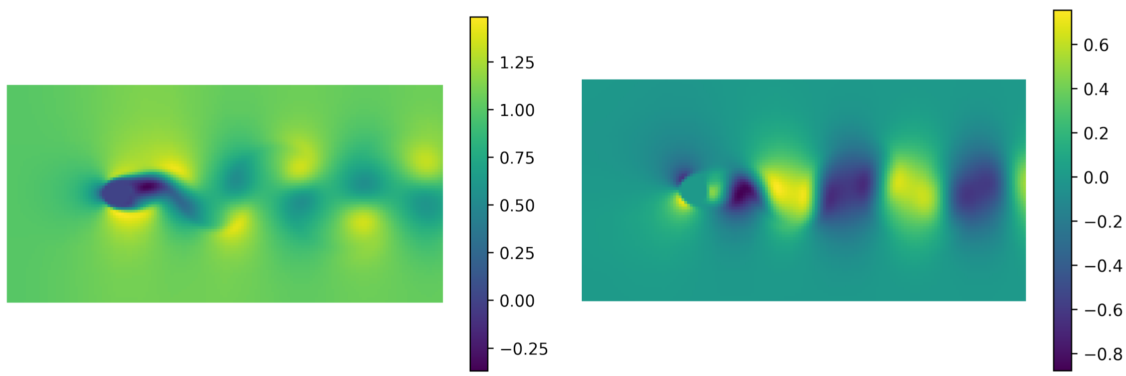
ROB313: Introduction to Learning from Data
University of Toronto Institute for Aerospace Studies

Assignment 5 (12.5 pts)

Due April 14, 2023, 23:59 EST

Starter code for this assignment is provided in pca_gp_rom.py *on Quercus.*

In this assignment, you will be using the cylinder2d dataset [1], which is a time-series dataset of fluid flow around a cylinder. This dataset consists of a 2D velocity field over a uniform 160×80 grid over 1501 time steps. The velocity field at the final time step is visualized below:



(a) Horizontal velocity component

(b) Vertical velocity component

Figure 1: Cylinder flow: State at final time step

You'll be doing *reduced-order modelling*, which involves encoding high-dimensional states as lower-dimensional representations, and solving for the dynamics in those terms. The goal is to reduce the complexity of simulating high-dimensional systems, and since the cylinder flow dataset consists of 2 flow variables on a 160×80 grid, it is effectively a 25600-dimensional system! To find the low-dimensional representations, you'll be using PCA to encode each 25600-dim state into a 4-dim representation. To then simulate the dynamics, you will form a Gaussian process model in terms of the encoded state. Because this is a time-series model, the feature $\mathbf{x}$ is the time and the target $\mathbf{y}$ is the flow state.

[1] Tobias Günther, Markus Gross, and Holger Theisel. "Generic Objective Vortices for Flow Visualization". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 36.4 (2017), 141:1–141:11.

Q1) 1.5pts Consider the function load_cyl2d_dataset provided in the starter code. This function reads in the cylinder2d dataset and partitions it into training, validation, and test sets. In the starter code, it does this by a simple partition of the data where the first 1001 data points are the training set, the next 200 are the validation set, and the final 300 are the test set. Is this a reasonable approach? Why or why not? If not, what would be a more appropriate partition strategy?

Q2) 1.5pts Consider the function find_pca_matrix provided in the starter code. This function takes the training set and forms a PCA matrix $\mathbf{U}$ such that $\mathbf{U}^T(\mathbf{y} - \mathbf{b}) = \mathbf{z}$. Complete the implementation of this function. Use a latent state dimension of $\dim(\mathbf{z}) = 4$, encode the training/validation/test sets, and plot the resulting encoded state over time. For the state $\mathbf{y}_f$ at the final time step in the test set, report the mean squared error between it and the reconstructed state $\tilde{\mathbf{y}}_f = \mathbf{U}\mathbf{U}^T(\mathbf{y}_f - \mathbf{b}) + \mathbf{b}$. The GP model will be formed using the encoded state $\mathbf{z}$ as the target.

Q3) 1.5pts Consider the function gp_prediction provided in the starter code. This function forms a GP model given a training set of feature-target pairs and returns target predictions on the test set. This is similar to the provided code in gaussian_process_implementation.ipynb, however the provided code is designed for scalar targets. Because we are dealing with a vector target, we form separate GP predictions for each component. Write a version of gp_prediction that forms separate posterior μ and Σ in this way. Assign noise_var a fixed value of 10^{-6}.

Q4) 1.5pts Consider the function gp_evidence provided in the starter code. This function estimates the GP log evidence (log marginal likelihood) over a given dataset given hyperparameters. Again, this is similar to the provided code in gaussian_process_implementation.ipynb, however the provided code is designed for scalar targets. As with gp_prediction, write a version of gp_evidence that calculates a log-evidence term for each component of the target vector.

Q5) 3.5pts For each component GP, use a squared-exponential kernel with unknown parameters σ_i^2 and θ_i for $i = 1, \ldots, \dim(\mathbf{z})$. Using the training dataset, perform Type-II inference by maximizing the log evidence of each component in terms of σ_i^2, θ_i. You may use any optimization algorithm of your choice, including functions from numpy or scipy. Repeat this process using a kernel of your choice (recall that kernels can be defined as a sum of other kernels), and compare its mean squared error to that of the squared-exponential kernel on the validation set. Plot the GP predictions from each on the validation set with uncertainty bounds of one standard deviation. Also show the true encoded states for the validation set.

Q6) 3pts Using the best kernel and hyperparameters found in Q5, plot the GP prediction over the validation and test sets with uncertainty bounds of one standard deviation. Also show the true encoded states for the validation and test sets. At the final time step of the test set, decode the prediction mean via $\mathbf{U}\mathbf{z} + \mathbf{b}$, reshape it onto the 160×80 grid, and show the result compared to the true state. How accurate is it? The plots should be formatted like Figure 1. Note that because $\mathbf{U}\mathbf{z} + \mathbf{b}$ is a linear operation and $\mathbf{z}$ is a Gaussian random variable, the decoded state is also a Gaussian random variable. Decode the prediction variance as well, reshape it onto the 160×80 grid, and plot it. Where is the variance the highest and why?

Submission guidelines: Submit an **electronic copy** of your report (**maximum 10 pages** in at least 10pt font) in **pdf** format and **documented** Python scripts. You should include a file named "README" outlining how the scripts should be run. Upload both your report in pdf format and a single tar or zip file containing your code and README to Quercus. You are expected to verify the integrity of your tar/zip file before uploading. Do not include (or modify) the supplied *.npz data files or the data_utils.py module in your submission. The report must contain

- Objectives of the assignment
- A brief description of the structure of your code, and strategies employed
- Relevant figures, tables, and discussion

Do not use scikit-learn for this assignment, the intention is that you implement the simple algorithms required from scratch. Also, for reproducibility, always set a seed for any random number generator used in your code. For example, you can set the seed in numpy using `numpy.random.seed`.