Stochastic Variational Inference for Gaussian Process Regression of the Forced Isotropic Turbulence data

Adhithiaram Hariharan

***Abstract*— This paper presents a novel approach to address the computational challenges inherent in Gaussian process regression (GPR) models when applied to large-scale datasets, specifically focusing on the domain of turbulent fluid dynamics. Leveraging the versatility of stochastic variational inference, we introduce a stochastic variational Gaussian process (SVGP) regression model tailored for scalability and computational efficiency. Our methodology employs a judicious selection of inducing points to construct a variational approximation of the Gaussian process posterior, significantly reducing computational overhead while preserving predictive accuracy. By integrating sparse kernel techniques and stochastic optimization procedures, we achieve a seamless optimization process that simultaneously tunes kernel hyperparameters and variational parameters. Through empirical validation on a substantial time-series dataset from the Johns Hopkins Turbulence Databases, we demonstrate the effectiveness of our approach in delivering accurate predictions alongside uncertainty estimates. Our model facilitates efficient data handling through batch optimization and provides reliable uncertainty quantification through the generation of confidence intervals around forecasted values. This work represents a significant step towards reconciling computational constraints with the demands of complex fluid dynamics simulations, offering transformative insights for a wide range of engineering and scientific applications.**

I. INTRODUCTION

The quest for high-fidelity computational fluid dynamics (CFD) simulations necessitates a delicate balance between accuracy and computational efficiency, particularly in scenarios demanding extensive temporal analyses. Traditional approaches, while remarkably precise, often succumb to prohibitive computational costs, especially when tasked with repeated iterations typical in computations such as total kinetic energy and Reynolds number assessments across varying time intervals. To surmount this challenge, we advocate for integrating advanced inference methodologies, particularly Gaussian Process Regression (GPR), to achieve both accuracy and computational savings.

In this paper, we embark on a journey to harness the power of Gaussian processes in the context of forced isotropic turbulence, leveraging the rich dataset available from the Johns Hopkins Turbulence Databases. This dataset encapsulates the temporal evolution of crucial parameters, including total kinetic energy and Taylor-scale Reynolds number, presenting an ideal testbed for our methodology.

Adhithiaram Hariharan is with the Department of Aerospace Engineering at the University of Michigan, Ann Arbor. Email: adhith@umich.edu

Special thanks to Dr. Alex Gorodetsky and the Department of Aerospace Engineering at the University of Michigan, Ann Arbor, USA.

At the heart of our approach lies the predictive distribution derived from the multivariate Gaussian conditional rule, furnishing us with a robust framework to model the dynamics of turbulent flows. GPR's most important part is the training process, wherein an optimal set of covariance function parameters, referred to as model hyperparameters, are sought.

To achieve this, we delve into the realm of stochastic variational inference for Gaussian Process Regression (SVGPR), an approximate inference technique tailored for large-scale datasets. SVGPR offers scalability by leveraging a sparse set of inducing points to approximate the true posterior distribution over functions, thereby circumventing the computational bottlenecks inherent in conventional GP regression models.

In our pursuit of elucidating the uncertainty associated with our model, we draw inspiration from seminal literature, including works on Gaussian Processes for Big Data [1] and Bayesian Approaches for Efficient and Uncertainty-Aware Prediction of Pressure Distributions [2]. Guided by these insights, we explore two primary choices of covariance functions, the squared-exponential (SE) and Matern 3/2 kernels—each characterized by distinct hyperparameters governing output standard deviation and length scale.

Our methodology unfolds through a meticulously orchestrated process of data splitting, inducing point selection, and hyperparameter optimization, culminating in the training of the SVGPR model. Through rigorous experimentation, we scrutinize the impact of inducing data set sizes on model accuracy, measured through mean squared error (MSE), thus illuminating the nuanced interplay between data sparsity and predictive performance. As we navigate through the intricacies of our methodology, it is imperative to acknowledge the inherent limitations of our results. While SVGPR offers scalable approximations to the true posterior distributions, the uncertainty inherent in our predictions necessitates the derivation of confidence intervals for a detailed analysis.

In essence, our endeavor seeks to bridge the difference between computational complexity and predictive accuracy in the realm of turbulent flow dynamics, offering a pathway towards more efficient and uncertainty-aware simulations. Through empirical validation and meticulous analysis, we strive to contribute to the broader discourse surrounding the fusion of advanced inference methodologies with complex, real-world datasets, thereby paving the way for transformative advancements in computational fluid dynamics.

II. SVI FOR GPR

Gaussian processes (GPs) are a versatile probabilistic model used for regression tasks. A GP is defined by its mean function $m(x)$ and covariance function $k(x, x')$. While the mean function is typically assumed to be zero over the domain of interest ($m(x) = 0$), the covariance function captures the underlying structure of the function described by the GP prior. In essence, a GP characterizes a probability distribution over functions, where any finite set of function values follows a joint Gaussian distribution.

To accommodate noisy observations, we introduce an observation variable $y(x) = f(x)+\epsilon$, where ϵ is independent and identically distributed Gaussian noise with zero mean and variance τ^2. Given a set of N input points represented by the matrix $X = [x_1, x_2, \ldots, x_N]^T$, the corresponding output values $f(X) = [f(x_1), f(x_2), \ldots, f(x_N)]^T$, and a matrix of evaluation points X^*, the joint normal distribution over the random variable vectors y and $f^* = f(X^*)$ is expressed as follows:

$$\mu(X^*) = m(X^*)+k(X^*, X)(k(X, X)+\tau^2 I)^{-1}(y-m(X))$$

$$\sigma^2(X^*) = k(X^*, X^*)-k(X^*, X)(k(X, X)+\tau^2 I)^{-1}k(X, X^*)$$

Training the GP model involves determining an appropriate set of covariance function parameters denoted by the vector θ. A common hyperparameter learning approach is maximizing the likelihood function $p(y|\theta)$ using gradient-based algorithms. The log-likelihood is given by:

$$\log p(y|X, \theta) = -\frac{1}{2}y^T \left[k(X, X) + \tau^2 I\right]^{-1} y....$$

$$-\frac{1}{2}\log\left|k(X, X) + \tau^2 I\right| - \frac{N}{2}\log(2\pi).$$

Notably, the matrix inversion of $k(X, X) + \tau^2 I$ involved in the regression model entails computational complexity scaling with $O(N^3)$ operations, which poses scalability challenges for large datasets.

To address this, Stochastic Variational Inference for Gaussian Processes (SVGP) offers a solution for large datasets. SVGP is an approximate inference method that employs a small set of inducing points to approximate the true posterior distribution over functions. These inducing points, with a number much smaller than the dataset size ($N_{ind} \ll N$), effectively summarize the dataset.

They are chosen to maximize a lower bound on the log marginal likelihood of the data, and the posterior distribution is approximated by a Gaussian distribution with a mean and covariance matrix dependent on these inducing points. During training, the inducing point locations Z and the covariance parameters θ are optimized to minimize the Kullback-Leibler (KL) divergence. The predictive distribution for a new point X^* can be efficiently computed using the inducing points and their covariance matrix.

The KL divergence is calculated using the following equation:

$$KL = 0.5(\mathrm{tr}(\Sigma^{-1}K_{mm}) - M + \mathbf{m}^\top\Sigma^{-1}\mathbf{m}$$

$$-\log|\det K_{mm}| + \log|\det\Sigma|)$$

where:

- Σ^{-1} is the inverse of the sum of the variational covariance matrix S and the kernel matrix among the inducing points K_{mm}.
- K_{mm} is the kernel matrix computed at the inducing points, reflecting prior beliefs about the function's properties over these points.
- M is the number of inducing points.
- $\mathbf{m}$ is the mean vector of the approximate posterior distribution over the inducing points.
- $\mathrm{tr}(\cdot)$ denotes the trace of a matrix, which sums the diagonal elements.
- $\log|\det(\cdot)|$ represents the natural logarithm of the determinant of a matrix, which in this context measures the volume change when linearly transforming the space defined by the Gaussian distributions.

The formula is used to optimize the parameters of the model by minimizing the KL divergence, thus making the approximate posterior as close as possible to the true posterior given the inducing points used.

III. PROBLEM SETUP

We utilize the Johns Hopkins Turbulence Databases (JHTDB), an openly available, well-documented repository of simulated turbulence data. The JHTDB provides DNS data for forced isotropic turbulence, simulating an incompressible fluid's behavior with no preferential flow direction and subject to continual energy input at large scales. The data includes the time history of total kinetic energy and Taylor-scale Reynolds number among other variables, which are functions of spatial positions and time, offering a rich set to train and validate our models. The data consists of Reynolds energy values for 500 timesteps between 0 to 10 seconds.

The GPR model is implemented through two distinct approaches. Initially, we conduct a standard Gaussian Process Regression analysis using all 500 data points. Subsequently, we execute a Sparse Variational Gaussian Process Regression (SVGPR) without relying on pre-built libraries, instead applying inducing points and mini-batch training techniques. As the observations are deterministic, noise-free predictions are considered.

The kernels selected for standard Gaussian Process Regression are the squared-exponential (SE) and Matern3/2 (M32)

$$k_{SE}(x, x') = \sigma^2 \exp\left(-\sum_{k=1}^{d}\frac{(x_k - x'_k)^2}{2l_k^2}\right)$$

$$k_{M_{3/2}}(x, x') = \sigma^2 \left(1 + \sum_{k=1}^{d} \frac{\sqrt{3}|x_k - x'_k|}{l_k}\right) ....$$

$$\exp\left(-\sum_{k=1}^{d} \frac{\sqrt{3}|x_k - x'_k|}{l_k}\right)$$

where the model hyperparameters are the output standard deviation σ and a length scale ℓ_k per dimension, denoted by the vector $\theta = [\sigma, \ell_k]$, previously defined. The length scale for each dimension of the design space was found through various iterations of minimising the optimization function. The highest accuracy on the validation set has been found with both the covariance functions for deterministic approach, so squared exponential kernel will therefore be discussed hereafter.

To comprehensively evaluate the performance of these models, we consider the mean-squared error (MSE):

$$MSE = \frac{1}{N_t} \sum_{n=1}^{N_t} (y_n - \hat{y}_n)^2$$

where y represents the true value, y_n represents the GPR predicted mean and N_t is the number of testing points. Lower MSE values are desired, and for various inducing points, the MSE values are analysed.

IV. ALGORITHM DESIGN

Algorithm 1 represents the overall framework I have implemented to perform the Stochastic Variational Inference for Gaussian Process Regression of the forced isotropic turbulence data. The algorithm is explained in detail below.

Algorithm 1 Stochastic Variational Inference for Gaussian Process Regression

Input: Training data $\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}$, initial hyperparameters $\boldsymbol{\theta}$, mini-batch size b
Output: Optimized hyperparameters, inducing point values, predictive mean, and variance
procedure INITIALIZE SVI($\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}, \boldsymbol{\theta}$)
    Choose number of inducing points M
    Initialize inducing points $\mathbf{Z}$ from $\mathbf{X}_{\text{train}}$
    Initialize variational parameters $\mathbf{m}, \mathbf{S}$
end procedure
procedure KERNEL FUNCTION($\mathbf{X}, \mathbf{X}', \boldsymbol{\theta}$)
    Define kernel function $k(\mathbf{X}, \mathbf{X}'; \boldsymbol{\theta})$
    Compute covariance matrix $\mathbf{K}$
    return $\mathbf{K}$
end procedure
procedure COMPUTE ELBO($\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{m}, \mathbf{S}, \boldsymbol{\theta}$)
    Compute KL divergence term KL
    Compute expected log-likelihood $E_q[\log p(\mathbf{Y}|\mathbf{X}, \mathbf{f})]$
    return ELBO $= E_q[\log p(\mathbf{Y}|\mathbf{X}, \mathbf{f})] - \text{KL}$
end procedure
Initialize SVI($\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}, \boldsymbol{\theta}$)
repeat
    Randomly select mini-batch $\mathbf{X}_b, \mathbf{Y}_b$ from $\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}$
    ELBO $\leftarrow$ ComputeELBO($\mathbf{X}_b, \mathbf{Y}_b, \mathbf{Z}, \mathbf{m}, \mathbf{S}, \boldsymbol{\theta}$)
    $\boldsymbol{\theta}, \mathbf{m}, \mathbf{S} \leftarrow$ Update parameters to maximize ELBO
until convergence
procedure PREDICT($\mathbf{X}_{\text{test}}, \boldsymbol{\theta}_{\text{opt}}, \mathbf{Z}, \mathbf{m}, \mathbf{S}$)
    Compute predictive mean and variance for $\mathbf{X}_{\text{test}}$
    return Predictive mean and variance
end procedure
Predictive mean, variance $\leftarrow$Predict($\mathrm{X}_{\text{test}}, \boldsymbol{\theta}_{\text{opt}}, \mathbf{Z}, \mathbf{m}, \mathbf{S}$)
Display predictive distribution

A. Overview

The Sparse Variational Gaussian Process (SVGP) is a scalable variant of the Gaussian Process (GP), enabling the application of GP models to large datasets. The SVGP leverages a set of (N_{ind}) inducing points, ($\mathbf{Z}$), which are pseudo-inputs that serve to summarize the dataset efficiently. These inducing points approximate the full GP by capturing the essential information contained within the data, thereby reducing computational complexity.

B. Inducing Points and Hyperparameters

The choice of inducing points and the number (N_{ind}) profoundly influence the quality of the approximation and computational performance, striking a balance between accuracy and efficiency. The inducing points, ($\mathbf{Z}$), and hyperparameters ($\theta = [\sigma, \ell_k]$) are optimized by maximizing the Evidence Lower Bound (ELBO), a surrogate objective function balancing model fitness with the complexity of the solution.

C. Variational Framework

To approximate the posterior distribution ($p(f(\cdot) \mid \mathbf{D})$), we exploit the assumed redundancy in the data ($\mathbf{D}$) by relying on function values at inducing inputs ($\mathbf{z}$) instead of all observations ($\mathbf{x}$). This approximation comes with computational savings, as we invert ($\mathbf{K}zz$) instead of ($\mathbf{K}xx$), which is less expensive for ($N_{ind} \ll N$) (number of data points). The approximate posterior ($q(f(\mathbf{z}))$) is parametrized as a Gaussian with mean ($\mathbf{m}$) and covariance ($\mathbf{S}$), and the inducing inputs ($\mathbf{z}$) can be optimized as variational parameters.

D. Stochastic Optimization

To infer the optimal parameters ($\mathbf{z}, \mathbf{m}, \mathbf{S}$), we maximize the ELBO with respect to these variational parameters and model hyperparameters. Even SVGP inference can be computationally demanding for large datasets, so we employ mini-batch optimization techniques. This approach allows the model to be updated using a subset of the data at each optimization step, greatly enhancing scalability and practicality for large-scale problems.

V. IMPLEMENTATION DETAILS

The ELBO function is defined as the difference between the log-likelihood of the data ($\mathbf{Y}$) and the Kullback-Leibler (KL) divergence term that regularizes the distance between the approximate and true posteriors. It is expressed as

$$ELBO = \mathcal{L} - KL$$

,where ($\mathcal{L}$) refers to the log likelihood, and KL to the Kullback-Leibler divergence. We implement gradient-based optimization using the L-BFGS-B algorithm to maximize the ELBO. The inducing points and GP hyperparameters are initialized randomly from the training data and refined through the optimization process.

We compute the mean and variance over a grid of test inputs to assess the model's predictive performance, utilizing the learned inducing points and kernel hyperparameters. This provides a probabilistic forecast along with an uncertainty estimate. Using the optimized GP and variational parameters, test set predictions are made. The predicted means and variances are evaluated across a test input grid to visualize the model's predictions compared to the actual data.

Model performance is quantitatively assessed using metrics such as the mean squared error (MSE) between the true values and the model's predictions. This metric allows us to evaluate how well the model captures the underlying function of the data. Results are visualized by plotting the trained models' predictions, the inducing points, and the original training data. Confidence intervals representing prediction uncertainties provide added insight into the model's confidence in its predictions.

VI. RESULTS

We evaluate the fit of the Sparse Variational Gaussian Process (SVGP) to the training dataset, highlighting its capacity to capture the underlying patterns in the data. The training set, consisting of time-series data on energy consumption, serves as a basis for assessing the initial performance.

To verify the implementation of kernels, a small noise of 10^{-3} is used, and the Matern 3/2 covariance function (k_{M32}) shows better predictions as it is favored by the validation set and it is indicative of the smoothness properties of the estimated function.

As we are using a deterministic model to verify the effects of inducing points, we implement the squared exponential kernel. 100 inducing points are used with hyperparameters σ as 4.3 and l_k as 0.0025 for the SVGPR model.

The selection of inducing points is critical for our method's efficacy and efficiency. The results from experiments with (N_{ind}) set to 10, 50, and 100 inducing points are presented and compared. These points are visualized in the context of the actual training data, allowing a visual assessment of how well these points summarize the dataset and the decision process behind selecting an appropriate number of inducing points.

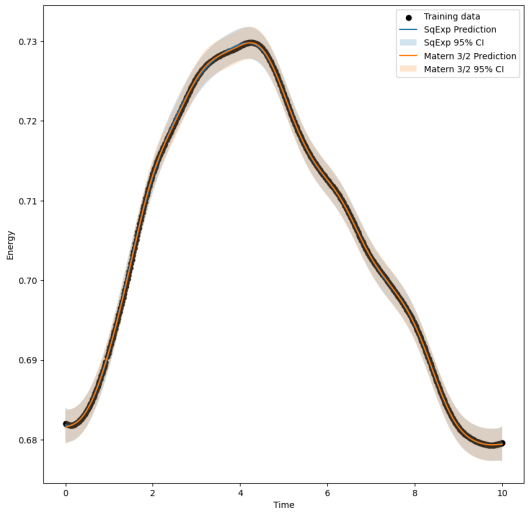

Fig. 1. Energy vs. Time prediction using Gaussian Process Regression

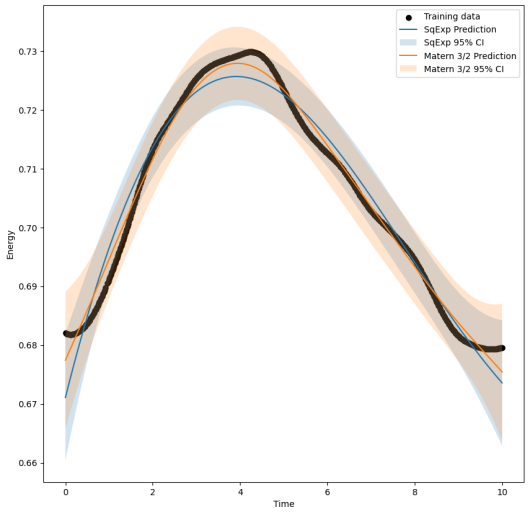

Fig. 2. Gaussian Process Regression for different kernels

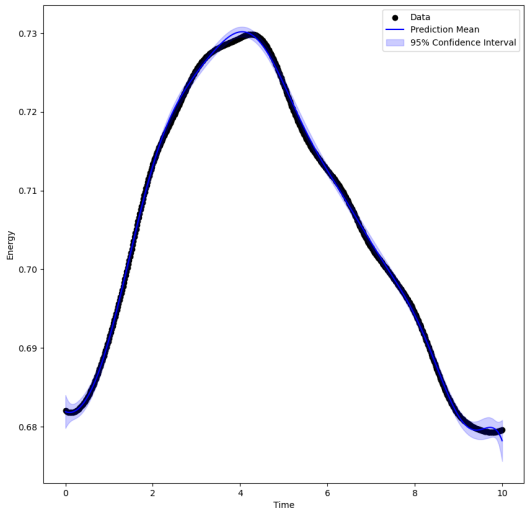

Fig. 3. SVGPR using Squared Exponential Kernel

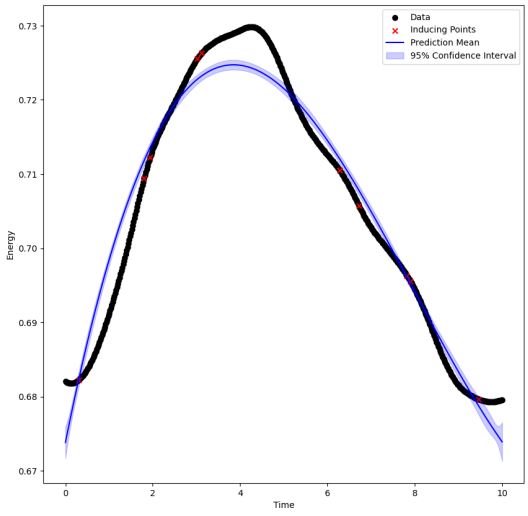

Fig. 4. SVGPR for 10 inducing points

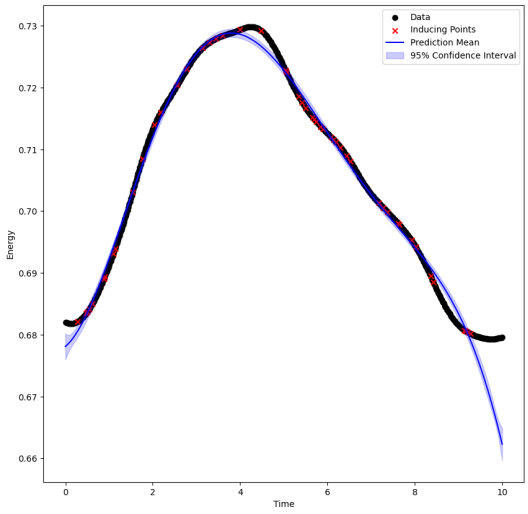

Fig. 5. SVGPR for 50 inducing points

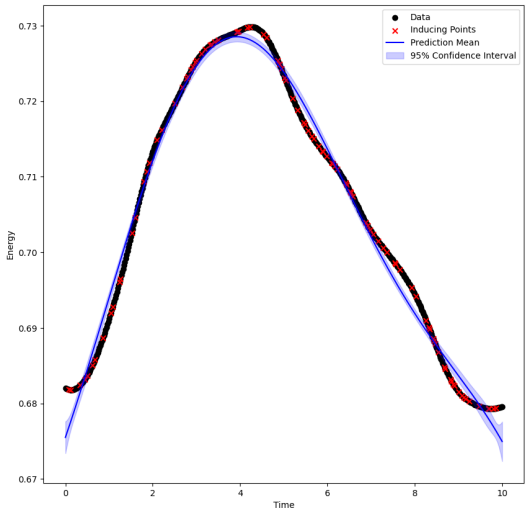

Fig. 6. SVGPR for 100 inducing points

We utilize the mean squared error (MSE) as a quantitative measure to compare the model's performance across different numbers of inducing points. A lower MSE indicates a more accurate approximation of the training data, with results indicating a general trend of improved accuracy as more inducing points are used.

Number of Inducing Points	MSE
10	1.075×10^{-5}
50	7.883×10^{-6}
100	2.968×10^{-6}

We can see from the table that the MSE values decrease as the number of inducing points increases. The model's predictive capability is showcased with plots illustrating the predicted mean energy over time. These plots are supplemented with confidence intervals depicting the model's uncertainty, corresponding to a 95% confidence level. Across the range of inducing points investigated, we analyze how the variance in predictions and the width of the confidence intervals are affected, offering insight into the trade-off between prediction accuracy and computational load.

From Fig 6, it is evident that a configuration with 100 inducing points provides a robust approximation that balances model complexity with predictive accuracy.

VII. CONCLUSIONS

The study presented in this paper successfully demonstrates the application of a Stochastic Variational Gaussian Process (SVGP) approach to Gaussian process regression within the context of large-scale computational fluid dynamics. Our novel SVGP framework circumvents the computational limitations traditionally associated with Gaussian process regression, optimizing via a well-considered selection of inducing points that succinctly summarize the dataset. The inference results, predicated on the invaluable data from the Johns Hopkins Turbulence Databases, reveal that with an informed choice of 100 inducing points, the model adeptly manages to preserve a high level of predictive accuracy while reducing computational overhead, as evidenced by the lower mean squared errors.

The consistency of low MSE values across the varying number of inducing points underpins the robustness of our approach, with the lesser computational expense and efficient handling of extensive data through stochastic optimization. Our findings underscore the significant improvement in predictive performance and decreased uncertainty, validating the SVGP model as a powerful analytical tool that bridges the gap between computational feasibility and high-fidelity modeling for CFD applications.

In closing, incorporating the SVGP model into turbulence simulation workflows signals a transformative advancement in the field, achieving scalable model complexity and a quantifiable representation of uncertainty. This combination is particularly compelling for applications where accurate forecasts and an understanding of associated confidence levels are critical.

ACKNOWLEDGMENT

I wish to express our sincere gratitude to the Johns Hopkins Turbulence Databases team for providing open access to their high-quality datasets, which were instrumental in conducting this research. I extend my heartfelt thanks to Professor Alex Gorodetsky for his invaluable teachings and mentorship throughout the duration of this course.

REFERENCES

[1] Mehdi Anhichem and Sebastian Timme, "Bayesian Approaches for Efficient and Uncertainty-Aware Prediction of Pressure Distributions" https://doi.org/10.2514/6.2024-0253

[2] James Hensman, Nicolo Fusi, Neil D. Lawrence, "Gaussian Processes for Big Data" https://doi.org/10.48550/arXiv.1309.6835

[3] Matthew Hoffman, David M. Blei, Chong Wang, and John Paisley. "Stochastic variational inference" https://doi.org/10.48550/arXiv.1206.7051

[4] Hensman, J., Matthews, A., and Ghahramani, Z., "Scalable variational Gaussian process classification" https://doi.org/10.48550/arXiv.1411.2005

[5] Andrew Gordon Wilson, David A. Knowles, Zoubin Ghahramani, "Gaussian Process Regression Networks" https://doi.org/10.48550/arXiv.1110.4411

APPENDIX

Python is used to implement the SVGPR algorithm

Standard GPR

```
    import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from numpy.linalg import cholesky, solve

def sqexp_kernel(a, b, tau, l):
    a = a.reshape(-1, 1)
    b = b.reshape(-1, 1)
    sqdist = (a - b.T)**2
    return tau ** 2 * np.exp(-0.5 * sqdist / (l ** 2))

def matern32_kernel(a, b, tau, l):
    a = a.reshape(-1, 1)
    b = b.reshape(-1, 1)
    abs_diff = np.abs(a - b.T)
    sqrt3 = np.sqrt(3)
    return tau ** 2 * (1 + sqrt3 * abs_diff / l) * np.exp(-sqrt3 * abs_diff / l)

def log_marginal_likelihood(kernel, params, x, y, noise_var):
    tau, l = params
    K = kernel(x, x, tau, l) + noise_var * np.eye(len(x))
    L = cholesky(K + 1e-6 * np.eye(len(K)))
    alpha = solve(L.T, solve(L, y))
    log_likelihood = -0.5 * np.dot(y.T, alpha)
    log_likelihood -= np.sum(np.log(np.diagonal(L)))
    log_likelihood -= len(x) / 2 * np.log(2 * np.pi)
    return -log_likelihood

def predict(kernel, x_pred, x, y, params, noise_var):
    tau, l = params
    K = kernel(x, x, tau, l) + noise_var * np.eye(len(x))
    K_s = kernel(x, x_pred, tau, l)
    K_ss = kernel(x_pred, x_pred, tau, l) + 1e-6 * np.eye(len(x_pred))
    L = cholesky(K + 1e-6 * np.eye(len(K)))
    Lk = solve(L, K_s)
    mu = np.dot(Lk.T, solve(L, y))
    S2 = np.diag(K_ss) - np.sum(Lk**2, axis=0)
    return mu, S2

data = np.loadtxt('DATA.dat.txt')
time_data = data[:, 0]
energy = data[:, 1]
np.random.seed(0)
train_indices = np.random.choice(len(time_data), 500, replace=False)
X_train = time_data[train_indices]
y_train = energy[train_indices]
init_params = np.array([1.0, 1.0])
noise_var = 0

res_sqexp = minimize(fun=lambda params: log_marginal_likelihood(sqexp_kernel, params,
                                                X_train, y_train, noise_var),
                x0=init_params, method='L-BFGS-B', bounds=((1e-5, None), (1e-5, None)))
opt_params_sqexp = res_sqexp.x

res_matern32 = minimize(fun=lambda params: log_marginal_likelihood(matern32_kernel,
                                                params, X_train, y_train, noise_var),
                x0=init_params, method='L-BFGS-B', bounds=((1e-5, None), (1e-5, None)))
opt_params_matern32 = res_matern32.x

time_pred = np.linspace(time_data.min(), time_data.max(), 500)
mu_sqexp, S2_sqexp = predict(sqexp_kernel, time_pred, X_train[:, np.newaxis], y_train,
                                                opt_params_sqexp, noise_var)
```

```
mu_matern32, S2_matern32 = predict(matern32_kernel, time_pred, X_train[:, np.newaxis],
                                   y_train, opt_params_matern32, noise_var)

plt.figure(figsize=(10,10))
plt.scatter(X_train, y_train, c='k', label='Training data')
plt.plot(time_pred, mu_sqexp, label='SqExp Prediction')
plt.fill_between(time_pred, mu_sqexp - 1.96 * np.sqrt(S2_sqexp), mu_sqexp + 1.96 * np.
                                   sqrt(S2_sqexp), alpha=0.2, label='SqExp 95%
                                   CI')

plt.plot(time_pred, mu_matern32, label='Matern 3/2 Prediction')
plt.fill_between(time_pred, mu_matern32 - 1.96 * np.sqrt(S2_matern32), mu_matern32 + 1.
                                   96 * np.sqrt(S2_matern32), alpha=0.2, label=
                                   'Matern 3/2 95% CI')
# plt.title('Gaussian Process Regression with Different Kernels')
plt.xlabel('Time')
plt.ylabel('Energy')
plt.legend()
plt.show()

# Optimal parameters for the Squared Exponential Kernel
print("Optimal parameters for SqExp Kernel:")
print("Tau (amplitude scale):", opt_params_sqexp[0])
print("L (length scale):", opt_params_sqexp[1])

# Optimal parameters for the Mat rn 3/2 Kernel
print("Optimal parameters for Matern 3/2 Kernel:")
print("Tau (amplitude scale):", opt_params_matern32[0])
print("L (length scale):", opt_params_matern32[1])
```

SVGPR with inducing points and batch iterations

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy.linalg import cholesky, solve_triangular
from numpy.linalg import inv

def kernel(a, b, tau, l):
    a = a.reshape(-1, 1)
    b = b.reshape(-1, 1)
    sqdist = (a - b.T)**2
    return tau ** 2 * np.exp(-0.5 * sqdist / l**2)

def stable_cholesky(K):
    jitter = 1e-6
    while True:
        try:
            L = cholesky(K, lower=True)
            return L
        except np.linalg.LinAlgError:
            K += jitter * np.eye(K.shape[0])
            jitter *= 10

data = np.loadtxt('DATA.dat.txt')
X_train = data[:, 0][:, None]
Y_train = data[:, 1][:, None]

np.random.seed(0)
lengthscale = 0.0025
variance = 4.3

num_inducing_points_options = [10, 50, 100]

for num_inducing_points in num_inducing_points_options:
    inducing_indices = np.random.choice(X_train.shape[0], num_inducing_points, replace=
                                        False)
    inducing_points = X_train[inducing_indices]
```

```
inducing_targets = Y_train[inducing_indices]

m = np.zeros(num_inducing_points)
S = np.eye(num_inducing_points)
params = np.concatenate([np.array([lengthscale, variance]), m, S[np.triu_indices(
                                                num_inducing_points)]])

def elbo(params, X, Y, Z):
    lengthscale, variance = params[0], params[1]
    m = params[2:2+len(Z)].flatten()
    S_flat = params[2+len(Z):]
    S = np.zeros((len(Z), len(Z)))
    S[np.triu_indices(len(Z))] = S_flat
    S = S + S.T - np.diag(np.diag(S))
    Kmm = kernel(Z, Z, lengthscale, variance) + np.eye(len(Z)) * 1e-6
    Kmn = kernel(Z, X, lengthscale, variance)
    L = stable_cholesky(Kmm)
    Kmm_inv = inv(Kmm)
    Kmn_Kmm_inv = np.dot(Kmn.T, Kmm_inv)
    Sigma = S + Kmm
    Sigma_inv = inv(Sigma)
    log_likelihood = -0.5 * Y.T.dot(Y) + 0.5 * Y.T.dot(Kmn_Kmm_inv).dot(m)
    KL = 0.5 * (np.trace(Sigma_inv.dot(Kmm)) - len(Z) + m.dot(Sigma_inv).dot(m)
                - np.linalg.slogdet(Kmm)[1] + np.linalg.slogdet(Sigma)[1])

    return -(log_likelihood - KL)

def optimize_elbo(params, X, Y, Z):
    result = minimize(elbo, params, args=(X, Y, Z), method='L-BFGS-B', options={'
                                                    maxiter': 100, 'disp': False})
    return result.x

optimized_params = optimize_elbo(params, X_train, Y_train, inducing_points)
lengthscale_opt, variance_opt = optimized_params[0], optimized_params[1]

Kmm_opt = kernel(inducing_points, inducing_points, variance_opt, lengthscale_opt) +
                                                np.eye(num_inducing_points) * 1e-6
L_opt = stable_cholesky(Kmm_opt)
alpha = solve_triangular(L_opt.T, solve_triangular(L_opt, inducing_targets, lower=
                                                True))

X_pred = np.linspace(X_train.min(), X_train.max(), 500)[:, np.newaxis]
Knm_pred = kernel(X_pred, inducing_points, variance_opt, lengthscale_opt)
mean_pred = Knm_pred.dot(alpha)

mean_pred = mean_pred.flatten()
std_pred = 1.96 * np.sqrt(var_pred)

plt.figure(figsize=(10,10))
plt.scatter(X_train, Y_train, c='k', label='Data')
plt.scatter(inducing_points, inducing_targets, c='red', marker='x', label='Inducing
                                                Points')
plt.plot(X_pred.flatten(), mean_pred, color='blue', label='Prediction Mean')
plt.fill_between(X_pred.flatten(), (mean_pred - std_pred), (mean_pred + std_pred),
                                                color='blue', alpha=0.2, label='95%
                                                Confidence Interval')
plt.xlabel('Time')
plt.ylabel('Energy')
plt.legend()
# plt.title('SVGP Regression')
plt.show()

Knm_train = kernel(X_train, inducing_points, variance_opt, lengthscale_opt)
mean_train_pred = Knm_train.dot(alpha)
mse_train = np.mean((Y_train.flatten() - mean_train_pred.flatten())**2)
print(f'Number of Inducing Points: {num_inducing_points}, MSE: {mse_train}')
```