gaussian_process	– GaussianProcessRegressor, kernels.ConstantKernel, kernels.RBF

2.2 Clean Data

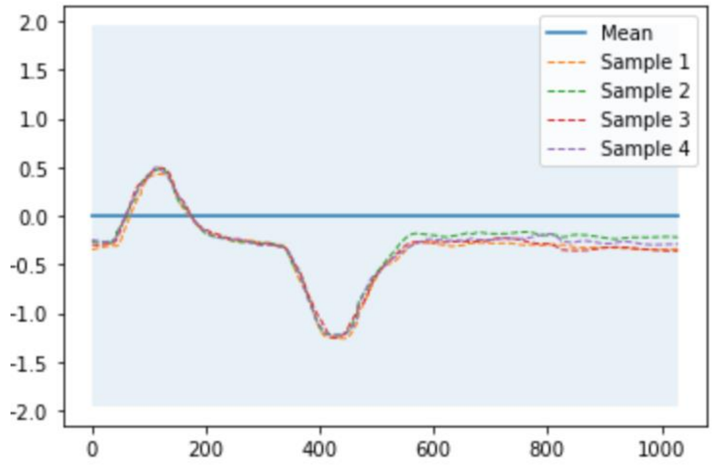
The data provided contains an additional variable, c, which indicates whether the input signal was good ($c > 0$) or not ($c < 0$). To clean the data, missing points will be predicted using previous data points.



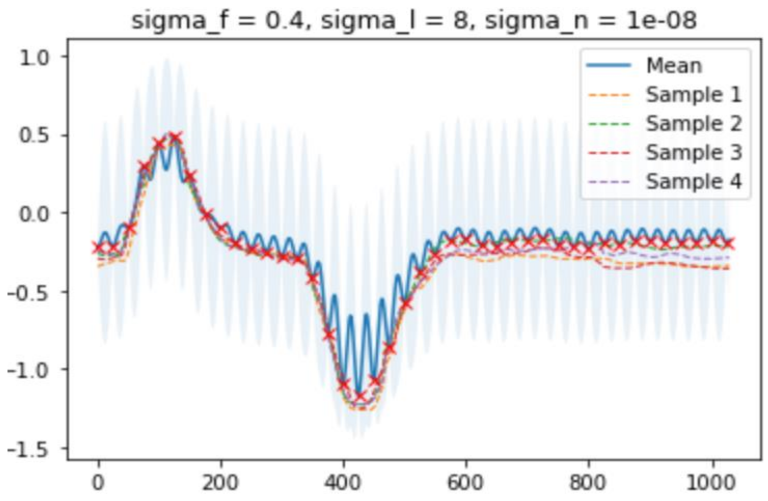
2.3 NumPy Implementation

The NumPy implementation did not require any additional libraries or pre-build learning algorithms. Rather, the kernel and posterior distributions were written manually (see Jupyter Notebook). The following steps describe the method I took to train the GP:

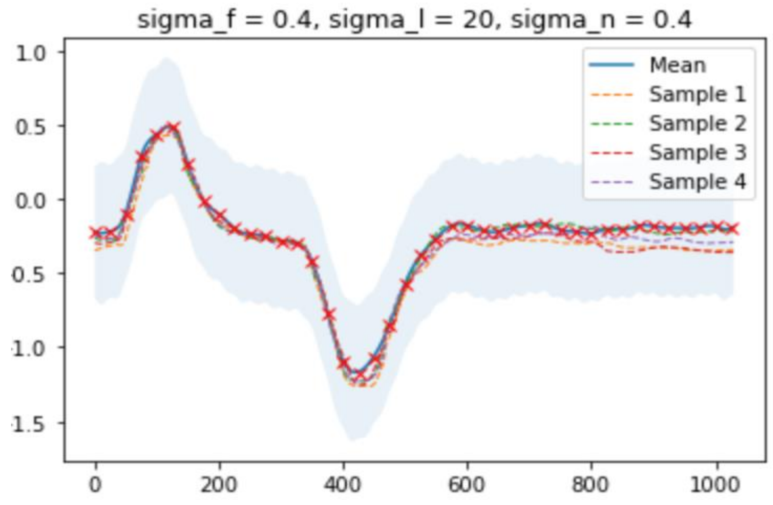
1. Define the range (t), zero mean vector (mu), and covariance matrix ($kernel(t, t)$) for the prior distribution, and plot against four sample x-trajectories (blocks 1-4 for subject AG):



2. Train GP on noise free data: obtain training points from the fifth and last sample (block 5) at intervals of 25 (arbitrarily chosen) and find the posterior distribution given these training points.



3. Train GP on noisy data: obtain training points from the last sample (block 5) at intervals of 25 and find the posterior distribution given these training points with noise=0.4 (for illustrative purposes).



2.4 Sklearn Implementation – Optimize Hyperparameters

To optimize the hyperparameters, I will use the Sklearn libraries listed above. I will use this implementation to optimize hyperparameters for both the global window and sliding window techniques. The following steps describe the method I took to train the GP using a **global window**:

1. Obtain training **x-y-z** trajectories from **block 1** of subject AG. The training data points will be taken from the actual trajectory at intervals of 25 (for illustrative purposes).
2. Using the training data, fit the GP using the $.\mathrm{fit}()$ method with initial hyperparameters of $\mathrm{sigma\_f} = \mathbf{1.0}$, $\mathrm{sigma\_l} = \mathbf{1.0}$, and $\mathrm{sigma\_n} = \mathbf{0.5}$. The hyperparameters are inputs to the Sklearn kernel functions ($\mathrm{ConstantKernel}$ and RBF).
3. Predict the x-y-z trajectories using the $.\mathrm{predict}()$ method on the entire window.

4. Use the $.\mathrm{get\_params}()$ methods to find the optimized hyperparameters.
5. Sample from the trained GP to obtain a predicted trace of the 3D trajectory.

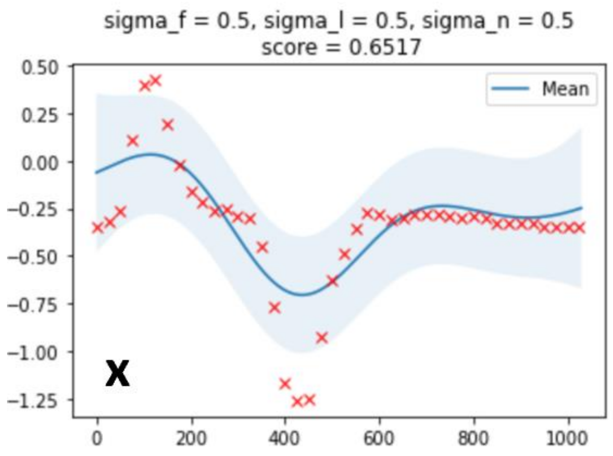
The following steps describe the method I took to train the GP using a **global window**:

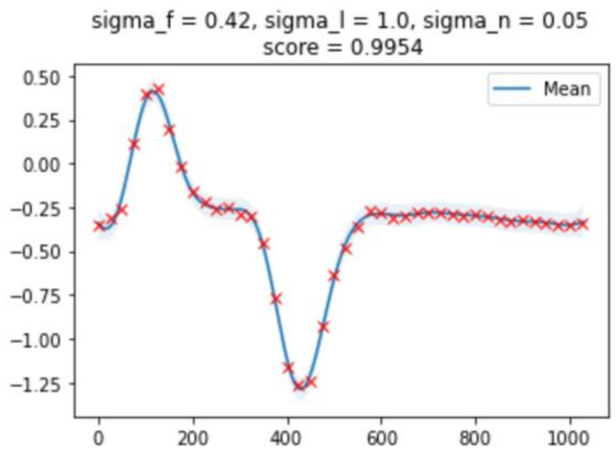
1. Define a window size ($\mathrm{win\_size} = \mathbf{102}$) and the increment length ($\mathrm{delta} = \mathbf{51}$).
2. Use points from each window to fit a GP using the $.\mathrm{fit}()$ method and with initial hyperparameters of $\mathrm{sigma\_f} = \mathbf{1.0}$, $\mathrm{sigma\_l} = \mathbf{1.0}$, and $\mathrm{sigma\_n} = \mathbf{0.5}$.
3. Predict the x-y-z trajectories for the window and save the output mu_s and cov_s values.
4. Use the $.\mathrm{get\_params}()$ method to find the optimized hyperparameters for the window and save the values.
5. Once steps 2-4 have been completed on all windows, average the mean and covariances over the overlapping window areas.
6. Plot the trained GP.

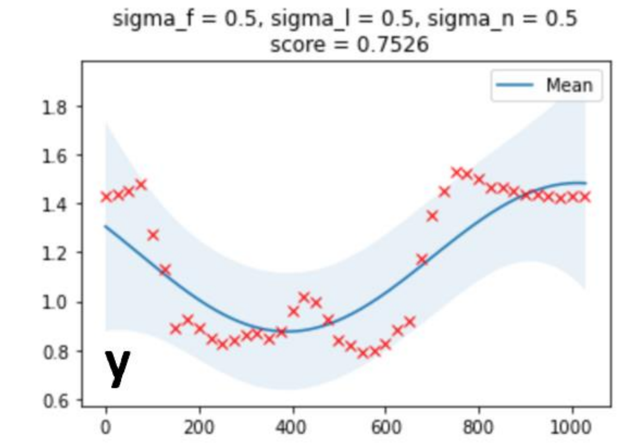
3 Results

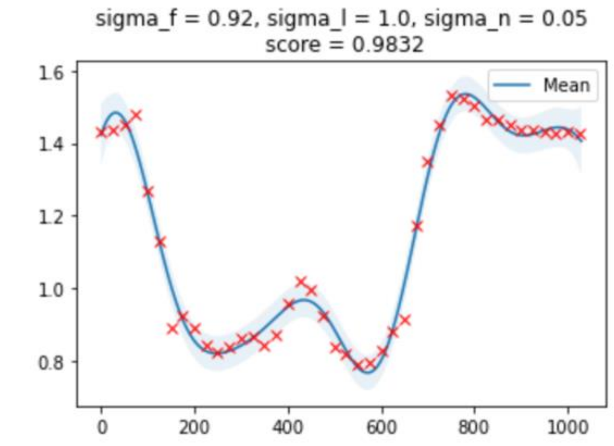
3.1 Global Window

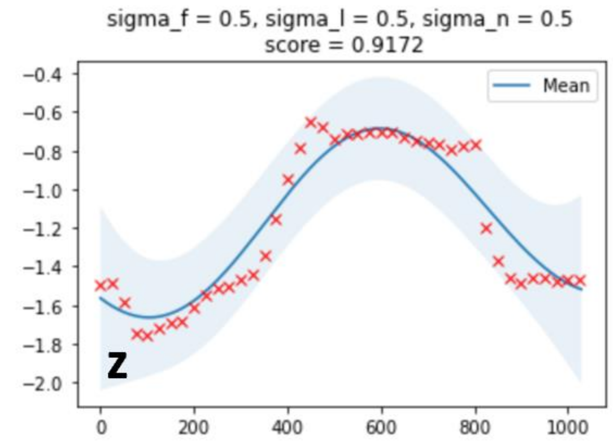
For the global window technique, we can see the optimal hyperparameters in the plots below (right) compared with sub-optimal hyperparameters (left) for comparison.

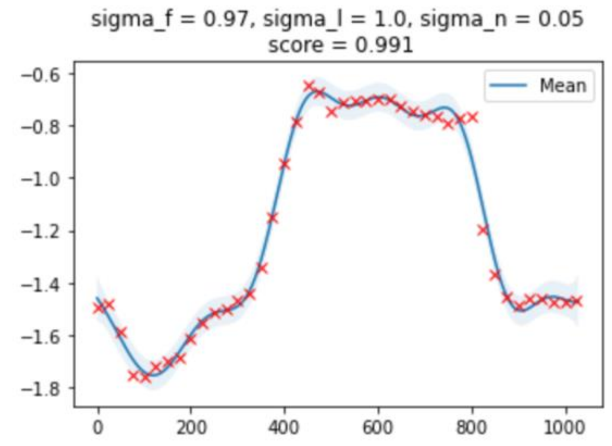




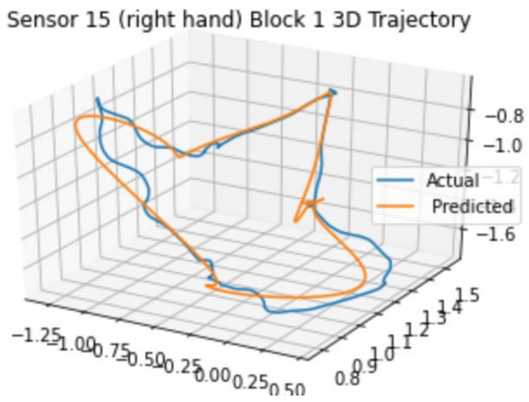








After sampling a x-y-z trajectories from the trained GPs, I obtained a predicted trace of the 3D trajectory.



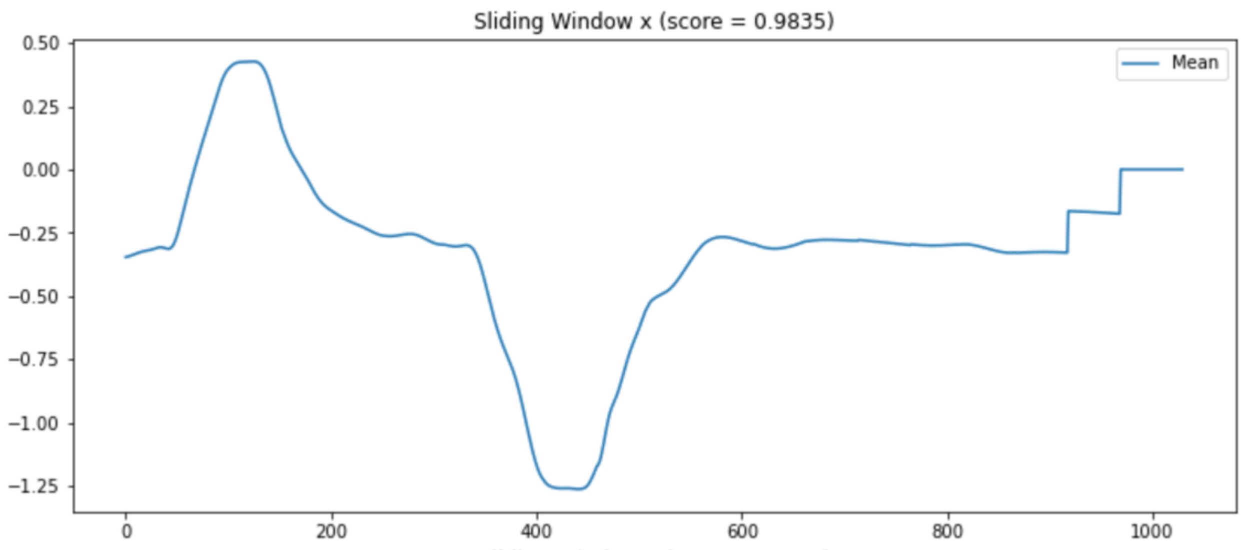
3.2 Sliding Window

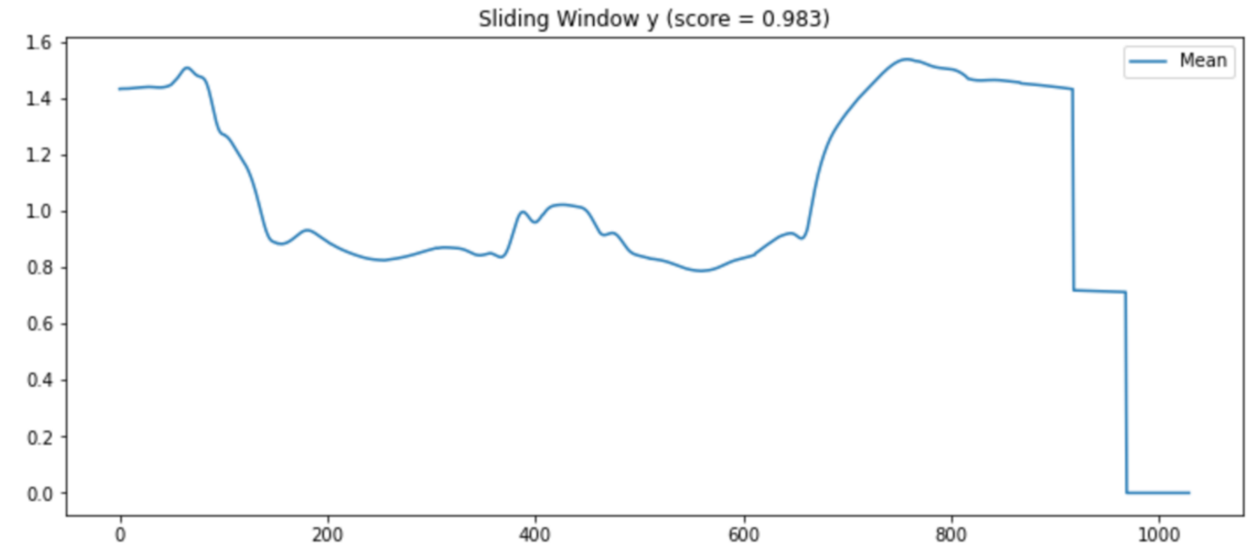
Next, for the sliding window method, a window size of 102 was used with an increment length of 51. In total, 18 windows were trained across the 1030 data points. The table below indicates the average hyperparameter values for the 18 windows for the x-y-z trajectories.

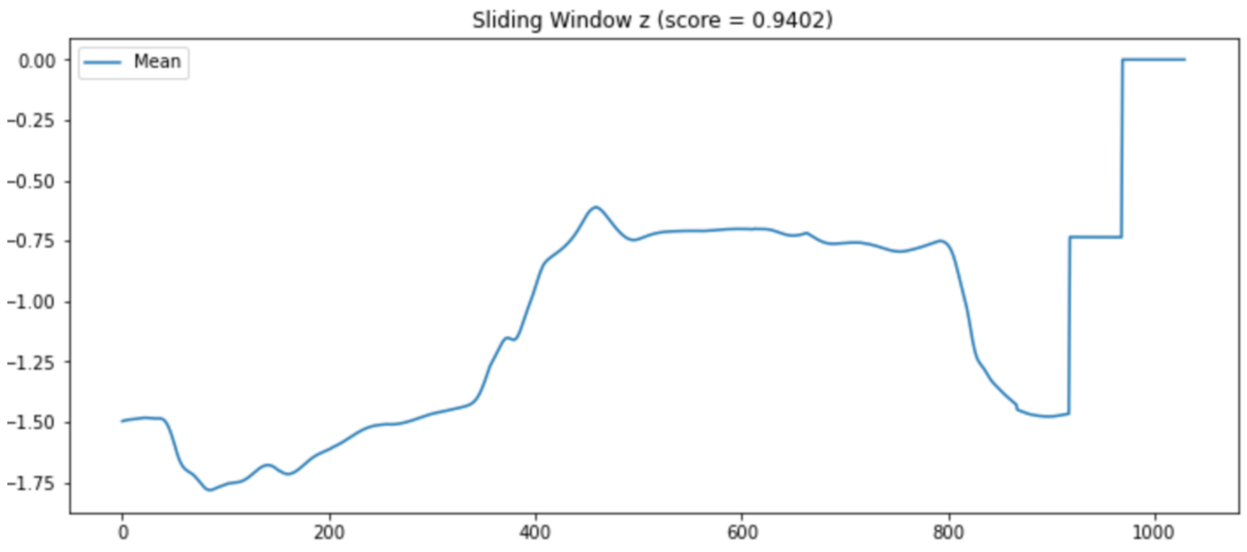
	σ_f	σ_l	σ_n
AG Block 1 x	0.34	65.56	0.05
AG Block 1 y	0.81	68.27	0.05
AG Block 1 z	0.82	67.29	0.05

As the averages indicate, the $\sigma_f$values seemed to average at 0.34 for the x-trajectory and at 0.80 for the y-z-trajectories. The $\sigma_l$values seemed to average at 66 and the $\sigma_n$values seemed to average at 0.05.

The plots below indicate the fitted parameter values of local windows at each frame. Because of the uneven window size, the mean values towards the end of the curve are not accurate.


Sliding Window x (score = 0.9835)
Mean
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
0
200
400
600
800
1000


Sliding Window y (score = 0.983)
Mean
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000


Sliding Window z (score = 0.9402)
Mean
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
0
200
400
600
800
1000

4 Summary

Overall, GP regression was an effective method to predict the 3D trajectory of human kinematics. From the NumPy implementation, we found how the hyperparameters affect the GP results: σ_f controls the vertical variation and σ_l controls the smoothness. From the Sklearn implementation I were able to train a GP on a global window with an average score of 0.9899. Using the Sklearn library, I was able to sample from the trained GP to create a predicted trace of the same 3D trajectory. Moreover, I was able to train a GP using a sliding window technique. Although the variance of the GP is very low, the average score was found to be 0.9689, which is slightly lower than the global window technique. The lower score could be attributed to the uneven window size. With the global window method, the optimal hyperparameters for the x-y-z trajectories were found to be 0.40-0.90, 1.0, 0.05 respectively. With the sliding window method, the average optimal hyperparameters for the x-y-z trajectories were found to be 0.34-0.80, 66.0, 0.05 respectively. In conclusion, the global window might serve as a simpler and more effective method to train GP on human kinematics.