AIAA Aviation
22-26 June 2015, Dallas, TX
22nd AIAA Computational Fluid Dynamics Conference

Machine Learning Methods for Data-Driven Turbulence Modeling

Ze Jia Zhang* and Karthik Duraisamy †
University of Michigan, Ann Arbor, MI 48109, U.S.A.

As part of a larger effort on data-driven turbulence modeling, this paper investigates machine learning models in their capability to reconstruct the functional forms of spatially distributed quantities extracted from high fidelity simulation and experimental data. Such datasets typically involve very high dimensional feature spaces with sparsely populated and noisy data. A new multiscale Gaussian process regression technique is described and is compared to 'conventional' Gaussian process regression and artificial neural networks. All these techniques are applied to the reconstruction of functions arising from Bayesian inference applied to turbulent channel flow and bypass transition. The efficiency, accuracy and effectiveness of each learning algorithm as well as factors that influence their output is assessed. The results highlight the potential of machine learning as an enabling tool in data-driven turbulence modeling.

I. Introduction

Turbulence is prevalent in almost every flow problem of practical interest. Although the governing equations of turbulent flow are well-known, the presence of a wide range of time and length scales present significant challenges in direct simulation, requiring prohibitively expensive mesh and time step resolutions. As a result, much effort has been directed towards the modeling of turbulent flow, with the Reynolds Averaged Navier–Stokes (RANS) equations being the most popular option because of their feasibility in high Reynolds number flows and complex geometries. The resources required to use Large Eddy Simulations (LES) and Direct Numerical Simulations (DNS) for these types of flows are often not readily available, despite continued rapid advances in computational technology. It is, however, well-recognized that RANS models are less accurate than desired in many situations, especially those that involve separation, transition, secondary flows, etc. Our belief is that data from LES, DNS and experiments can be leveraged in a comprehensive manner to improve RANS models.

This work is part of a larger effort to improve closure models[1–3] in which inverse modeling is used to infer the functional forms of modeling discrepancies and machine learning (ML) is used to reconstruct the information from the inference process into modeling terms to be used in a predictive setting. In this paper, we focus on the latter aspect and explore existing and newly-developed machine learning techniques for use in turbulence modeling. Traditionally, alterations made to RANS models are based on a combination of theory, empirical reasoning and calibration using a limited set of flows. Over the past decade, there has been progress made in incorporating machine learning into this process. Yarlanki *et al.*[5] used ML to estimate optimal values for calibration constants in the $k-\omega$ model. Milano and Koumoutsakos[6] used it to perform model reduction on LES simulations of turbulent flow over a flat plate. Finally, Sarghini *et al.*[7] performed a procedure similar to what will be described in this paper, but applied to the improvement of LES subgrid-scale models. In related work, Tracey *et al.* performed a detailed investigation of the capability of neural networks to reconstruct *known* functions in the Spalart Allmaras[4] turbulence model and demonstrated that it is indeed possible to replace analytical representations by machine-learned routines in a production CFD solver.

The goal of our approach is not to calibrate existing constants in RANS models, nor to introduce new constants. Rather, it is to **introduce new terms** to the model equations. These terms that can vary spatially and temporally, and comprehensively alter the equations to produce more accurate results, and are learned directly from data. The terms will have no analytical forms, but rather forms defined by the ML method and the data used to derive them.

*Graduate Research Assistant, Department of Aerospace Engineering, 1320 Beal Avenue, Ann Arbor, MI 48109-2140.
†Assistant Professor, Department of Aerospace Engineering, 1320 Beal Avenue, Ann Arbor, MI 48109-2140. AIAA Member.

Overview of the data-driven approach

The overall procedure, which is explained in Refs. 1, 2 is as follows: To begin, data is obtained from LES, DNS, or experiment. Then, a modified version of the model equation, one that includes a term (or terms) to be inferred is devised. In general, this new term is adjusted to improve RANS solver performance according to some metric. The adjustable term in the RANS model can be determined using, say, a Bayesian inversion process. It must again be emphasized that the goal is to explore the impact of spatio-temporal modifications in the model, and not just parameteric modifications. This adjustable term is then reconstructed into a function form (over features that will be accessible by a predictive RANS solver) using ML. During a predictive process, the solver will query the function whenever the turbulence model is computed. A schematic of this general procedure can be found in Figure 1.

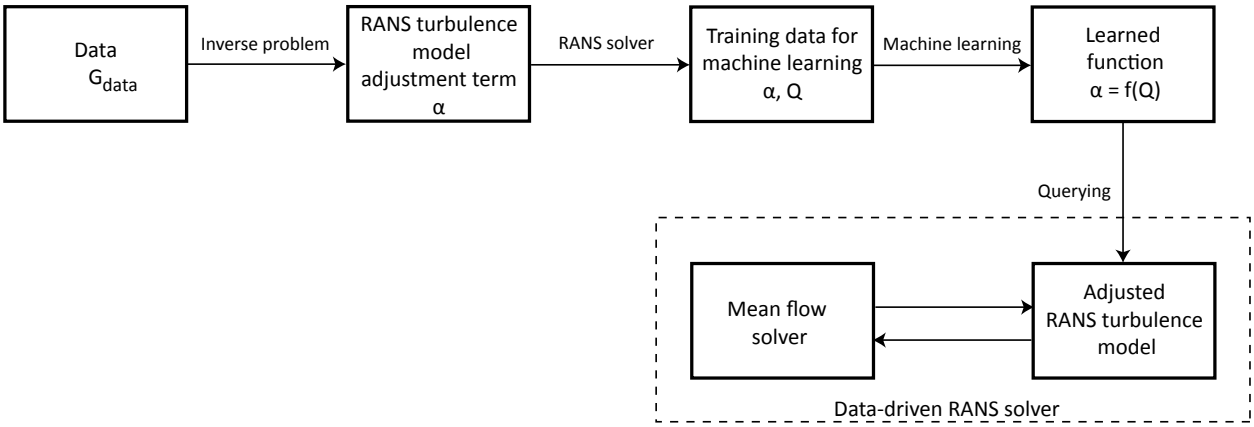


Figure 1: Process for building a data-driven RANS solver. G_{data} is a performance measure. LES, DNS, and experimental data may all be used for the first step. $\mathbf{Q}$ is a set of flow features used to learn α.

This paper will introduce different machine learning techniques and examine the efficacy of each method in the context of turbulence and transition modeling examples.

II. Machine Learning

We use supervised learning techniques, the goal of which is to predict an output y given a vector of inputs $\mathbf{q}$. This is also referred to as regression, since both the input and the output are continuous variables.[12] In this instance, the output y is the factor α (the inferred modeling term or function), and the inputs are $\mathbf{q} = [q_1\ q_2\ \cdots\ q_M]$. These inputs are chosen from a number of possibilities through hill-climbing feature selection,[13] an algorithm that appends inputs to the usable set $\mathbf{q}$ until increasing the size of $\mathbf{q}$ no longer improves the performance[a] when fitting y.

Given a set of outputs $\mathbf{Y}$ and its corresponding set of inputs $\mathbf{Q}$, it is customary[12] to divide them into training, validation, and test sets $(\mathbf{Q}_{\mathrm{train}}, \mathbf{Y}_{\mathrm{train}})$, $(\mathbf{Q}_{\mathrm{val}}, \mathbf{Y}_{\mathrm{val}})$, and $(\mathbf{Q}_{\mathrm{test}}, \mathbf{Y}_{\mathrm{test}})$. A ML algorithm is first applied to the training set to obtain f, where $f(\mathbf{q}) \approx y$. Then, the error on the validation set can be used to adjust any hyperparameters the ML algorithm may possess. In this work, the normalised sum of squared errors (SSE) is used for neural networks:

$$\mathrm{SSE} = \frac{||\mathbf{Y} - f(\mathbf{Q})||^2}{||\mathbf{Y}||^2} \tag{1}$$

Through cross-validation,[12] hyperparameters that minimise the validation SSE are determined. For Gaussian processes, a different metric for error is used, but the cross-validation is the same. Finally, the SSE of the test set is used to compare the performances of different ML algorithms. Sometimes, the complete dataset $(\mathbf{Q}, \mathbf{Y})$ is too large to be used in training at once, not to mention slow for injection purposes. However, since the optimal hyperparameters are found in a way that takes advantage of all available data, randomly choosing a smaller subset of data for training the model that will feed into the RANS solver can be sufficient. In this work, the complete available datasets are used. The inputs are scaled to $[0, 1]$ before being used as training data. Three ML approaches are explored: neural networks (NNs), Gaussian processes (GPs), and multiscale Gaussian processes (MGPs), an extension to GPs. Details on how each approach develops $f(\mathbf{q})$ are presented in the next sections.

[a]The performance metric used in the current work for input selection is the normalised squared error on the validation set.

II.A. Neural Networks

A standard feed-forward neural network[12] (NN) is used. The NN operates by constructing linear combinations of inputs and transforming them through nonlinear activation functions. The process is repeated once for each hidden layer in the network, until the output layer is reached. Figure 2 presents a sample NN. For this sample network, the values of the hidden nodes $z_{1,1}$ through z_{1,H_1} would be constructed as

$$z_{1,j} = a_{(1)}\left(\sum_{i=1}^{3} w_{ij}^{(1)} q_i\right) \tag{2}$$

where $a_{(1)}$ and $w_{ij}^{(1)}$ are the activation function and weights associated with the first hidden layer, respectively. Similarly, the second layer of hidden nodes is constructed as

$$z_{2,j} = a_{(2)}\left(\sum_{i=1}^{H_1} w_{ij}^{(2)} z_{1,i}\right) \tag{3}$$

Finally, the output is

$$f(\mathbf{q}) = a_{(3)}\left(\sum_{i=1}^{H_2} w_{ij}^{(3)} z_{2,i}\right) \approx y \tag{4}$$

Given training data, error backpropagation algorithms[12] are used to find $w_{ij}^{(n)}$.

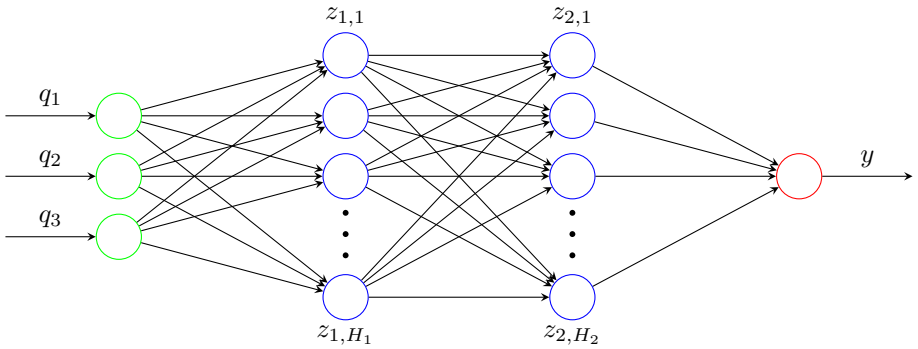


Figure 2: Network diagram for a feed-forward NN with three inputs, two hidden layers, and one output.

One main advantage of NNs over GPs is efficiency. Once the weights are found, computing $f(\mathbf{Q}_{\text{test}})$ depends only on the number of hidden nodes, and not on the (typically high) volume of training data. Hyperparameters of the NN method include the number of hidden layers, the number of nodes in each hidden layer, and the forms of the activation functions. In the current work, the open-source Fast Artificial Neural Network (FANN) library[14] is used for training and for integration with the RANS solver.

II.B. Gaussian Processes

In contrast, the advantage of Gaussian Processes (GPs) is adaptability. Whereas NNs ultimately depend on combinations of predetermined basis functions, GPs are able to construct a basis that suits the the problem, so to speak. A second advantage is that, if the output y is known to noisy, i.e. generated by adding ideally Gaussian noise with variance λ to an original function y_0,

$$y(\mathbf{q}) = y_0(\mathbf{q}) + \mathcal{N}(0, \lambda) \tag{5}$$

obtaining the variance of the ML output f, σ_f^2, becomes very straightforward. This is useful when λ is known, for example, from the resulting distribution of the inverse problem or even from experimental data. Even without that information, λ and σ_f^2 can be estimated. The variances are especially useful for producing error bounds on functions that incorporate the GP, the RANS solver being one such function. Doing the same for NNs is less intuitive.

A GP assumes that the output function at the training points, $\mathbf{Y}_{\text{train}}$, is drawn from the distribution

$$\mathbf{Y}_{\text{train}} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Phi} + \lambda \mathbf{I}) \tag{6}$$

$\mathbf{\Phi}$ is a matrix of covariances between training points. The mean of the distribution is unknown, so it is set to zero. The noise of the output is added to the covariance between training points because they are assumed independent. The current work employs a radial basis function (RBF)[15,16] kernel for computing GP covariance:

$$\mathbf{\Phi}(m,n) = \phi(\mathbf{q}_m, \mathbf{q}_n) = e^{-\frac{||\mathbf{q}_m - \mathbf{q}_n||^2}{h^2}} \tag{7}$$

where $\mathbf{q}_m$, $\mathbf{q}_n$ are points from $\mathbf{Q}_{\text{train}}$. Since the covariance matrix must be populated by computing the value of the the kernel function between all possible pairs of training points, it is evident that, unlike for NNs, computing the output of a GP depends on the size of the training set. Also, note that points that are closer in input space are assumed to be more correlated, and points that are further, less.

Now, given a test point $\mathbf{q}_{\text{test}}$, the corresponding $f(\mathbf{q}_{\text{test}})$ can be defined via a conditional probability distribution, $p(f|\mathbf{Y}_{\text{train}})$. The mean of this distribution is the predicted value of f given the training points, and the variance is σ_f^2. Using standard probability formulae for conditional distributions, it is found that

$$p(f|\mathbf{Y}_{\text{train}}) \sim \mathcal{N}(\underline{\phi}^T(\mathbf{\Phi} + \lambda\mathbf{I})^{-1}\mathbf{Y}_{\text{train}}, 1 + \lambda - \underline{\phi}^T(\mathbf{\Phi} + \lambda\mathbf{I})^{-1}\underline{\phi}) \tag{8}$$

and hence, both the expected value and variance of f are obtained:

$$f = \underline{\phi}^T(\mathbf{\Phi} + \lambda\mathbf{I})^{-1}\mathbf{Y}_{\text{train}} \tag{9}$$

$$\sigma_f^2 = 1 + \lambda - \underline{\phi}^T(\mathbf{\Phi} + \lambda\mathbf{I})^{-1}\underline{\phi} \tag{10}$$

$\underline{\phi}$ is a vector whose elements are $\phi(\mathbf{q}_{\text{train},i}, \mathbf{q}_{\text{test}})$, with $\mathbf{q}_{\text{train},i}$ being the ith data point in $\mathbf{Q}_{\text{train}}$. Given the form of the kernel function, the value of f will be closer to the values of y_{train} for which $\mathbf{q}_{\text{train}}$ is close to $\mathbf{q}_{\text{test}}$.

In cases where λ is supplied, σ_f^2 can be found directly. If there is a different λ assigned to each y_{train}, the scalar λ may be replaced by the vector $\mathbf{\Lambda}$ consisting of all the individual variances at the training points, and the rest of the equations would still hold. In cases where λ is unknown, it can be estimated by maximising the log marginal likelihood (LML):

$$\log p(\mathbf{Y}_{\text{train}}|\mathbf{Q}_{\text{train}}, h, \lambda) = -\frac{1}{2}\log|\mathbf{\Phi} + \lambda| - \frac{1}{2}\mathbf{Y}_{\text{train}}(\mathbf{\Phi} + \lambda)^{-1}\mathbf{Y}_{\text{train}} - \frac{N}{2}\log(2\pi) \tag{11}$$

where N is the number of training points. The LML measures the probability that an estimate of the hyperparameters h and λ would yield a distribution that produces $\mathbf{Y}_{\text{train}}$ when sampled at $\mathbf{Q}_{\text{train}}$, so maximising it would mean finding the most suitable values for h and λ. For GPs, the LML is used in lieu of the SSE.

II.C. Multiscale GPs

We are developing a Multiscale Gaussian process (MGP) method[17] in which the RBF in Eq. (7) is altered to

$$\phi(\mathbf{q}_m, \mathbf{q}_n) = e^{-\frac{||\mathbf{q}_m - \mathbf{q}_n||^2}{h_n^2}} \tag{12}$$

where each training point $\mathbf{q}_n$ is now assigned an individual h_n. The aim is to ensure accuracy in situations where the parameter space contains regions that are sparsely populated and densely populated. For example, if training points are highly clustered in one region of input space, it may be desirable to assign these points smaller values h_n, so that the region is finely resolved. Alternatively, it could be reasonable to choose a single point and give it a larger h_n, so that it can represent the entire region and reduce computational cost by eliminating the need to compute the kernel function at all the other nearby points. Conversely, if a region of the input space has few training points, they could be assigned larger h_n as well to compensate for the sparsity. MGP is useful for the task at hand because the regression is in a high dimensional parameter space which will be differentially populated by data from simulation grids. Hence, the training data is expected to have regions of varying input density. This will be shown to be the case in the next section.

Assigning a different h_n to each training point is a challenging task; MGP streamlines the procedure by reducing the choice of possible h_ns to a finite number, the number of 'scales' K, which may be freely chosen. For example, if $K = 2$, training points will either have one of two possible values for h_n, or have none assigned (to be clarified later).

Given $\mathbf{Q}_{\text{train}}$, h_ns are assigned through the multiscale clustering algorithm below.

1. The hyperparameters h_0 and r are supplied. For the first scale, initialise $k = 1$.

2. For current scale k, $h_k = h_0 r^{k-1}$ is computed. (r is set to be greater than 1, so that h_k decreases with increasing k.)

3. Choose a point from $\mathbf{Q}_{\text{train}}$ at random. Assign h_k to this point, and add the point to the set of cluster centres for the current scale, $\mathbf{Q}_k$. Remove it from original set $\mathbf{Q}_{\text{train}}$.

4. Repeat the previous step, except instead of choosing from $\mathbf{Q}_{\text{train}}$, choose only from points in $\mathbf{Q}_{\text{train}}$ that are at least h_k away from any of the centres in $\mathbf{Q}_k$ in terms of Euclidean distance, i.e. $||\mathbf{q}_m - \mathbf{q}_n|| > h_k$. Continue to repeat until no more points may be chosen, i.e. until every point in $\mathbf{Q}_{\text{train}}$ is within h_k distance of a centre in $\mathbf{Q}_k$.

5. Repeat from Step 2 for the next scale $k = k + 1$, up to K scales in total.

This algorithm may be performed until $\mathbf{Q}_{\text{train}}$ is empty, but an advantage of stopping prematurely at a low value of K is that the new training set, now defined to be the union of the $\mathbf{Q}_k$s and corresponding $\mathbf{Y}_k$s for all k from 1 to K, is much smaller than $\mathbf{Q}_{\text{train}}$. Points that have not been selected as cluster centres and do not have an h assigned after K iterations of the algorithm are not used for training. For large $\mathbf{Q}_{\text{train}}$, this significantly speeds up the regression process.

Once the new training set and the corresponding hs are found, regression may be performed in the same manner as described in Eq. (10). The output noise λ is still random and independent of the covariance between training points, so the LML can still be used to optimise hyperparameters; nothing is changed except the training set and the kernel matrix $\mathbf{\Phi}$. In addition to h_0 and r, the number of scales K is also a hyperparameter for MGP.

II.D. 1-D Example

The goal of this simple numerical example is to demonstrate the capabilities of the methods described above, as well as to offer a graphical explanation of the inner workings of MGP. Here, the output is taken to be a noisy sine function $y = \sin(8\pi q) + \mathcal{N}(0, 0.1)$, with q limited to $[0, 1]$. Two cases are tested: the first with training values for q chosen from a uniform distribution, and the second with q clustered around 0.1, 0.5, and 0.9. 300 training points are used in each case.

Before comparing the different methods, it is useful to explain the MGP algorithm using this example, and thereby show why it is useful. Figures 3 and 4 illustrate the process of selecting cluster centres for the uniform and clustered input cases, with non-optimal hyperparameters. In each figure, the leftmost plot shows the cluster centres chosen with $h_k = h_1$, the middle plot centres with h_2, and the rightmost centres with h_3. Since h_k decreases with k, more centres are chosen with increasing k because they are not constrained to be as far apart from each other. The centres are randomly chosen, so they are not evenly distributed across the domain, but the spacing between them is fairly regular. With these sample parameters, approximately 50 centres cumulative are selected across these three scales, and the rest of the training points are disregarded. This means the size of the training dataset has effectively been reduced from 300 to 50. The consequent increase in speed is very important when coupling a ML model to a RANS solver, since numerous test points must be evaluated per iteration of the flow solver.

Following this graphical example, the optimal hyperparameters for each method are found, and the results of testing on the optimised ML methods are summarised in Table 1. For the clustered case, MGP used only 20 cluster centres out of 300 original training points to achieve this result. Visually, the results do not look very different from each other, as seen in Figures 5 and 6. On a final note, Figure 7 shows that optimal parameters selected through maximising the LML are also near optimal according to the test error, validating the use of LML for hyperparameter optimisation.

Table 1: Normalised test errors for ML methods, defined as $||\mathbf{Y}_{\text{test}} - f(\mathbf{Q}_{\text{test}})||/||\mathbf{Y}_{\text{test}}||$. 1000 uniformly spaced test points on $[0, 1]$ were used. Time is test time in seconds. Training time is part of preprocessing and thus less relevant.

	NN	GP		MGP	
	Error	Error	Time	Error	Time
Uniform input	0.0513	0.0257	9.9×10^{-3}	0.0407	1.8×10^{-3}
Clustered input	0.1185	0.0516	1.2×10^{-2}	0.0716	4.2×10^{-3}

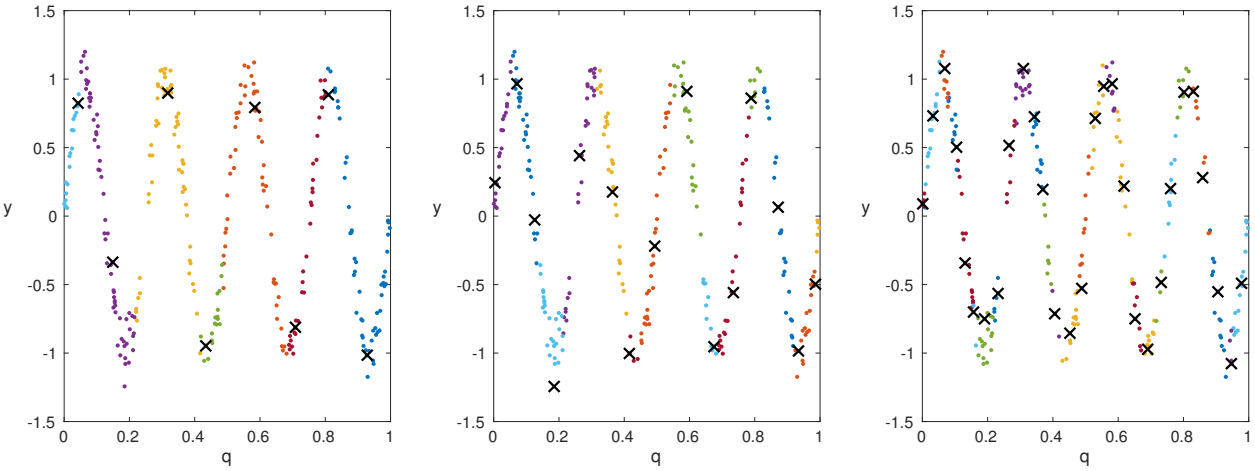


Figure 3: Clustering produced by MGP for uniform distribution of inputs. Left to right: scale $k = 1$, $k = 2$, and $k = 3$. Black crosses are cluster centres selected at each scale. Surrounding each cluster centre is a colour-coded group of training points, signifying points that were within h_k of the cluster centre at the time of its selection. $h_0 = 4$, $r = 2$.

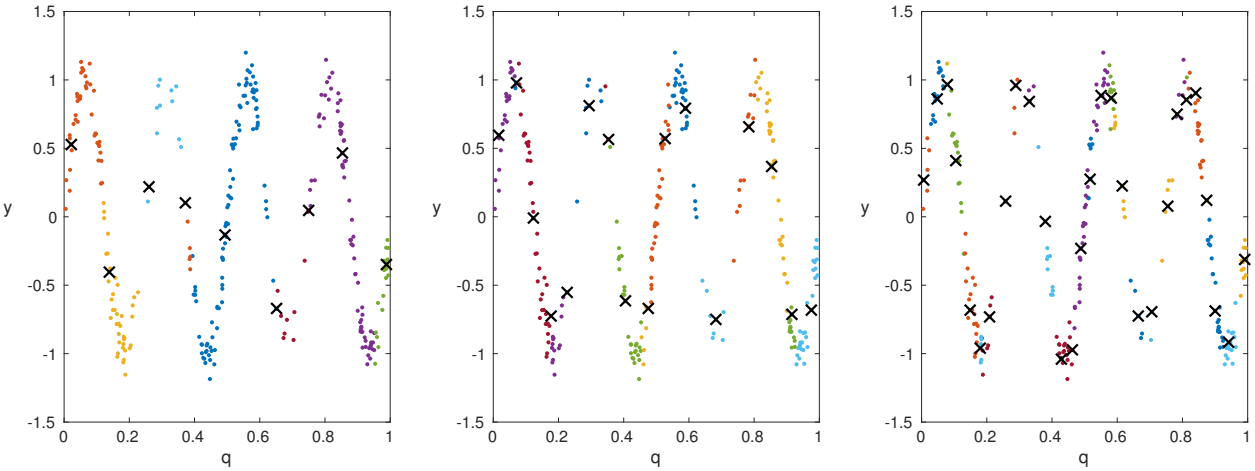


Figure 4: Clustering produced by MGP for clustered distribution of inputs. Left to right: scale $k = 1$, $k = 2$, and $k = 3$. Black crosses are cluster centres selected at each scale. Surrounding each cluster centre is a colour-coded group of training points within h_k of the centre. $h_0 = 4$, $r = 2$ as before.

III. Application to RANS

III.A. Channel Flow

This test case, explained in detail in Ref. 1, involves turbulent channel flow using the Wilcox $k-\omega$ turbulence model, where k and ω are computed through the following transport equations:

$$\frac{\mathrm{D}k}{\mathrm{D}t} = P_k - \beta^*\omega k + \frac{\partial}{\partial x_j}\left[(\nu + \sigma_k \nu_t)\frac{\partial k}{\partial x_j}\right] \tag{13}$$

$$\frac{\mathrm{D}\omega}{\mathrm{D}t} = \gamma\frac{\omega}{k}P_k - \beta\omega^2 + \frac{\partial}{\partial x_j}\left[(\nu + \sigma_\omega \nu_t)\frac{\partial \omega}{\partial x_j}\right] \tag{14}$$

β^*, γ, β, σ_k, and σ_ω are constants, and the production term P_k is

$$P_k = \tau_{ij}\frac{\partial u_i}{\partial x_j} \tag{15}$$

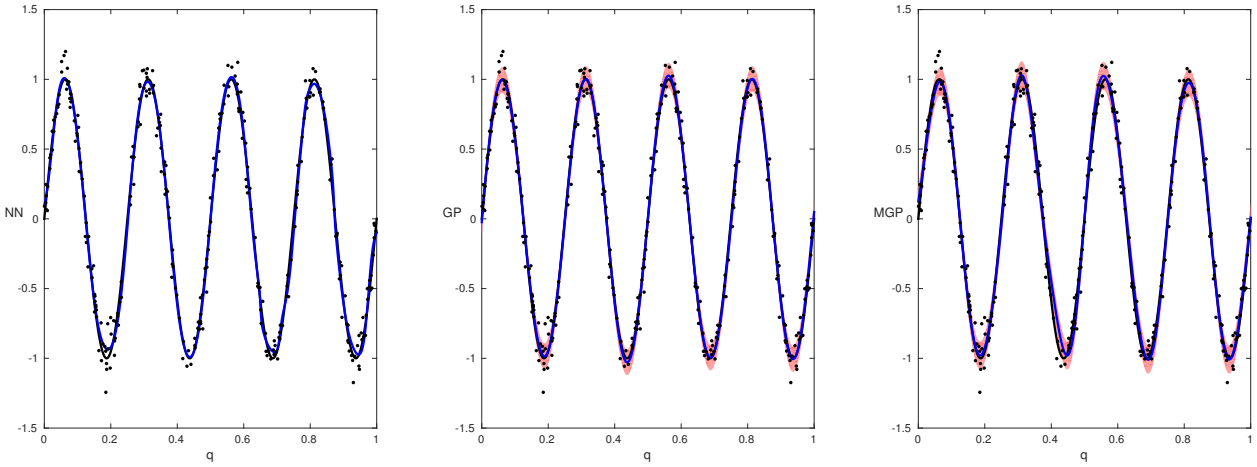


Figure 5: Uniform distribution testing results for all methods. Black line is the original function, black dots are training points with noise added. Blue line is the ML output. For GP and MGP, shaded red regions indicate confidence intervals of $\pm 1\sigma$.

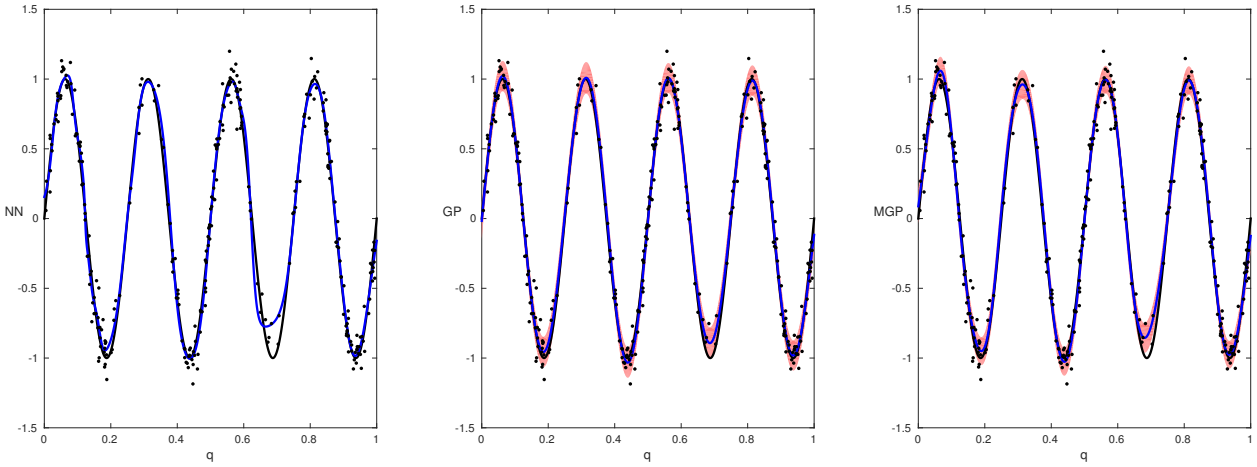


Figure 6: Clustered distribution testing results for all methods. Black line is the original function, black dots are training points. Blue line is the ML output. For GP and MGP, red regions are confidence intervals of $\pm 1\sigma$. There are noticeable divergences from the true function, especially near maxima and minima, compared to the uniformly distributed training point case.

The 1-D channel is a stepping stone from testing ML methods on basic analytical functions. It will be shown that ML works well for this case, and therefore may be extensible to more complex cases.

Recalling the framework from Figure 1, a function α must be introduced. In the case of the 1-D channel, the equation for turbulent kinetic energy, Eq. (13), is modified to be:

$$\frac{\mathrm{D}k}{\mathrm{D}t} = \alpha P_k - \beta^* \omega k + \frac{\partial}{\partial x_j}\left[(\nu + \sigma_k \nu_t)\frac{\partial k}{\partial x_j}\right] \tag{16}$$

α is a function of space and its role is to account for deficiencies in the turbulence model. DNS data is taken as exact, and α is found so that the velocity profile u produced by using αP_k in place of P_k would match the DNS velocity profile.

α is found by solving an inverse problem, for which the method of Bayesian inversion was chosen[10,11] Bayesian inversion produces the maximum a posteriori solution, which is to be reconstructed using machine learning.

Data from five different Reynolds numbers are used: $Re_\tau = 180, 550, 950, 2000, 4200$, each with approximately 150 data points in the wall-normal direction. Only points within the inner and log layers are considered in this exercise. After some deliberation, it was decided that the parameters used for ML, that is, $\mathbf{q}$, would consist of Sk/ϵ, $d\sqrt{k}/\nu$,

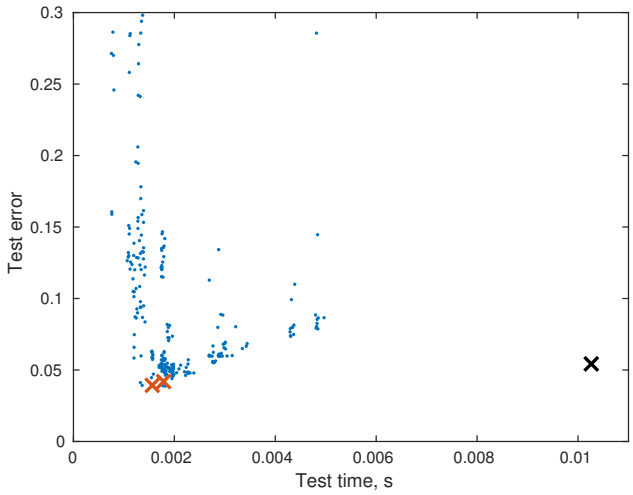


Figure 7: Tradeoffs in speed and accuracy for MGP. Blue are for equally-spaced hyperparameters, orange are optimal parameters from maximising LML for different values of number of scales K. Black is GP for comparison. The error rises very sharply as time decreases.

P_k/ϵ, and y^+. d is defined as the normal coordinate divided by channel half-width, y/h, and P_k is the production terms in the $k-\omega$ equations. The first three parameters are completely local, while y^+ depends on wall shear.

In Figure 8, α is plotted as a function of these parameters. For the most part, α is multi-valued near the wall and single-valued near the centreline. Also, as is typical, there are more data points near the wall than far. Both multi-valuedness and sparsity of data can present difficulties for ML.

Instead of choosing training and testing points randomly as in the analytical example in the previous section, the $Re_\tau = 2000$ case was set aside for testing, and the other four cases were used for training. The tradeoff between accuracy and speed for MGP is depicted in Figure 9, where once again the optimal hyperparameters yield results near the pareto front. Note that GP is also plotted in the same figure, and appears to be a better choice if speed is not a concern.

The test results are plotted in Figures 10 and 12, and errors listed in Table 2. The near-wall estimates are accurate but have high variance. This variance is close to the estimated input variance λ, which is higher for MGP than for GP because the former uses fewer points and hence can be less certain of what lies between them. Both figures suggest that the learning is less accurate near the centreline, especially for MGP. This suggests that the sparsity of training points hinders it severely, which seems counterintuitive because MGP is supposed to take precisely this into account. Figure 11 helps shed some light on this. As evidenced by the speed of the method, MGP summarises the available training points with a handful of cluster centres, chosen near-randomly. In the figure, it can be seen that, for the first two scales, fewer centres lie near the centreline. Conceptually, if the first centre is chosen near the wall, the next will be by default further from the wall *despite* the sparsity, because taking the first centre near the wall excludes all the other near-wall points from being the next centre. However, as seen in Figure 8 and 11, the training points are distributed so that some points far from the wall can be excluded as well. Hence, for the first few scales, points that are far from the wall are poorly represented. As h_k decreases, more of them are selected as centres, but they are unable to accurately represent the region near the centreline because h_k is small, but the variation in α is larger. To mitigate this effect, one may choose a small h_0. However, for higher values of k, h_k may grow too small to be effective. This method requires fine tuning of the scaling factor for h_k, r, but the example is simple enough that GP should prove to be superior in any case.

Table 2: Normalised test errors and testing times for ML methods. Times are in seconds.

	NN	GP		MGP	
Test case	Error	Error	Time	Error	Time
Re 2000	0.0215	0.0161	5.6×10^{-3}	0.0431	4.3×10^{-3}

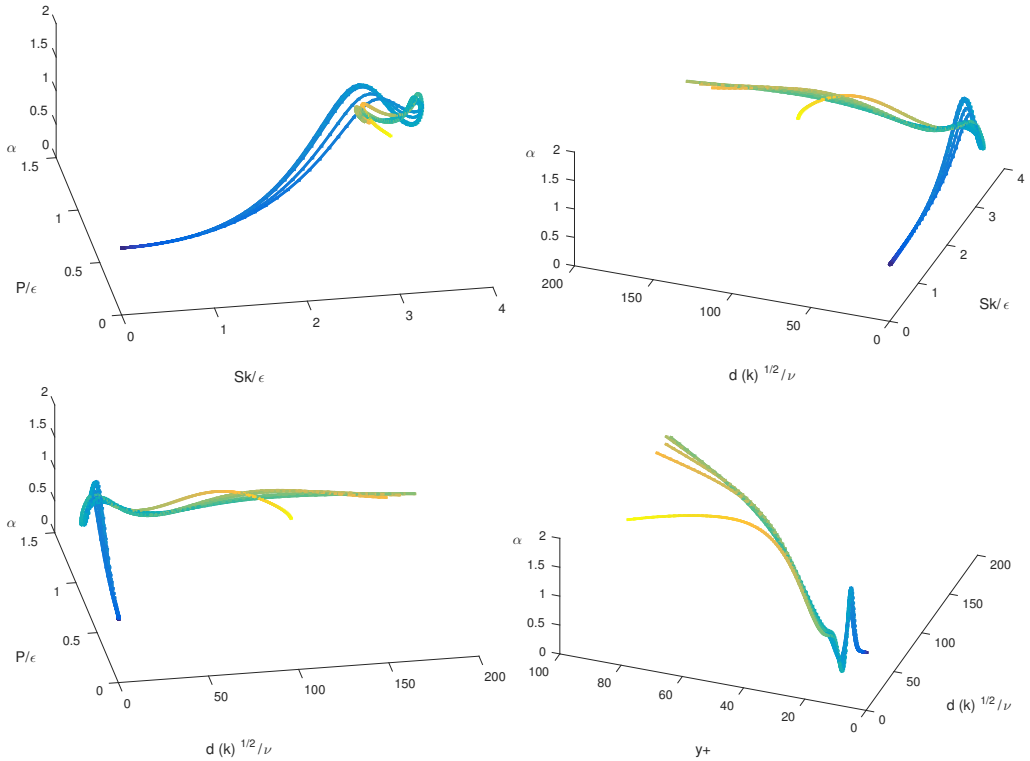


Figure 8: Visualisation of training data, all five cases. Note that the lines, each representing a different case, are easily distinguishable from one another due to the sparsity of data. Blue indicates data points near the wall, yellow data points near the centreline.

III.B. Bypass transition

Bypass transition occurs when free-stream turbulence introduces instabilities in the evolving boundary layer to become turbulent. A class of modelling approaches use the concept of an intermittency factor, γ, to represent the fraction of time during which turbulence is active. It is useful as a way of representing and modelling the phenomenon, but γ is not a property that can be directly found from a DNS solution.

Ge et al.[9] use the intuitive idea that γ diffuses into the boundary layer from the turbulent free-stream and transition occurs via molecular and turbulent diffusion. To this end, they write a transport equation for γ:

$$\frac{\mathrm{D}\gamma}{\mathrm{D}t} = P_\gamma - D_\gamma + \frac{\partial}{\partial x_j}\left[(\sigma_l\nu + \sigma_\gamma\nu_t)\frac{\partial\gamma}{\partial x_j}\right] \tag{17}$$

The source term P_γ contributes to producing intermittency inside the boundary layer. The sink term D_γ ensures that the boundary layer initially is laminar. Definitions of these terms can be found in Ref. 9.

A method that leverages the transition model would be to introduce an α that modifies Eq. (17) as follows:

$$\frac{\mathrm{D}\gamma}{\mathrm{D}t} = \alpha + \frac{\partial}{\partial x_j}\left[(\sigma_l\nu + \sigma_\gamma\nu_t)\frac{\partial\gamma}{\partial x_j}\right] \tag{18}$$

In other words, α is used to replace the production and destruction terms entirely. Experimental wall shear stress data from the T3 series[19] is taken to be the truth and α is found through Bayesian inversion. Further details about the problem setup can be found in Ref. 2.

The transitional flow cases, denoted T3A, T3B, and T3C1 through T3C5, are 2-D flat plates with different free stream turbulence intensities and pressure gradients. Each case has approximately 2000 to 8000 usable data points. Only points near the wall and with a vorticity magnitude Ω higher than a threshold value were used. The training parameters used for learning are $d^2\Omega/\nu$, γ, du/dx, du/dy, k, and ω. Figure 13 plots the training points for Case T3B in terms of these parameters. α is high in some regions, but near zero everywhere else. The contours of α in later figures show that this is also the case for T3A and the T3C series.

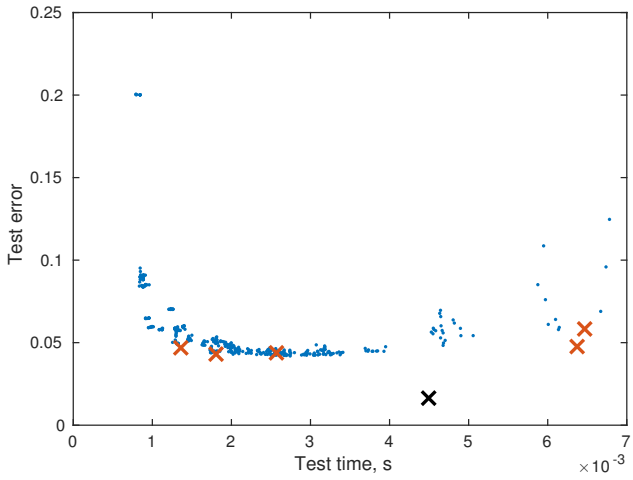


Figure 9: Tradeoffs in speed and accuracy for MGP. Blue are for equally-spaced hyperparameters, orange are optimal parameters from maximising LML for different values of number of scales K. Black is GP for comparison.

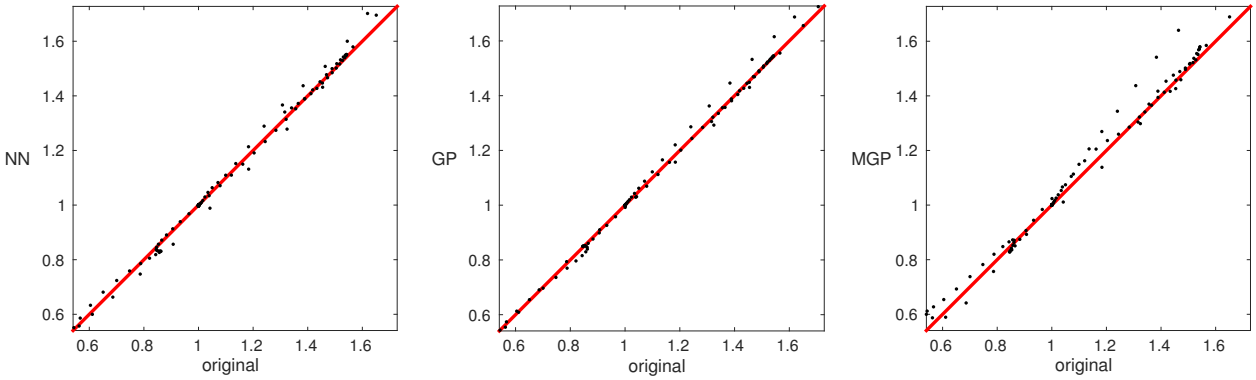


Figure 10: Test results for α, Re $= 2000$.

III.B.1. Training results

Numerical results are presented in Table 3. To obtain these, all of the training points doubled as test points, hence the errors being labelled as training errors. If a method trains well, it is likely to also have low test error, since the training space is more densely populated than in the case of the 1-D channel.

GP and MGP both outperformed NNs, with MGP being the more accurate of the two. This is a trend seen in Figures 15 through 24 as well. This is because GP and MGP take the training points themselves into account when testing, whereas NNs compound the training information into a predefined number of coefficients, depending on the number of layers and nodes. Therefore, when testing on the training points, GP and MGP have an advantage.

It is interesting to note that, while errors are low, testing times are high for MGP. This is likely due to some overhead in the program used to produce these results, since at worst MGP uses as many of the training points as GP does, and in most cases it uses fewer. Furthermore, Figure 14 shows that for Case T3B, the set of MGP hyperparameters used was not optimal, should the minimisation of the test time have been the goal. It can also be seen from the figure that the GP result does not lie on the pareto front. This is likely the case for all the transitional cases examined in this section.

III.B.2. Test results

Previously, it was assumed that a low training error implies a potentially low test error. Here, that assumption is tested using cases T3A, which is a flat plate, and T3C1, which has a pressure gradient. For each case, 200 randomly selected points were set aside as test points. The rest, around 4000 points for case T3A and around 8000 for case T3C1, were used for cross-validation. For each case, these points were divided into 24 sets, and optimal hyperparameters were found using 23 of the sets together as training and the remaining set for validation, where validation error is

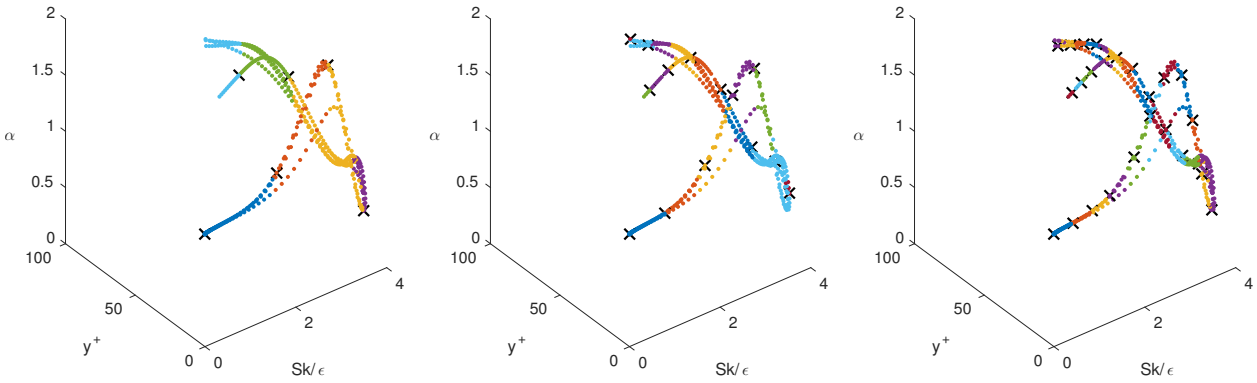


Figure 11: MGP clusters for the first three scales; $k = 1$ at left, $k = 2$ at centre, $k = 3$ at right.

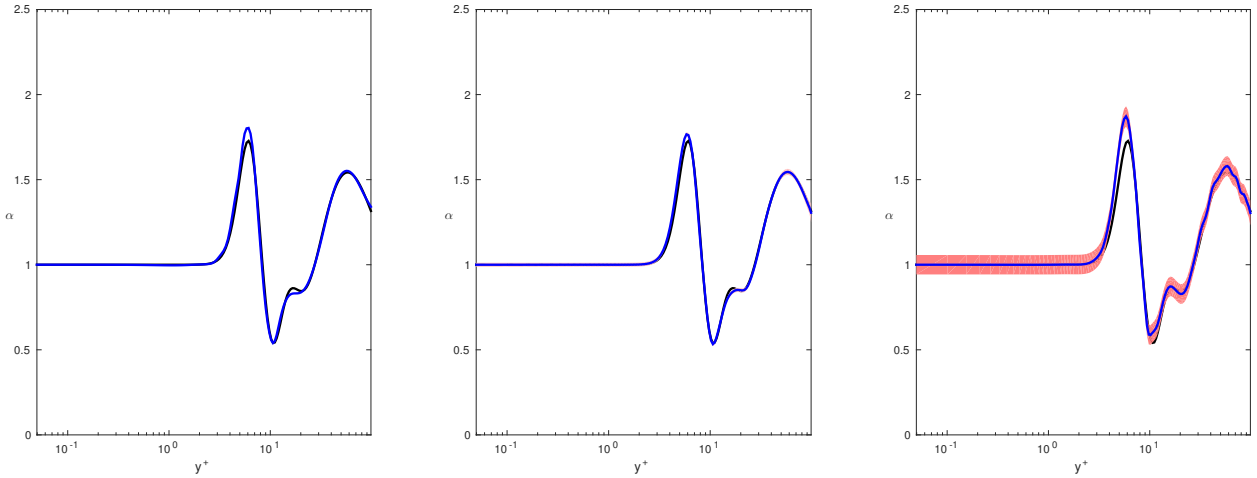


Figure 12: Test results for $Re_\tau = 2000$ as a function of y^+. The black line represents the actual α, blue is the ML mean, and red regions represent $\pm 1\sigma$. σ is very small for GP, and is only faintly visible.

the quantity that is minimised. This yields 24 sets of optimal hyperparameters, and the median of these was used for testing. Also, all training and validation points were used for computing the test results.

Table 4 lists the test errors, while Figures 25 and 26 plot the test points against the ML estimates. The test errors for T3A are much higher than those for T3C1, since the former had fewer training points. Examining Figure 25, it is likely that the high errors for case T3A are due to outliers. For example, at around $\alpha = 35$, MGP has captured a test point, whereas for NN and GP, this point lies outside the range of the graph. It is also worth noting that the NN and GP have errors of about ± 10, approximately the same spread pictured in Figure 15. In contrast, the spread of the MGP results is much greater than before. This suggests that MGP was overfitting the training data in the previous section. Comparing Figures 19 and 26 also supports this for case T3C1.

Further optimization and tuning is required to mitigate the propensity of the multi-scale GP method to overfit the training data.

IV. Conclusion

This paper focused on the machine learning aspect of a larger framework for data-driven turbulence modeling. A new multiscale Gaussian process regression method was presented and its performance was assessed with reference to conventional Gaussian process (GP) regression and a neural network (NN) algorithm. Two problems were examined; one that aims to reconstruct a multiplicative correction function arising from inversion applied to turbulent channel flow and the other deals with an corrective source term arising from inversion applied to bypass transition. All the machine learning methods are shown to be capable of reconstructing the spatial form of the correction terms into functional forms operating on high dimensional feature spaces. Such a capability is immensely valuable in data driven closure modeling in general. Though these tests cannot be considered to be representative of all turbulence modeling

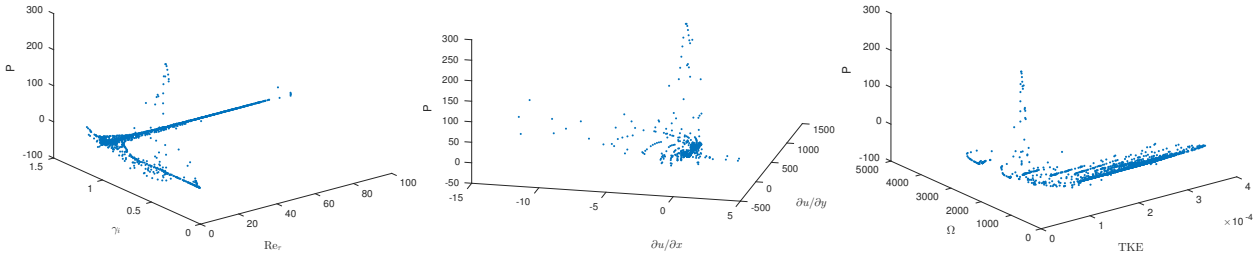


Figure 13: Training points for Case T3B, plotted in terms of the inputs. Every other point is shown.

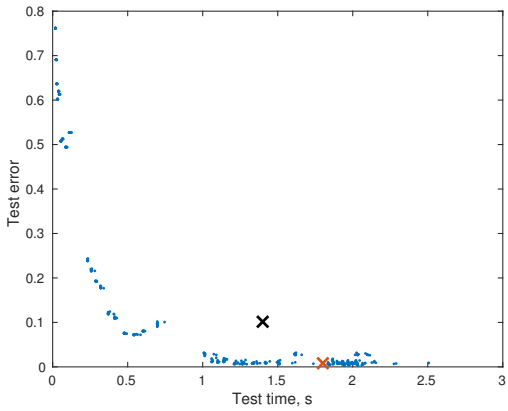


Figure 14: Speed vs. accuracy for Case T3B. Blue are MGP results using evenly spaced variations of σ^2 and h_0 hyperparameters. r and K were kept constant. Orange is chosen set of hyperparameters. Black is GP for comparison.

needs – and there are alternate formulations of the NN and GP algorithms – the flexibility of the multiscale method in representing the sparsely populated feature space is evident and its increased efficiency over the GP method suggests its viability as a candidate for functional reconstruction in closure modeling applications.

Acknowledgments

This work was funded by the Leading Edge Aeronautics Research for NASA grant titled "A Framework for Turbulence Modeling Using Big Data." The authors thank Prof. Juan Alonso and Brendan Tracey (Stanford University) for their inputs. The multiscale Gaussian Process regression is being developed as part of an on-going collaboration with Dr. Nail Gumerov and Prof. Ramani Duraiswami (University of Maryland).

Table 3: Normalised training errors and testing times. Times are in seconds.

	NN	GP		MGP	
Case	Error	Error	Time	Error	Time
T3A	0.5547	0.1719	1.1	0.0198	1.4
T3B	0.2116	0.1010	1.4	0.0081	1.8
T3C1	0.1163	0.0490	6.1	0.0092	10.8
T3C2	0.2351	0.0931	4.7	0.0403	5.3
T3C3	0.5311	0.1511	4.8	0.0725	3.8
T3C4	0.3352	0.1262	6.7	0.0867	5.2
T3C5	0.1956	0.0490	6.0	0.0274	10.0

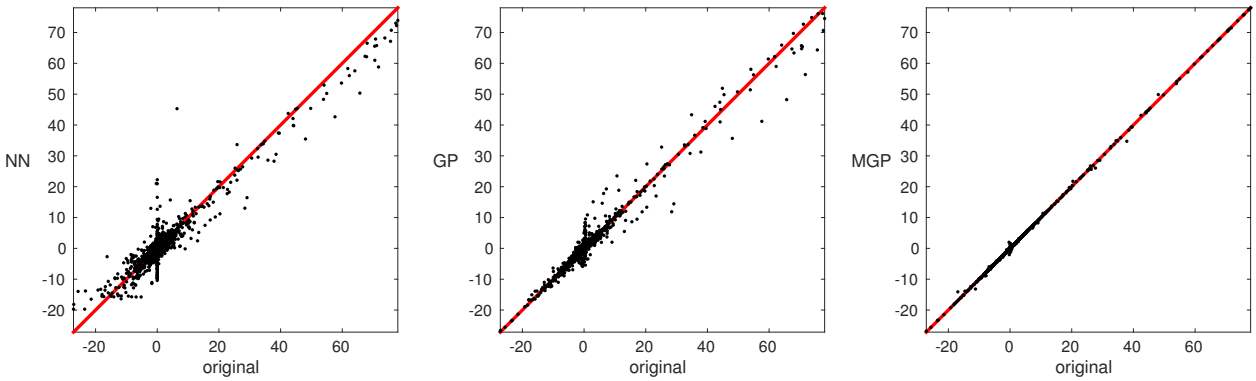


Figure 15: ML test results for the difference between source and sink terms $P_\gamma - D_\gamma$, i.e. α, for Case T3A.

Table 4: Normalised test errors and testing times. Times are in seconds.

	NN	GP		MGP	
Case	Error	Error	Time	Error	Time
T3A	1.4127	8.2442	0.076	0.5324	0.13
T3C1	0.2487	0.7474	0.20	0.1580	0.73

References

[1]Parish, E. and Duraisamy, K., "Quantification of Turbulence Modeling Uncertainties Using Full Field Inversion," *15th AIAA Aviation Technology, Integration, and Operations Conference,*, Dallas, TX, June 2015.

[2]Duraisamy, K., Zhang, Z., and Singh, A., "New Approaches in Turbulence and Transition Modeling Using Data-driven Techniques," *AIAA Modeling and Simulation Technologies Conference,*, Kissimmee, FL, January 2015.

[3]Tracey, B., Duraisamy, K. & Alonso, J., "A Machine Learning Strategy to Assist Turbulence Model Development," *Proc. AIAA Scitech conference*, Orlando, Florida, 2015.

[4]Spalart, P. R. and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," *30th Aerospace Sciences Meeting & Exhibit,*, Reno, NV, Jan. 1992.

[5]Yarlanki, S., Rajendran, B., and Hamann, H., "Estimation of turbulence closure coefficients for data centers using machine learning algorithms," *13th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, May 2012.

[6]Milano, M. and Koumoutsakos, P., "Neural network modeling for near wall turbulent flow," *Journal of Computational Physics*, Vol. 182, No. 1, 2002, pp. 1-26.

[7]Sarghini, F., de Felice, G., and Santini, S., "Neural networks based subgrid scale modeling in large eddy simulations," *Computers & Fluids*, Vol. 32, No. 1, 2003, pp. 97-108.

[8]Wilcox, D., *Turbulence modeling for CFD*, Vol. 2, DCW industries, 2006.

[9]Ge, X., Arolla, S., and Durbin, P., "A bypass transition model based on the intermittency function," *Flow, Turbulence and Combustion*, Vol. 93, No. 1, 2014, pp. 3761.

[10]Aster, R. C., Borchers, B., and Thurber, C. H., *Parameter estimation and inverse problems,* Academic Press, 2013.

[11]Isaac, T., Petra, N., Stadler, G., and Ghattas, O., "Scalable and efficient algorithms for the propagation of uncertainty from data through inference to prediction for large-scale problems, with application to flow of the Antarctic ice sheet," *arXiv preprint,* arXiv:1410.1221, 2014.

[12]Bishop, C. M., *Pattern Recognition and Machine Learning,* Springer, 2006.

[13]Kohavi, R. and John, G. H., "Wrappers for Feature Subset Selection," *Artificial Intelligence,* Vol. 97, No. 1, 1997, pp. 273-324.

[14]Nissen, S., "Fast Artificial Neural Network Library - FANN."

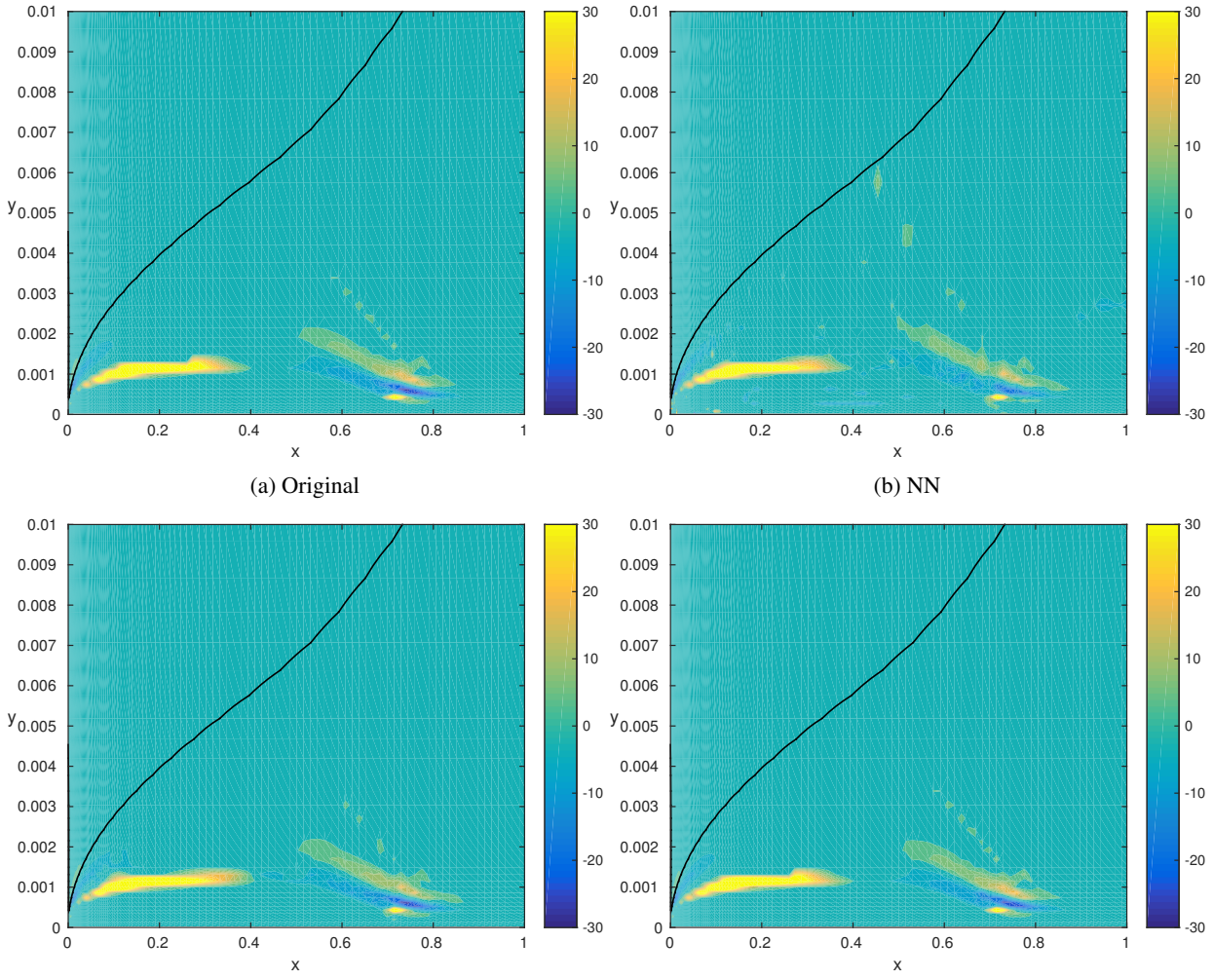
[15]Park, J. and Sandberg, I. W., "Universal Approximation Using Radial-Basis-Function Networks," *Neural Computation,* Vol. 3, No. 2, 1991, pp.246-257.

[16]Rasmussen, C. E. and Williams, C. K. I., *Gaussian Processes for Machine Learning,* MIT Press, 2006.

[17]Zhang, Z., Gumerov, N. A. and Duraisamy, K., "High-dimensional Multiscale Gaussian Process Regression," *Manuscript under preparation* 2015.

[18]Walder, C., Kim, K. I., and Schlkopf, B., "Sparse multiscale Gaussian process regression," *Proceedings of the 25th international conference on Machine learning*, July 2008.

[19]Roach, P. E. & Brierley, D. H., "The influence of a turbulent free-stream on zero pressure gradient transitional boundary layer development Part 1: Test cases T3A and T3B," *Numerical Simulation of Unsteady Flows and Transition to Turbulence*, pp. 319–347, ERCOFTAC, 1992.



(a) Original

(b) NN

(c) GP

(d) MGP

Figure 16: Contours of α for Case T3A.

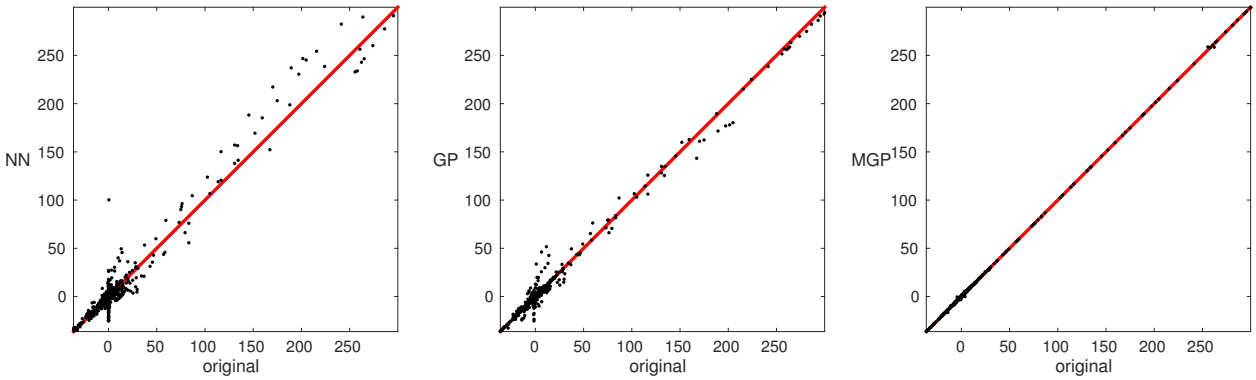
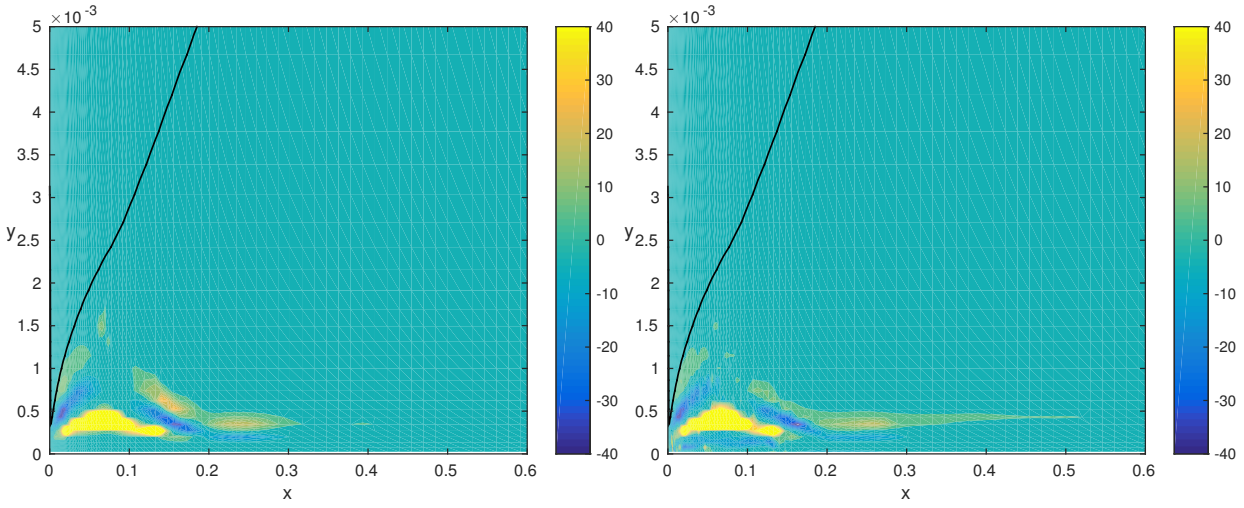
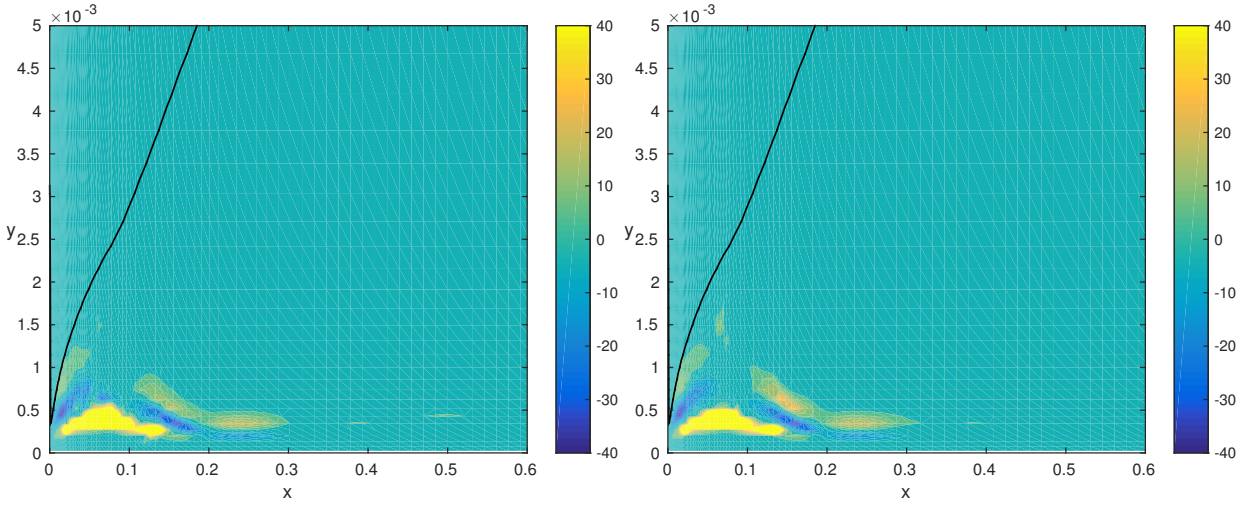


Figure 17: Case T3B test results.



(a) Original

(b) NN



(c) GP

(d) MGP

Figure 18: Contours for Case T3B.

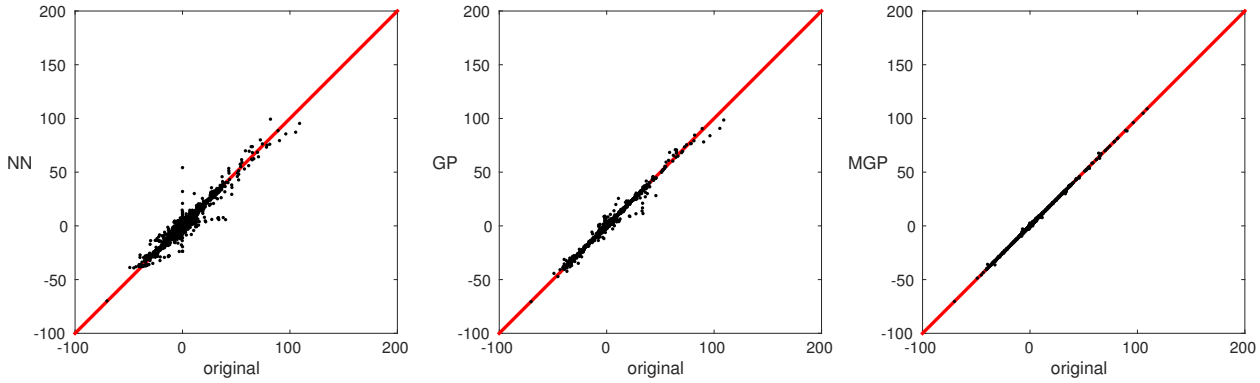
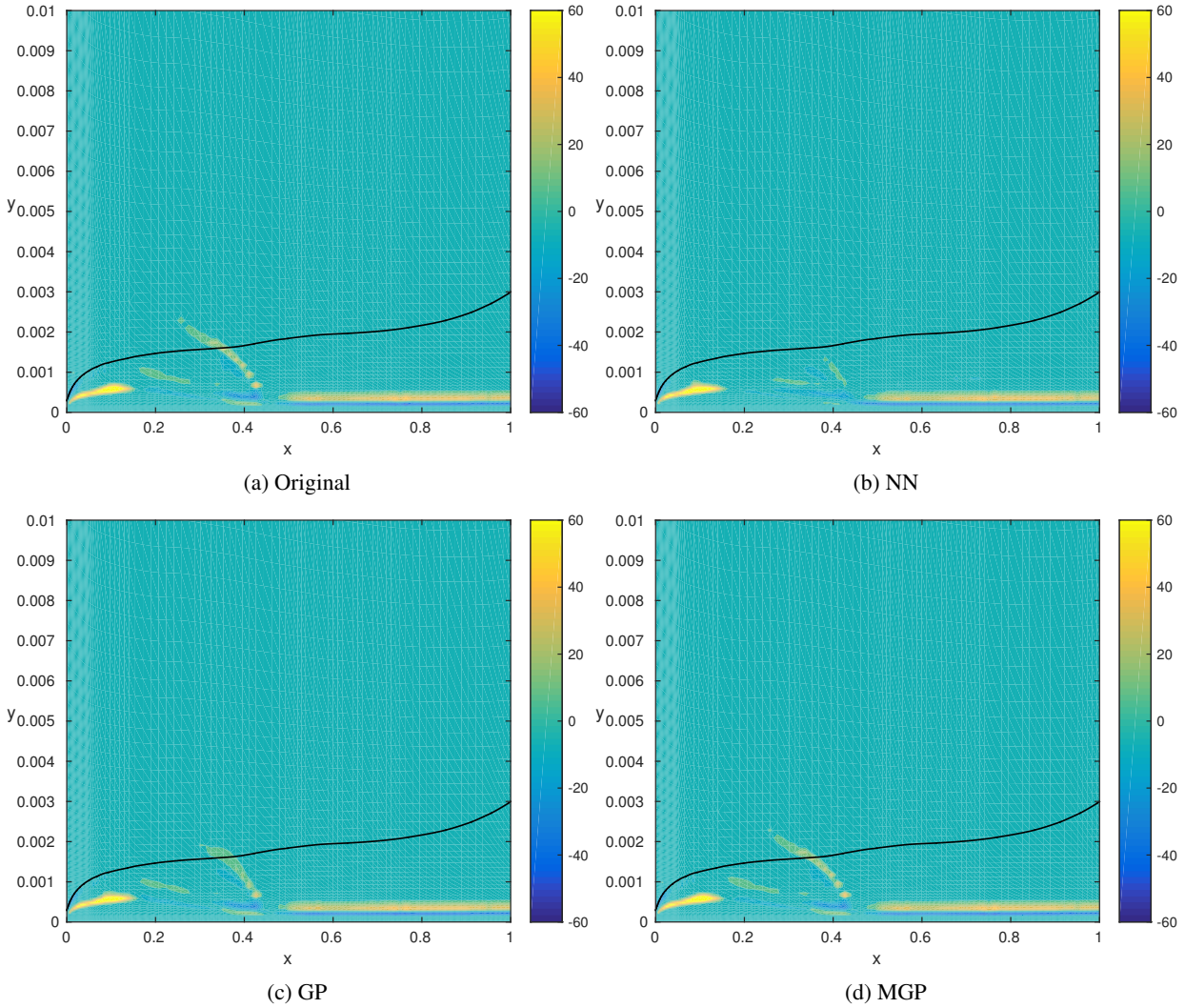


Figure 19: T3C1 test results.



(a) Original (b) NN

(c) GP (d) MGP

Figure 20: Contours for Case T3C1.

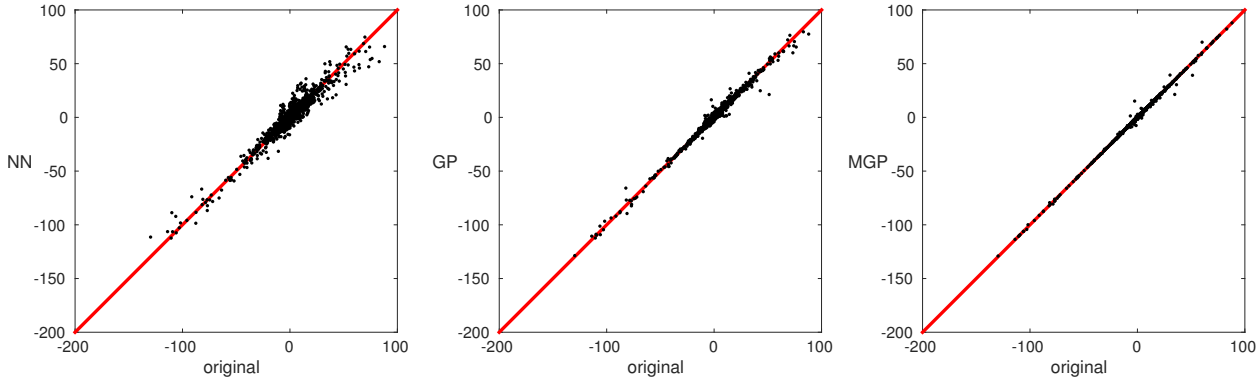


Figure 21: Test results for Case T3C2.

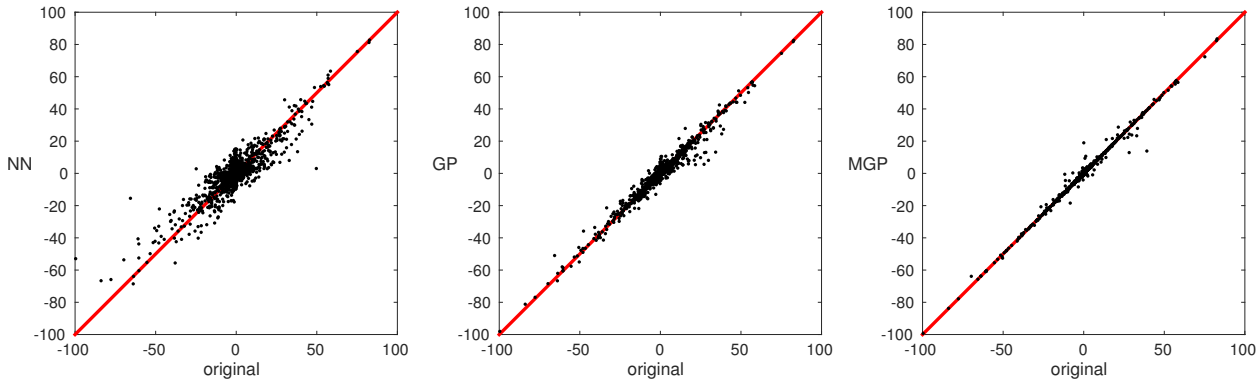


Figure 22: Test results for Case T3C3.

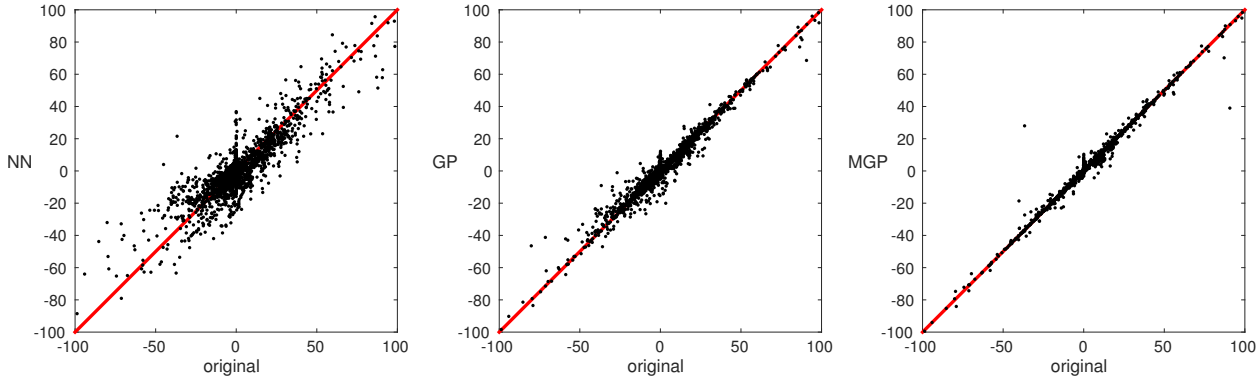


Figure 23: Test results for Case T3C4.

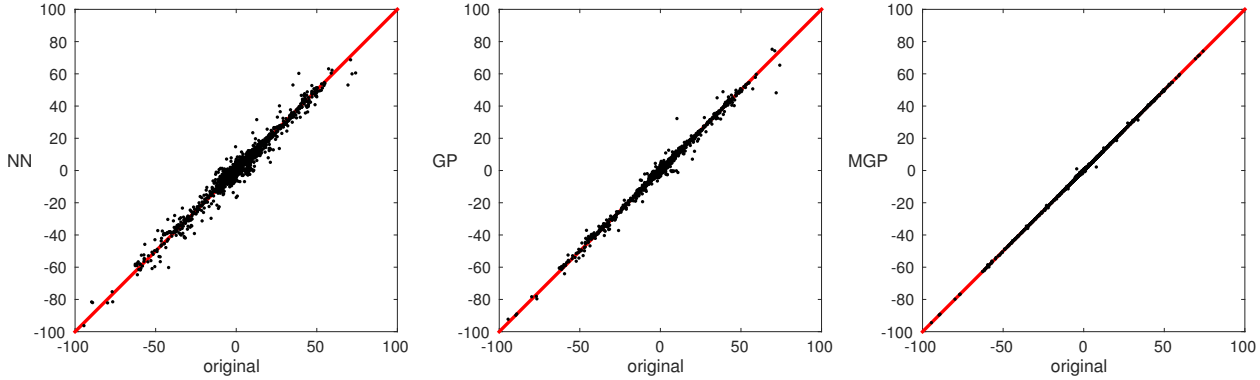


Figure 24: Test results for Case T3C5.

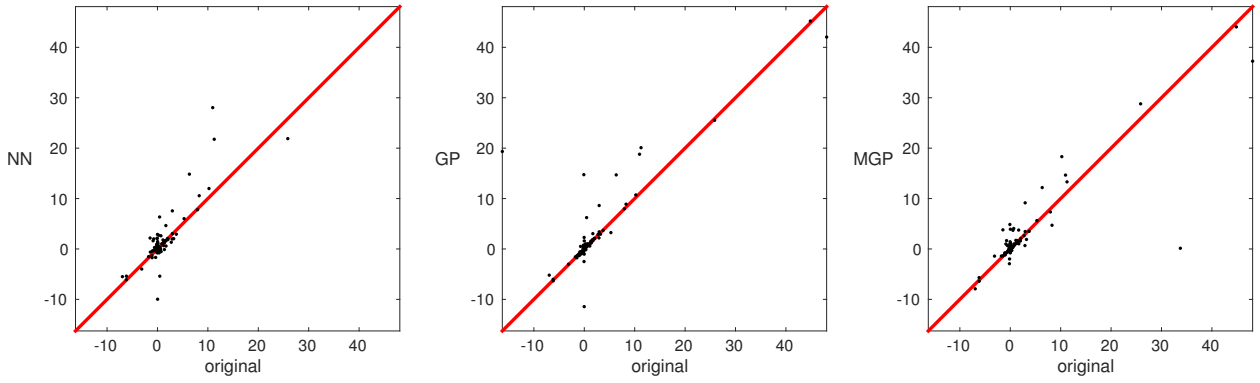


Figure 25: Case T3A results for 200 randomly selected test points not used in training.

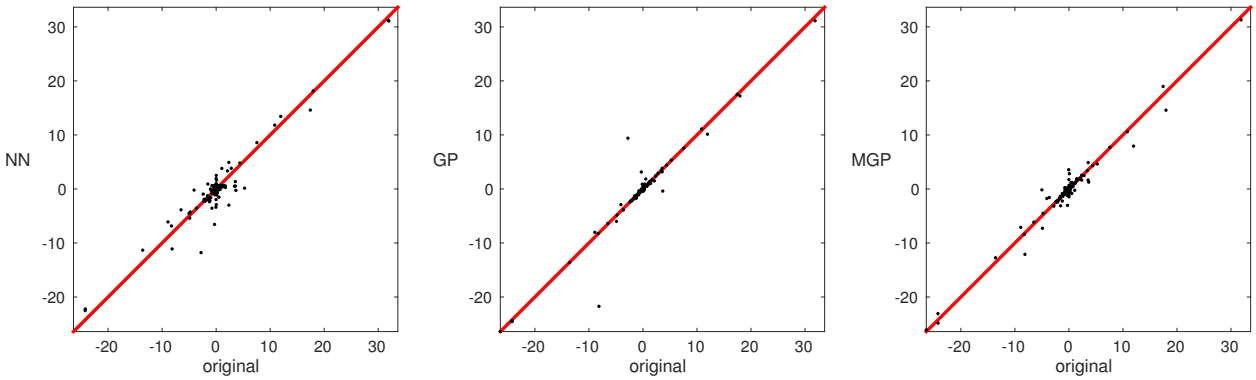


Figure 26: Case T3C1 results for 200 test points not used in training.