Development of a Computational Fluid Dynamics Emulator using Surrogate Modelling Techniques

Magnus Jackson
ENGM010 Data-Centric Engineering
University of Exeter

I. Introduction

Computational Fluid Dynamics (CFD) is a continuously evolving research field that allows for the determination of flow patterns, forces, and many more aspects of complex fluid flow problems. As the field is developed further through both commercial and open source codes, an increasing problem complexity can be analysed to give highly accurate results. However, as the problem domain grows, or additional flow features are incorporated, the computational demand grows significantly. This means that simulations either require a large computational resource, at a high cost, or a reduction in the accuracy or scope of the problem to be evaluated [1].

The advances in understanding of Gaussian processes has led to the development of new surrogate modelling methods, which allows for the construction of emulators. An emulator is a statistical model that uses probabilistic theory and numerical simulation data to attempt to learn a specific, unknown black box function that maps the input to the output [1] [2]. This could also be described as a regression problem, fitting multiple dimensions to a set of training data. As a result of this, emulators can be effectively employed to investigate a wide range of input values, based on a comparatively small number of data points.

CFD is a research area that is well applied to surrogate model methods due to its non-linear characteristics and the high computational cost associated with complex problems [1]. With CFD, emulators can be constructed with just a handful of inputs and outputs, which is particularly efficient when just CFD is being used to determine one or two single value outputs, such as drag or lift. Emulators can also be constructed for vector field outputs, where it will take the independent variable of interest and the associated vector field as inputs, which allows for a wider investigation and understanding of the independent variable. However, using a high number of dimensions in training can result in a complex and time consuming process, therefore when evaluating vector fields, principal component analysis (PCA) is used to reduce the dimensions of the input [2] [3].

Emulators have been effectively applied to a number of numerical problems, including examples such as the non linear stochastic mechanics of a 3D MEMS device [3], full fields stress analysis of a connecting rod [2], thermal block analysis in 2D [4], and on fluid examples such as the Von Kalman vortex street and the Saint Venant equations [4]. This demonstrates the wide range of applications the surrogate modelling approach is capable of handling.

As a relatively new research area, the methods of implementation vary significantly. There are 'traditional' approaches that use the reduced basis coefficients to generate a map between raw input to output fields [2]. This method was shown to be effective for predicting accurate values, with the added advantage of having a measure of confidence in the prediction. It was also shown to be effective for both linear and non linear systems, with its efficacy increasing with the DOFs and non linearity as the computational cost of solving these more complex systems grows massively. There have also been recently proposed active learning methods, which takes an initial coarse sample to build the model and subsequently predicts values at new parameter locations which is backed up by a separate data driven surrogate [4]. This methods allows for the construction of a high fidelity model that supports a parametric system, which again has the advantage of reducing the computational cost of a problem.

II. Project Overview

This project attempts to implement surrogate modelling techniques to develop an emulator for the backwards facing step flow problem. This is a widely used bench marking test in CFD for the investigation of separation flows, seen in applications such as airfoils, engine flows, and flows around buildings [5]. This in turn means that there is a good base understanding of the problem to compare emulator results to the widely understood characteristics of a backwards facing step flow problem.

The emulator attempts to predict the recirculation length as a ratio of the step height in response to changes in the inlet velocity and the turbulent kinetic energy intensity at the inlet. The emulator used in this project utilises Gaussian processes to achieve these predictions, which is governed by the mean, μ, and by the covariance matrix, Σ. To define these values, a kernel is used which controls how the input data is read and handled by the model [6]. This project uses the RBF kernel which is a very commonly used kernel and has the advantage of being easily applied to a range of problems.

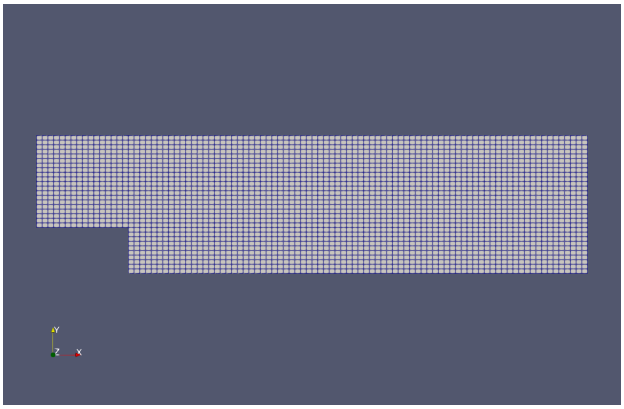


Fig. 1: Computational Mesh

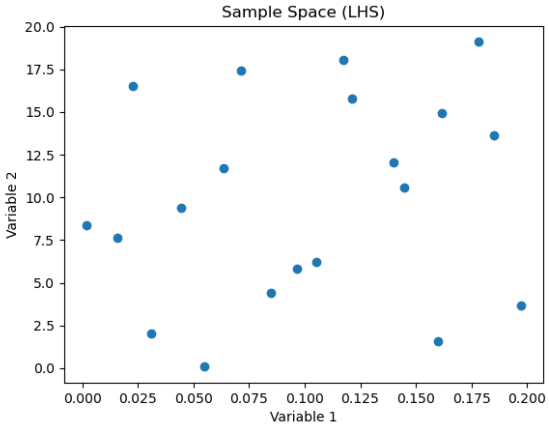


Fig. 2: Latin Hypercube Sampling Example

III. Computational Methodology

As mentioned, this paper will investigate the recirculation length of the backwards facing step, in response to changes in inlet velocity and turbulent kinetic energy intensity. In addition to the reasons discussed for the choice of this case from a validation point of view, the simplicity of the case set up will allow for the quick generation of training data. The final reason for this choice is that a mesh independence study has been completed for this geometry, and as the focus of this paper is on the construction of the emulator, it reduces time spent on the computational set up [7]. Both of these reasons remove the focus from the data gathering, and allow more effort to be put into the surrogate modelling methods that form the basis of this paper.

Based on the mesh independence study, a 2D mesh of 2720 cells was used for the simulations, shown in Fig. 1. This used only hexahedral type cells with a fixed aspect ratio of 1.25, and no skewness issues.

For the boundary conditions, wall type was used for the top and bottom faces including the step, the sides used the empty type and the inlet and outlet faces were patches with initial conditions assigned to them. The initial conditions were kept the same for everything except for the inlet velocity and turbulent kinetic energy intensity. The inlet velocity was varied between $0ms^{-1}$ and $20ms^{-1}$, and the turbulent kinetic energy intensity was varied between $0.001ms^{-1}$ and $0.2ms^{-1}$.

To determine the length of the recirculation zone, the 'plot over line' function in Paraview was used. Here, the velocity in the x-direction was plotted with the y value being tuned to be just above the boundary layer cells. Using this plot, the length of the recirculation zone could be taken as the point that the x velocity switched from negative to positive. Once the recirculation zone was measured, it was divided by the step height (0.2m), as is commonplace in investigations into the backwards facing step. With all data collected, this represented the raw data to be used for the training of the emulator.

IV. Surrogate Modelling Theory & Methodology

A. Sampling

For this paper, Latin Hypercube Sampling (LHS) was used and was implemented via the Surrogate Modelling Toolbox (SMT) in Python [8]. This toolbox was used as it allows for a quick and simple way to implement LHS without spending unnecessary time writing a new algorithm.

LHS is a semi random sampling method that divides the sample space into n intervals in n dimensions, and within each section created, a central position is assigned. The algorithm uses five main criteria to generate samples which essentially attempts to place one sample within each division of each dimension with a focus on maximising the minimum distance between points while ensuring no dimension has more than one value in each division. SMT added the fifth criteria to the framework, which uses an optimisation algorithm to enhance the results [8]. This method could also be described in more simple terms as a pseudo-sudoku problem. An example of the sample space LHS can generate is shown in Fig. 2. Careful examination of this plot shows that each sample does not have a repeated value in either dimension.

B. Emulator Theory & Design

The emulator was constructed using surrogate modelling techniques that employ Gaussian processes to generate a black box function, mapping input to output. This black box function is a multivariate Gaussian distribution over vectors of two random variables X and Y, and the shape is governed by the mean vector μ and the covariance matrix Σ, as shown in (1) [6] .

$$P(X,Y) = \begin{bmatrix} X \\ Y \end{bmatrix} \backsim \mathcal{N}(\mu, \Sigma) \tag{1}$$

Therefore, the emulator requires a way to covert the raw input to output into two sets of values, μ and Σ. This is done through the use of a kernel. In order to simplify the calculations, it is often assumed that $\mu = 0$, even if $\mu \neq 0$ [6].

In this situation, μ is added back to the distribution after the prediction step. This process is known as centering the data.

This means that the focus is on the covariance matrix, which is determined via the kernel, k, or the covariance function. There are many types of kernels which can each be used for separate tasks and applications, with two main classes being stationary or non-stationary [9]. For this paper, the RBF kernel will be used (2) as it is a versatile and well documented function.

$$k(t, t') = \sigma^2 \exp\left(-\frac{\|t - t'\|^2}{2l^2}\right) \tag{2}$$

This function takes t and t' as inputs and produces a similarity measure of the two points. As a result, the entry Σ_{ij} describes the influence that the i-th and j-th point have on each other, which in turn controls the possible shapes the fitted function can take. For the RBF kernel, the performance is controlled by the variance, σ, and the length scale, l, which are known as the hyperparameters. The variance determines the average distance away from the functions mean [6], while the length scale controls the reach of influence on neighbors which affects the fitting of the model to the data. Varying the length scale is the main control for under-fitting and over-fitting of the training data.

With the kernel set, the regression task can be implemented. Firstly, the raw data requires some manipulation before it can be effectively used for the training process. The input data requires normalisation to the interval $\begin{bmatrix}0 & 1\end{bmatrix}$ which is calculated via (3), and the output data requires standardisation to have zero mean and unit variance, calculated via (4) [1]. As the data only has 2 input dimensions and 1 output dimension, there is no need to decompose the data any further.

$$S[:, j] = \frac{S_{raw}[:, j] - s_{min,j}}{s_{max,j} - s_{min,j}} \tag{3}$$

$$Y[i, :] = \frac{Y_{raw}[i, :] - y_{avg,i}}{y_{std,i}} \tag{4}$$

The posterior distribution, $P(X|Y)$, can now be obtained by observing the modified training data. The first step is to form the joint probability distribution, $P_{X,Y}$, giving a multivariate Gaussian distribution with dimensions equal to that of the test points and of the training data. Through the process of conditioning, the posterior $P(X|Y)$ can be found from $P_{X,Y}$ which produces derived versions of the mean and standard deviation [6]. This results in a conditional distribution which forces the set of function to pass through the training points. Marginalisation is then used to extract the mean function value and standard deviation from each i-th point. This gives the final distribution to map the input to output.

The model was then improved using an acquisition function to generate the next sample point to evaluate the design space at. For this paper, an arbitrary aim was taken to minimise the recirculation length, although this method could easily be modified to maximise a value too. This function aims to select a point that will yield a good quality solution and explore areas where the uncertainty is high [10]. This was used to select the initial conditions for 4 more training points to improve the model, following the initial training using the 20 data points from LHS.

To implement this process in Python, the toolbox `sciKitLearn` was used [11]. This was used as it easily implements the kernel for Gaussian regression problems, and has the capability to tune the hyperparameters for optimal performance. In addition to this, the kernel optimisation and subsequent training can be easily controlled through keyword arguments.

V. Results

A. Training Data Results

Using LHS, an initial set of 20 points was generated for simulation initial conditions. The recirculation length and associated step height ratio was found, with each member of the group doing 5 simulations each. An example of the velocity magnitude plot is shown in Fig. 3. The raw data initially gathered is shown in Table I.

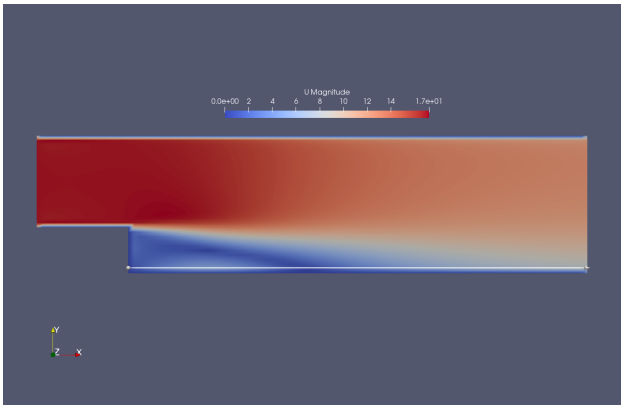


Fig. 3: Velocity Magnitude Plot

TABLE I: Raw Data

Sample	k	U ms^{-1}	Recirc. Length (m)	Ratio of H
1	0.1283970	16.92780	0.3815	1.9075
2	0.0571957	1.84931	0.4165	2.0825
3	0.1637660	3.72207	0.3595	1.7975
4	0.1767490	12.51520	0.3525	1.7625
5	0.0794834	9.15597	0.4206	2.103
6	0.0933685	4.1109	0.4043	2.02134
7	0.0120645	8.79247	0.4061	2.03035
8	0.0631101	10.67490	0.4092	2.045915
9	0.1548780	2.79451	0.4048	2.024165
10	0.1024480	0.15698	0.3702	1.85084
11	0.0013705	5.03365	0.4077	2.038455
12	0.0302337	11.55950	0.4074	2.037135
13	0.1971570	7.37160	0.4062	2.03079
14	0.1109730	15.61010	0.4060	2.030245
15	0.0877914	13.78100	0.4085	2.04263
16	0.1829830	19.36090	0.447	2.235
17	0.1403020	14.24820	0.439	2.195
18	0.0468786	18.21260	0.471	2.355
19	0.1379370	17.49050	0.448	2.24
20	0.0379453	6.11915	0.428	2.14

B. *Emulator Results*

The initial data set from Table I was used to produce Fig. 4, 5, and 6. The colour scale had to be limited in Fig. 4 due to high values in the corners, losing resolution in the middle where the main investigation was focused. Following this, the optimisation steps were carried out, generating the new points shown in Table II, with results shown in Fig. 7, 8, and 9.

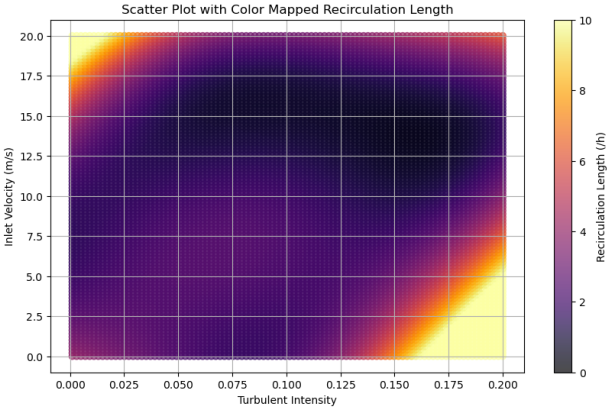


Fig. 4: Initial Plot of Recirculation Length

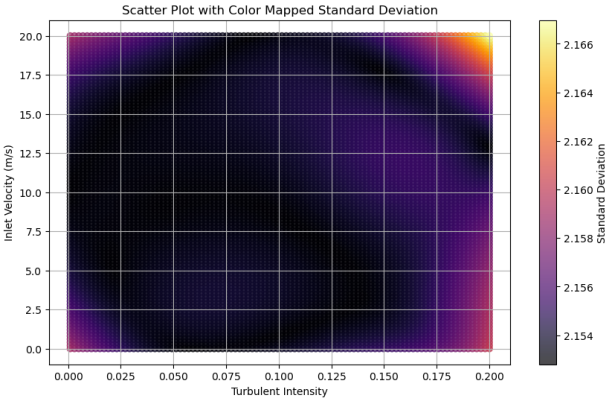


Fig. 5: Initial Plot of Standard Deviation

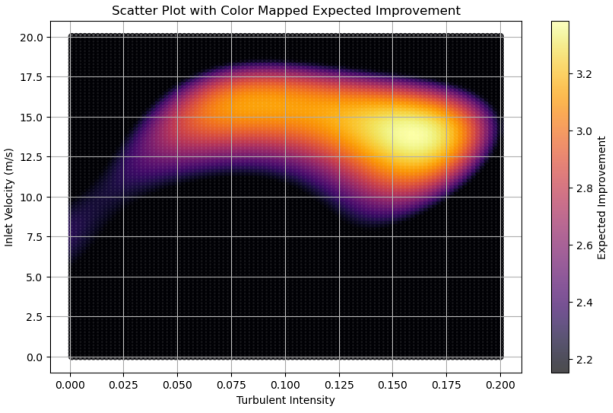
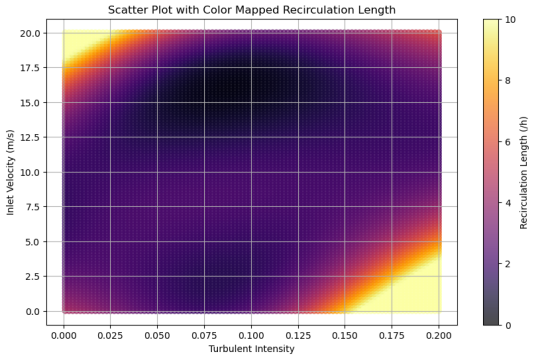
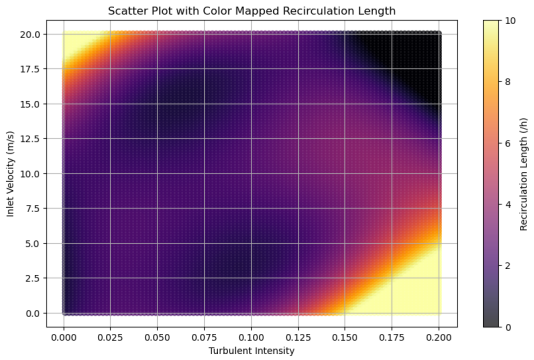


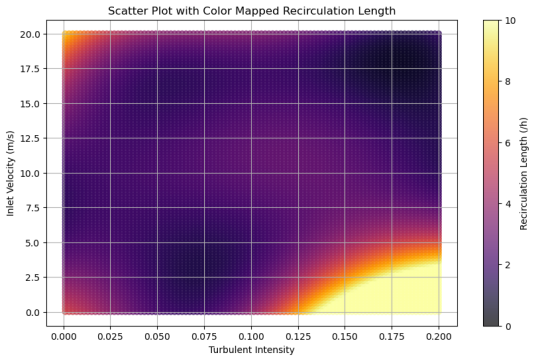
Fig. 6: Initial Plot of Expected Improvement



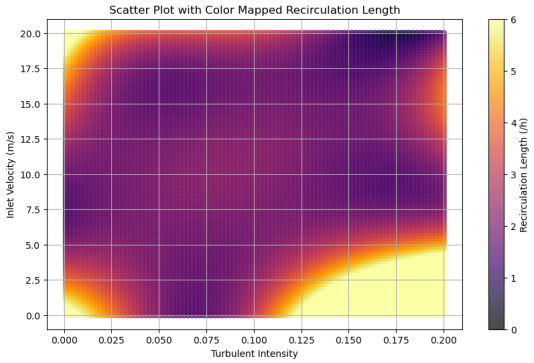
(a) Optimisation 1



(b) Optimisation 2

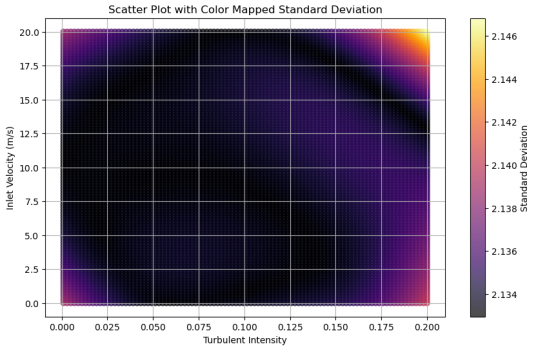


(c) Optimisation 3

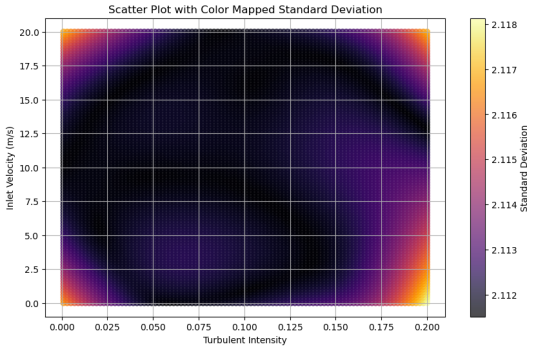


(d) Optimisation 4

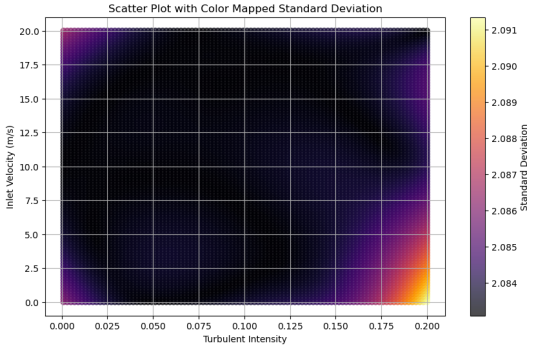
Fig. 7: Recirculation Length Optimisation



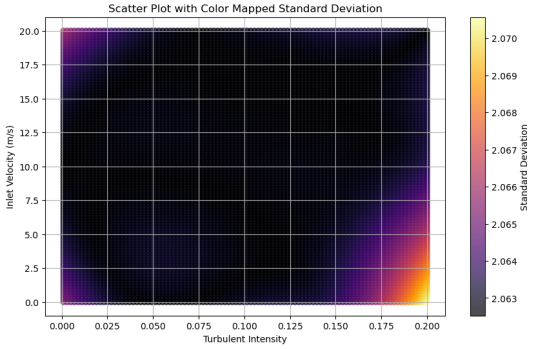
(a) Optimisation 1



(b) Optimisation 2

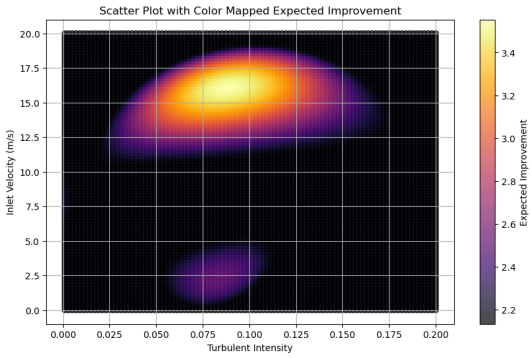


(c) Optimisation 3

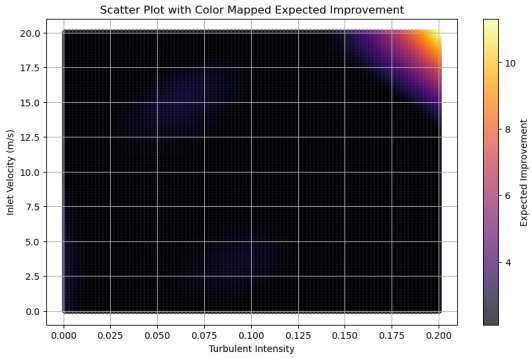


(d) Optimisation 4

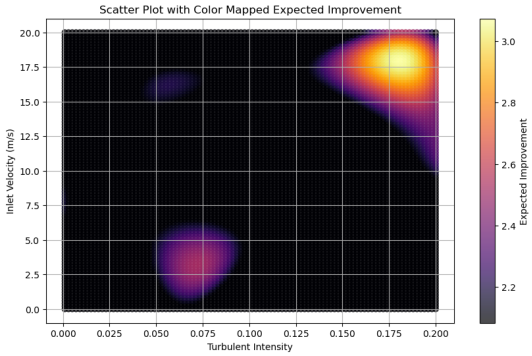
Fig. 8: Standard Deviation Optimisation



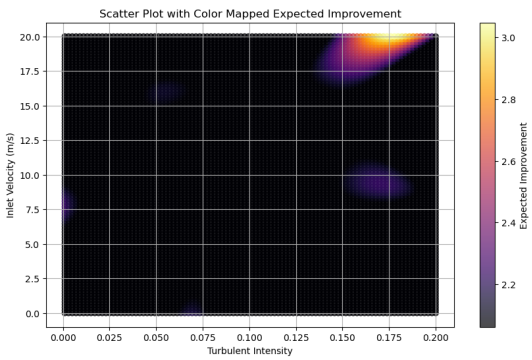
(a) Optimisation 1



(b) Optimisation 2



(c) Optimisation 3



(d) Optimisation 4

Fig. 9: Expected Improvement Regions

TABLE II: Optimisation Steps

Sample	k	U (ms^{-1})	Recirc. Length (m)	Ratio of H
O1	0.139697	15.959596	0.375	1.875
O2	0.161808	8.8888889	0.36227	1.81135
O3	0.200000	20.000000	0.33133	1.65665
O4	0.179899	17.979798	0.3467	1.7335

VI. Discussion

From the plots of recirculation length in Fig. 4 and 7, it can be seen that the emulator was able to predict a range of values that mostly lie within the accepted values for the backwards facing step [5]. The emulator was also able to demonstrate its ability to quantify the uncertainty in predictions through plots of standard deviation. In addition to this, the use of the acquisition function was successful at improving the quality of the overall model, shown through improvements in the recirculation length plot and in the reduction of the standard deviation.

From the plot of recirculation length resulting from the first 20 samples, shown in Fig. 4, it can be seen that the emulator predicts a mostly linear 'ridge' of longer recirculation zones as the turbulent kinetic energy intensity and inlet velocity increase together. This falls in line with what would be expected, as the free stream velocity will drag the recirculation bubble out further. However, it could be argued that an increase in turbulent kinetic energy would result in the shear layer mixing faster with the recirculation bubble, resulting in a shorter reattachment length. This figure also has several parts that generate some questions and interest. Firstly, there is a darker (shorter) patch in the top half of the plot, and secondly, two of the corners have hit the colour map limit. The dark area is likely the model returning to the prior mean of 0, and this is corroborated by the higher standard deviation in this area shown in Fig. 5. The expected improvement picks up on both of these metrics well, showing a clear region and peak for the next point to be evaluated in Fig. 6.

The optimisation steps shown in Fig. 7 demonstrate how the model changes with the addition of new and efficiently placed samples. The first optimisation step in Fig. 7a shows a shrinkage of the dark patch and introduces changes along the bottom edge of the plot. This change does not carry over to the standard deviation plot which remains very similar in Fig. 8a. However, examining the values on the colour bar, it can be seen that with each optimisation step, there is a reduction in the standard deviation values. With the 3rd and 4th optimisation steps, the standard deviation is lowered significantly across the whole plot, bar the bottom right corners. The high values seen in the bottom right corners of Fig. 7 represents a limitation of this implementation, where the true values are between 20-50. This shows the model does not have an understanding of the physical limits of the data it is representing, and in regions where the function carries on along the same gradient it left the last training point at. For this reason, it may be advisable to evaluate problems on a larger range than needed, and subsequently using just the inner portions of the models range where it is unaffected by a lack of samples near the edges. The lack of physical understanding may be remedied by the inclusion of prior knowledge which would likely reduce these issues, however this went beyond the scope of this project hence its exclusion.

From Fig. 9, it can easily be seen that the acquisition function was successful at choosing new spots to evaluate where the previous iteration had a low recirculation length as the model returned to a mean of 0 and where there was a higher standard deviation. In the context of CFD, where simulations are very time and energy intensive, the addition of this function represents a step in the efficiency of building an emulator when compared to purely stochastic sampling.

This model suffered due to the choice of the inlet velocity range, which meant that the majority of the sample space was operating in conditions that lend themselves to a stable recirculation area. This meant the model was attempting to predict very small changes in the wider sample space. To improve this project, instead of using inlet velocity as an input, a more appropriate choice would be the Reynolds number at the inlet. This would make the whole problem physically dimensionless and mean the flow characteristics could be easily controlled.

VII. Conclusion

This paper has effectively reviewed and implemented surrogate modelling methods to build an emulator for the CFD analysis of the backwards facing step flow problem. This utilised state-of-the-art methods implemented via the toolbox `sciKitLearn` to map the input to output. There were also learnings into the efficient implementation of these methods, which were mainly around the generation and handling of raw data to ensure the model has the best chance of predicting correct and accurate results. This paper also demonstrated the power of active optimisation via use of acquisition functions, showing a decrease in the standard deviation of the model and an improvement in the overall model.

This project was also a successful group project, with all members contributing equally to the project. For the literature review and data gathering, the workload was split evenly and each member completed their work in a timely manner and to a high standard. For the implementation, the group all brought different parts to the table, resulting in a well applied end result.

References

[1] P. Moonen and J. Allegrini, "Employing statistical model emulation as a surrogate for CFD," *Environmental Modelling & Software*, vol. 72, pp. 77–91, Oct. 2015, ISSN: 13648152. DOI: 10.1016/j.envsoft.2015.06.007. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1364815215300025 (visited on 02/14/2024).

[2] C. Ding, H. Rappel, and T. Dodwell, "Full-field order-reduced gaussian process emulators for nonlinear probabilistic mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 405, p. 115 855, Feb. 2023, ISSN: 00457825. DOI: 10.1016/j.cma.2022.115855. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0045782522008118 (visited on 02/14/2024).

[3] C. Soize and R. Ghanem, "Probabilistic-learning-based stochastic surrogate model from small incomplete datasets for nonlinear dynamical systems," *Computer Methods in Applied Mechanics and Engineering*, vol. 418, p. 116 498, Jan. 2024, ISSN: 00457825. DOI: 10.1016/j.cma.2023.116498. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0045782523006229 (visited on 02/14/2024).

[4] H. Kapadia, L. Feng, and P. Benner, "Active-learning-driven surrogate modeling for efficient simulation of parametric nonlinear systems," *Computer Methods in Applied Mechanics and Engineering*, vol. 419, p. 116 657, Feb. 2024, ISSN: 00457825. DOI: 10.1016/j.cma.2023.116657. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0045782523007806 (visited on 02/14/2024).

[5] L. Chen, K. Asai, T. Nonomura, G. Xi, and T. Liu, "A review of backward-facing step (BFS) flow mechanisms, heat transfer and control," *Thermal Science and Engineering Progress*, vol. 6, pp. 194–216, Jun. 2018, ISSN: 24519049. DOI: 10.1016/j.tsep.2018.04.004. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2451904918300167 (visited on 04/02/2024).

[6] J. Görtler, R. Kehlbeck, and O. Deussen, "A visual exploration of gaussian processes," *Distill*, vol. 4, no. 4, 10.23915/distill.00017, Apr. 2, 2019, ISSN: 2476-0757. DOI: 10.23915/distill.00017. [Online]. Available: https://distill.pub/2019/visual-exploration-gaussian-processes (visited on 03/23/2024).

[7] M. Jackson, *ECMM148 advanced CFD - tutorial 3*, 2023.

[8] P. Saves, R. Lafage, N. Bartoli, *et al.*, "SMT 2.0: A surrogate modeling toolbox with a focus on hierarchical and mixed variables gaussian processes," *Advances in Engineering Software*, vol. 188, p. 103 571, Feb. 2024, ISSN: 09659978. DOI: 10.1016/j.advengsoft.2023.103571. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S096599782300162X (visited on 04/02/2024).

[9] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning* (Adaptive computation and machine learning). Cambridge, Mass: MIT Press, 2006, 248 pp., OCLC: ocm61285753, ISBN: 978-0-262-18253-9.

[10] G. De Ath, J. E. Fieldsend, and R. M. Everson, "What do you mean?: The role of the mean function in bayesian optimisation," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, Cancún Mexico: ACM, Jul. 8, 2020, pp. 1623–1631, ISBN: 978-1-4503-7127-8. DOI: 10.1145/3377929.3398118. [Online]. Available: https://dl.acm.org/doi/10.1145/3377929.3398118 (visited on 03/25/2024).

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html.

Appendix

Code Implementation

```python
from scipy.stats import qmc
import matplotlib.pyplot as plt

#initialise the sampler
sampler = qmc.LatinHypercube(d=2) #set number of dimensions
sample = sampler.random(n=20) #set sample size and generate the inital sample

#define the first variable upper and lower bounds
var1low = 0.001
var1upper = 0.2

#define the second variable upper and lower bounds
var2low = 0
var2upper = 20

#format into combined upper and lower bounds for the sampler to read
lowerBound = [var1low, var2low]
upperBound = [var1upper, var2upper]

#scale the sample ranged from 0 - 1, into the desired scaling
scaledSample = qmc.scale(sample, lowerBound, upperBound)

#generate a plot of the sample
plt.plot(scaledSample[:,0], scaledSample[:,1],'o')
plt.xlabel('Variable 1')
plt.ylabel('Variable 2')
plt.title('Sample Space (LHS)')
plt.show()
```

Listing 1: Latin Hypercube Sampler Code

```python
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from matplotlib import pyplot as plt

from sklearn.gaussian_process import GaussianProcessRegressor
```

```
from sklearn.gaussian_process.kernels import RBF
from scipy.stats import norm

#normalise the input data
input_data_raw = np.array([[0.1283970, 16.92780],
                           [0.0571957, 1.84931],
                           [0.1637660, 3.72207],
                           [0.1767490, 12.51520],
                           [0.0794834, 9.15597],
                           [0.0933685,  4.11090],
                           [0.0120645,  8.79247],
                           [0.0631101, 10.67490],
                           [0.1548780,  2.79451],
                           [0.1024480,  0.15698],
                           [0.0013705, 5.03365],
                           [0.0302337, 11.55950],
                           [0.1971570,  7.37160],
                           #optimisation points - comment out to see original graphs
                           [0.1396969696969697, 15.959595959595958],
                           [0.161808081, 8.888888889],
                           [0.2, 20.0],
                           [0.1798989898989899, 17.97979797979798]
                           ])

#initialise the normaliser
normalizer = Normalizer()
scaler_input = MinMaxScaler()

input_data = scaler_input.fit_transform(input_data_raw)

#standardise the output data
output_value_raw = np.array([1.9075,
                             2.0825,
                             1.7975,
                             1.7625,
                             2.103,
                             2.025,
                             2.03,
                             2.035,
                             2.02,
                             1.845,
                             2.13822,
                             3.12001,
                             3.120128,
                             #optimisation points - comment out to see original graphs
                             1.875,
                             1.81135,
                             1.65665,
                             1.7335
                             ])

#reshape the output data
output_value_raw = output_value_raw.reshape(-1, 1)
scaler_output = StandardScaler() #initialise the standardiser
output_value = scaler_output.fit_transform(output_value_raw)

#normalise the range to be evaluated over
k_raw = np.linspace(0.001, 0.2, 100)
# Normalize the array
min_val = np.min(k_raw)
max_val = np.max(k_raw)
k = (k_raw - min_val) / (max_val - min_val)

inlet_speed_raw = np.linspace(0,20,100)
# Normalize the array
min_val1 = np.min(inlet_speed_raw)
max_val1 = np.max(inlet_speed_raw)
inlet_speed = (inlet_speed_raw - min_val1) / (max_val1 - min_val1)


#used to turn the two input variables into a single array
combinations = [(x, y) for x in k for y in inlet_speed]
all_points = np.array(combinations)

#initialise the Gaussian Process model
kernel = RBF()
# kernel = ConstantKernel(0.1, constant_value_bounds=(1e-5, 1e2)) * RBF(0.015, length_scale_bounds=(1e-5, 1e2))
gp_model = GaussianProcessRegressor(normalize_y=True, n_restarts_optimizer=50)

#train gp model using initial training data
gp_model.fit(input_data, output_value)

# Generate predictions and expected improvement for all possible points using the Gaussian process model

def expected_improvement(x, gp_model, best_y, epsilon):
    y_pred, y_std = gp_model.predict(x, return_std=True)
    print('The standard deviation: ',y_std)
    z = (best_y - y_pred - epsilon)/y_std
    ei = ((best_y - y_pred - epsilon) * norm.cdf(z)) + y_std*norm.pdf(z)
    return ei, y_pred, y_std

best_idx = np.argmin(output_value)
best_y = output_value[best_idx]

print('The minimum value for the recirculation length is:',best_y)

#all_points is a 2d array of all combinations of wave height and boat speed that you want to look at
ei, y_pred, y_std = expected_improvement(all_points, gp_model, best_y, 0.01)

print(y_std.shape)

print(y_std)

print(ei.shape)
print('The best expected improvement is: ', ei[np.argmax(ei)], 'at co-ordinates', all_points[np.argmax(ei),0],',', all_points[np.argmax(ei),1])
print(np.min(y_pred))

#revert normalised and standardised data for plotting

combinations = [(x, y) for x in k_raw for y in inlet_speed_raw]
all_points_standard = np.array(combinations)

# Reshape y_pred, y_std, and ei to have shape (n_samples, 1)
y_pred_reshaped = y_pred.reshape(-1, 1)
y_std_reshaped = y_std.reshape(-1, 1)
ei_reshaped = ei.reshape(-1, 1)

# Revert the scaled output back to its original scale
reverted_y_pred = scaler_output.inverse_transform(
```

```
    y_pred_reshaped)
reverted_y_std = scaler_output.inverse_transform(
    y_std_reshaped)
reverted_ei = scaler_output.inverse_transform(
    ei_reshaped)

best_idx = np.argmin(reverted_y_pred)
best_y_pred = reverted_y_pred[best_idx]
print('Lowest y pred', best_y_pred)

#plot the prediction and standard deviation
rX, rY = np.meshgrid(k_raw, inlet_speed_raw)

plt.figure(figsize=(10, 6))
#sc = plt.scatter(wave_height, boat_speed, c=y_pred,
     cmap='viridis', alpha=0.7)
sc = plt.scatter(rX, rY, c=reverted_y_pred, cmap='
    inferno', alpha=0.7)
plt.colorbar(sc, label='Recirculation Length (/h)')
plt.clim(0, 5)
plt.xlabel('Turbulent Intensity')
plt.ylabel('Inlet Velocity (m/s)')
plt.title('Scatter Plot with Color Mapped
    Recirculation Length')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
sc = plt.scatter(rX, rY, c=reverted_y_std, cmap='
    inferno', alpha=0.7)
plt.colorbar(sc, label='Standard Deviation')
plt.xlabel('Turbulent Intensity')
plt.ylabel('Inlet Velocity (m/s)')
plt.title('Scatter Plot with Color Mapped Standard
    Deviation')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
sc = plt.scatter(rX, rY, c=reverted_ei, cmap='
    inferno', alpha=0.7)
plt.colorbar(sc, label='Expected Improvement')
plt.xlabel('Turbulent Intensity')
plt.ylabel('Inlet Velocity (m/s)')
plt.title('Scatter Plot with Color Mapped Expected
    Improvement')
plt.grid(True)
plt.show()

print(reverted_ei.shape)

#show point for next evaluation
print('The best expected improvement is: ',
    reverted_ei[np.argmax(reverted_ei)], 'at co-
    ordinates', all_points_standard[np.argmax(
    reverted_ei),0],',', all_points_standard[np.
    argmax(reverted_ei),1])
```

Listing 2: Emulator Code