Ballistic missile range estimation and maximisation

Tomas Paulik tp530@cam.ac.uk
Mohd Sadiq ms2892@cam.ac.uk
Ilaria Sartori is541@cam.ac.uk
Vilde Schulerud Bøe vbs29@cam.ac.uk

17th of January 2023

Abstract

In today's world, ballistic missiles are becoming an increasingly important topic. This document explores Gaussian process based emulation of the underlying physical properties of ballistic missiles, in the context of the given simulation-emulation framework. Specifically, we investigate range estimation and maximisation of a ballistic missile. We use two different simulators: a low-fidelity one and an high-fidelity one, the latter being used as ground truth. Moreover, sensitivity analysis is performed and, in the end, a multi-fidelity approach proposed. The entire source code of the project is publicly available at our GitHub repository: https://github.com/tp530/ballistic-missile-range.

1 Introduction

1.1 Ballistic Missiles

Intercontinental ballistic missiles ("ICBMs") are often seen as a relic of the Cold War. However, with the ongoing war in Ukraine and growing geopolitical tensions related to the status of Taiwan, there exists a non-zero probability of ICBMs being used in the near future as post-diplomatic means to resolve these conflicts. Therefore, we consider the topic of ICBMs as increasingly important.

The work in this project is centred around range estimation of a ballistic missile based on its physical properties. This can have a number of practical applications in the design and construction of both the ICBMs and anti-missile systems.

Figure 1: A Trident II launch from a submerged Royal Navy submarine. The missile has an estimated operational range of 12,000 km.

1.2 Laplace's Demon and the need for simulation / emulation frameworks

In the history of science, Laplace's Demon is a philosophical concept arguing that if we had infinite knowledge (i.e., an accurate understanding of how the world functions, a way to model it, the data to combine it with, as well as sufficient means of computation), we could make accurate predictions of the state of entire universe at past, future, or present time. However, as the concepts argues further, because of the "gremlin of uncertainty", our predictions are not deterministic since we don't have infinite data, infinite compute, or a complete understanding of the physical world. Therefore, the phenomena in the physical world carry a level of uncertainty, and we need statistical tools to make reasonable predictions.
A simulator is a computer programme which models a real-world concept by using, for example, the laws of physics. Typically, simulators can deliver valuable results, but are computationally expensive to run. This limitation presents an opportunity for machine learning based tools known as emulators. An emulator is a data-driven model which attempts to capture the most important characteristics of the given phenomena in a computationally efficient way. Given the scope of our project, Gaussian Process ("GP") based emulators are particularly suitable, since they can utilise even a small number of data points to model the target behaviour and provide reliable estimates of their own uncertainty.

Submitted as a group project report for Machine Learning and the Physical World at the University of Cambridge.

Once we have a simulation-emulation framework in place, we can use it to study the underlying physical system. To determine how uncertainty of the output is affected by different uncertainties among the model inputs, we use a technique known as **global sensitivity analysis**. A detailed description of the theory and formulas for sensitivity analysis can be found in Appendix A.4.

In general, if we have different models of the same system with different levels of accuracy, we can attempt to combine them to create a more powerful model, while keeping the computational overhead reasonably low. This technique is known as **multi-fidelity analysis** ("MF").

1.3 The Physics of Ballistic Missiles

This project delves into simulating and emulating the functionality of a ballistic missile by studying the equations of laws of motion that apply to it. A general outline of the trajectory of ballistic missiles are described by the article presented by Wright et. al. (1992)[8]. The formula used for finding the total forces acting on a ballistic missile in trajectory is given by the equation 1.

$$\vec{F} = \vec{T} - m(t).\vec{g} - \frac{1}{2}\rho C_d A|v|\vec{v} \tag{1}$$

The first term (T) refers to the thrust generated by the missile. The second term encapsulates the gravitational forces acting on the missile and the third term refers to the drag force the missile faces in motion. $m(t)$ signifies the mass of the missile at a given time t, $\vec{g}$ refers to the acceleration due to gravity at time t (instantaneous acceleration due to gravity), ρ refers to the density of the atmosphere, A refers to the cross-sectional area of the missile in the direction of motion, $|v|$ refers to the velocity and C_d refers to the drag coefficient of the missile.

1.3.1 High-fidelity ("HF") simulator

The trajectory of the missile is assumed to be in a plane and the calculations are done in polar form with the centre of the earth taken as origin. The distance covered on the earth's surface is measured by the angle of displacement, and hence the range can be calculated as the product of the angle and the radius of the earth. This helps in defining the differential equations for the simulator as given in equations 2- 6.

$$\frac{dV}{dt} = \frac{T}{m}cos\eta - \frac{C_d\rho V^2 A}{2m} - gsin\gamma \tag{2}$$

$$\frac{d\psi}{dt} = \frac{Vcos\gamma}{R_e + h} \tag{3}$$

$$\frac{d\gamma}{dt} = \frac{d\psi}{dt} + \frac{T}{Vm}sin\eta - \frac{g}{V}cos\gamma \tag{4}$$

$$\frac{dm}{dt} = \frac{T}{g_0 I_{sp}} \tag{5}$$

$$\frac{dh}{dt} = Vsin\gamma \tag{6}$$

$\frac{dV}{dt}$ defines the rate of change of velocity (acceleration), $\frac{d\gamma}{dt}$ defines the rate of change of angle between the missile and the vertical axis, $\frac{d\psi}{dt}$ defines the rate of change of positional angle with respect to the earth's centre, $\frac{dh}{dt}$ defines the rate of change of height of the missile and $\frac{dm}{dt}$ refers to the rate of change of mass of the missile. A few extra terms are introduced in the given differential equations like η is the angle between the thrust vector and the missile. Since our missile will always be thrusting in the direction along the axis of the missile, η will always be 0^o.

For incrementally small time intervals, the rate of changes can be assumed to be constant for that duration of time. This leads to the values of these parameters to be calculated as

$$V_{n+1} = V_n + dt.dV; \;\; \psi_{n+1} = \psi_n + dt.d\psi; \;\; \gamma_{n+1} = \gamma_n + dt.d\gamma; \tag{7}$$

$$m_{n+1} = m_n + dt.dm; \qquad h_{n+1} = h_n + dt.dh; \tag{8}$$

Also, there are a couple more conditions that change with increase in height and velocity like the air density, temperature, coefficient of drag and thrust. These conditions are described in the Appendix.

The output that is the final range can be represented as a simple product of the angle travelled and Radius of the earth.

$$Range = \psi.R_e \tag{9}$$

1.3.2 Low-fidelity ("LF") simulator

For this simulator, there are a few assumptions that can be made to reduce the amount of calculation done for the trajectory of the missile. One such assumptions that most engineers/physicists make is to ignore the effects of air drag on the missile. Along with that the different conditions of density, drag, thrust and temperature that are effected by the altitude or speed that the missile achieves. Also in the high fidelity simulator the equation 5 is a constant term that is expressed in terms of thrust and specific impulse. To make calculations easy for this low fidelity simulator, the rate of change of mass is assumed to be a constant term as a parameter. Hence, the reduced form of differential equations used in this simulator are given by equations 10.

$$\frac{dV}{dt} = \frac{T}{m} - g.sin\gamma \tag{10}$$

$$\frac{dh}{dt} = V.sin\gamma \tag{11}$$

$$\frac{d\gamma}{dt} = \frac{d\psi}{dt} - \frac{g.cos\gamma}{V} \tag{12}$$

$$\frac{dm}{dt} = k \tag{13}$$

The range of the missile is calculated the same way as the high-fidelity simulator as the product of angle travelled and Radius of the earth as given in equation 9.

2 Methodology

2.1 Project code and its dependencies

The entire source code of the project, including the simulators, is publicly available at our GitHub repository. The experiments described in the following sections were run on Jupyter notebooks (see the `notebooks` folder), and the corresponding exported PDFs (i.e., the notebook code and all the outputs) can be found in the `NotebooksPDFs` subfolder.

To build the code for these analyses, we have utilised the code from Emukit's tutorials [5] and L48 lecture notes [6].

2.2 Simulator description and design

Our work is based on a simulator originally developed at MIT in 1992[8]. The original simulator, our new implementation, and the structure of the code are described in Appendix A.2.

For the purposes of this project, we have exported the simulator's application logic into a standalone ".ipynb" file, which can be easily imported into a Jupyter notebook.

The simulator is implemented in the "Simulation" class, which takes the following attributes as a part of its constructor object:

- *payload* (float) – The extra mass which is attached to the missile (in kg). It is typically the warhead which detonates on impact.
- *missilediam* (float) – The diameter of the cross-sectional area of the missile that is in the direction of motion (in meters).
- *rvdiam* (float) – The diameter of the cross-sectional area of the missile in the direction of motion, after it has burnt all its fuel and exhausted through all its stages. For a single stage missile considered in this project, the RV diameter is the same as the missile diameter.
- *numstages* (int) – The number of missile stages.
- *fuelmass* (List[float]) – the collection of fuel masses at different stages (in kg) burnt during the trajectory. Once the fuel runs out, thrust drops to zero. The total mass of the missile is defined as the sum of fuel mass, dry mass, and payload.
- *drymass* (List[float]) – The weight of the missile stages (in kg) without the fuel. It depends on the structure, density, and the amount of material used for the construction of the missile. Dry mass, combined with the payload, is the minimum mass the missile has throughout its trajectory.
- *Isp0* (List[float]) – Specific impulses of the missile at different stages (in seconds).
- *thrust0* (List[float]) – the force generated by the missile by burning of the fuel at different stages (in Newtons).

The last four attributes are represented as Lists given that the missile can have up to three stages.

As an output, the simulator generates the whole trajectory of the missile. For the purposes of this project, we're only interested in the range of the missile.

2.3 Emulator description and design

We will demonstrate how to build emulators to address the following two tasks:

1. Range estimation of a ballistic missile
2. Range maximisation of a ballistic missile

For the paragraphs below, we're introducing the following nomenclature:

- "Features" are the attributes of which values will be changed to investigate how the target depends on them.
- "Attributes" are the remaining attributes, which will be of a fixed value, taken from the pre-set missile (Iraq's Al-Husayn)[3].

3 Experiments

3.1 Defining the feature space

One of the important things to note when working with a physical world simulator is the range of values that are present in the domain of the simulator. These values need to be tested out in various combinations to accurately determine where the simulator works properly. There are two conditions where the simulator fails or doesn't return a value:

- *Simulation takes too long*
  This is a condition implemented by the original authors of the simulator to prevent the simulator taking too long. The stopping condition is when time (time used in simulation) crosses 20,000 seconds.
- *Missile is too heavy to lift off*
  The other condition depends on the mass of the missile. This condition can be checked when $\frac{T}{m} < g.sin\gamma$. If this condition holds true, then the missile is too heavy to lift off

A pseudo code implementing the conditions above is provided in AppendixA.3.

Even though the first condition could be potentially removed since it doesn't correspond to an actual physical constraint, we decided to keep it due to practical reasons. By keeping the first condition in the code, we create a more sophisticated version of our otherwise simple target function, and so we can test whether our emulator would be able to detect and deal with strange function shapes, such as the one in Figure 4. This could be particularly useful in other settings, characterised by target functions with natural discontinuities.

For the purposes of this project, we will consider only missiles with only one stage. Furthermore, out of the parameters listed in section 2.2, we will consider as features only the last four, given that they are the easiest ones to change when setting up a missile launch.

3.2 Ballistic missile range estimation

Following the nomenclature defined in the chapter above, we will build models with 1, 2, and 4 features. In other words, we will investigate how the missile range changes as we change either 1, 2, or 4 input values.

All the models are built using Gaussian Processes (GP). In the current and the following two sections, the standard implementation of GP is used, whereas in the last section we will employ GPs as budling blocks of the Multi-Fidelity (MF) analysis. In this section, the estimation is done using Random samples as initialisation and then running an experimental design loop. The loop identifies the data points (i.e., the input locations) where to perform the next simulation, to minimise the uncertainty of the estimated function. This candidate point selection is done maximising the integrated variance reduction ("IVR") acquisition function. [1]

A GP is completely defined by a mean function, indicating the mean at any point of the input space, and a covariance function (a.k.a. kernel), which sets the covariance between points. When trying to fit a model using GPs, two aspects are mainly optimized: the kernel and the likelihood. As far as kernels are concerned, a lot of different functions (positive definite) can be used. Once these have been set, we still need to tune hyperparameters. We will do it by providing initialisation values and, potentially, constraints to the built-in optimise function, that will find the set of values which maximises the marginal log-likelihood.
For simplicity we will consider the mean to be equal to zero, but we will add a Linear term when defining the kernels. Then, we will use RBF (i.e. Radial Basis Function) as building blocks to define our kernels, due to their flexibility and well-established usage tradition in similar settings.
For the likelihood, we will use the default EmuKit implementation, which describes a Gaussian distribution with zero mean, and fix the variance to a low value `var_noise = 1e-5` to indicate that we consider observation from the high fidelity simulator as almost exact realisations of our ground truth system.

More in detail, these are the kernels we use for the models in this section:

- One-feature (fuelmass) model: RBF(input_dim=1) + Linear(input_dim=1).
  We expect the changes in range with respect to fuelmass to be smooth, so in general RBF is a good choice. We choose a relatively large initial value for lengthscale (100) to encourage very slow changes. We also expect the range to be strictly increasing, so adding a linear kernel is a good idea. We add instead of multiply, since we don't want the amplitude of RBF swings to increase out from the origin[2].
- Two-feature (fuelmass+isp0) model: RBF(input_dim=2) * RBF(input_dim=2) + Linear(2)
  We would in this case expect a general increase in range for increased fuelmass and isp0, so again adding a linear kernel is a good idea. We use RBFs for the same reason described above, but we introduce a product of two to allow for modelling at two different lengthscales (starting points: 500 for fuelmass and 100 for isp0). As for what described above, we know that the simulated data will have a discontinuity for very high fuelmass and isp0, so, unless correctly sampled, the fit there will be harder.
- Four-feature (fuelmass+isp0+drymass+thurst) model: RBF(input_dim=4) + Linear(input_dim=4)
  Since this is the most difficult model to tune (high computational cost for running simulations and complex result visualisation), we opted for a fairly simple kernel. Note that the lengthscale is drastically reduced (initial value 0.1), since for this experiment we re-scale inputs to be in the [0-1] interval.

Note: the metric use to evaluate the accuracy of the models is Root Mean Squared Error (RMSE). This is computed on increasingly sparse grids as the number of features increases, due to computational reasons. Namely, we use 301 equally spaced points for 1D, 101 in each dimension for the 2D case, and 21 for the 4D.

For the **one-feature model**, we present the results generated with fuel mass as the feature. The two charts in the figure below represent the initial state (left) and the final state (right) of the experimental design loop. The initial RMSE is equal to 7, while as the experimental loop is concluded it falls to RMSE=1.

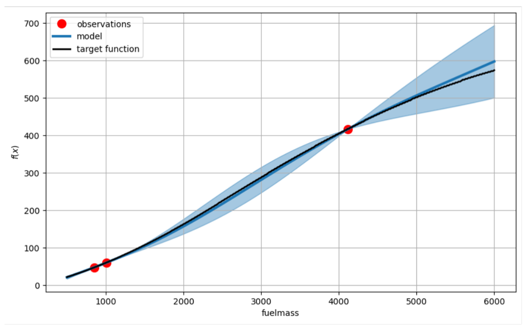


Figure 2: The initial state, with three data points randomly selected from the input domain.

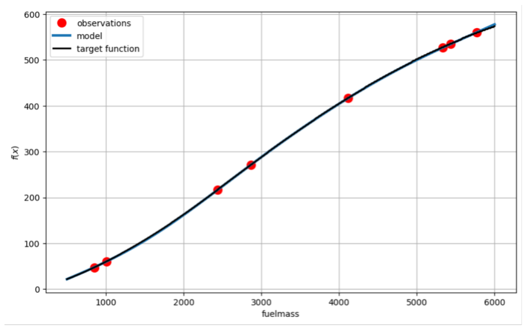


Figure 3: The final state, after five iterations of the experimental design loop.

For the **two-feature model** we focused on fuelmass and isp0, since from Sensitivity analysis (see 3.4) these appear to be the most influent features for the range definition. We will use this model to show-case range maximisation in the next section, since it allows us to visualise steps and results.

For the **four-feature model** we consider fuelmass, isp0, drymass and thrust. This is of course the most comprehensive model, which comes with the price of being quite complicated and expensive to tune. Moreover it's hard to visualise, unless we decide to observe various projections of the analysis freezing 2 features at different levels, but this would somehow defeat the purpose of the analysis. Nevertheless, from our basic experiment we can say that after just 20 iterations of the experiment design loop, the RMSE drops from 3043 to 1291. Moreover in 3.4 we will show how we can use this model to get a directionally correct approximation of the Sobol indices of the system.

Seemingly across all the models experimented with for the estimation of range, it was observed that by design, error in the measured points is zero and the variance increases as the data taken into consideration extends further across the domain of the dataset points. The model can interpolate the parameters with high confidence but introduces ambiguity when extrapolating the parameters.

3.3 Ballistic missile range maximisation

One great advantage of GPs is their flexibility. Indeed using a very similar strategy to the one seen in the previous section we can build a Bayesian optimisation loop that will allow us to find the feature values set which maximises the range. The main variation we need to introduce regards the acquisition function, since we would like it to be able to consider a trade-off between exploration of regions where the model is uncertain, and exploitation of regions where the model is confident our utility function will be high. Specifically, for this purpose we chose the`ExpectedImprovement` [9] function provided by EmuKit.

As anticipated we will use a two-feature model to present this use case. Note that, since the default implementation performs minimisation, we will use minus the range as target. Specifically we see that after 6 random evaluations of the target function (initialisation) and 10 iterations of the Bayesian optimisation loop, the location of the minimum found by the emulator is within 1 unit distance of the actual minimum of the target function, and the error on the estimated value is smaller than the 1%.

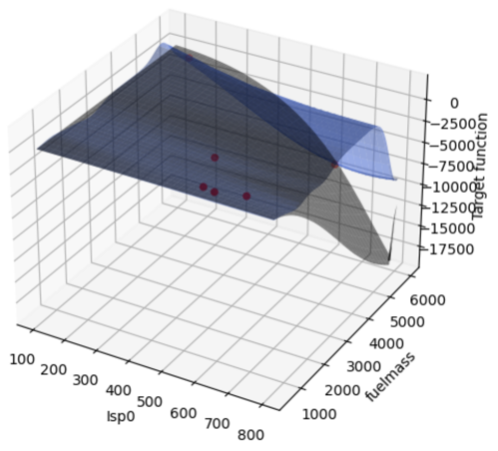


Figure 4: Initial state, 6 randomly selected data points. In black we have the target function, in blue the model mean prediction, and in red the points where the simulations have been run.

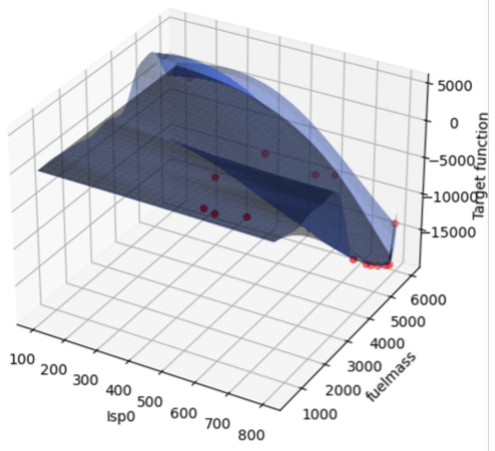


Figure 5: Final state, after 10 iterations of the Bayesian optimisation loop. In black we have the target function, in blue the model mean prediction, and in red the points where the simulations have been run.

From the bottom right image (Figure 7), we can observe how the emulator tends to sample multiple times in the top-right area, to get a good approximation of the target function around its minimum. This behaviour is even more evident when we compare this results with the ones we would get running instead an Experimental design loop as the one described in the previous section (see Appendix A.5). Indeed we see that, even with less iterations, the Bayesian loop returns more accurate predictions in the area where the function reaches the minimum. On the other hand, the total RMSE is higher, since exploitation drives us to sample less often and so produce noisier prediction in the less interesting areas.

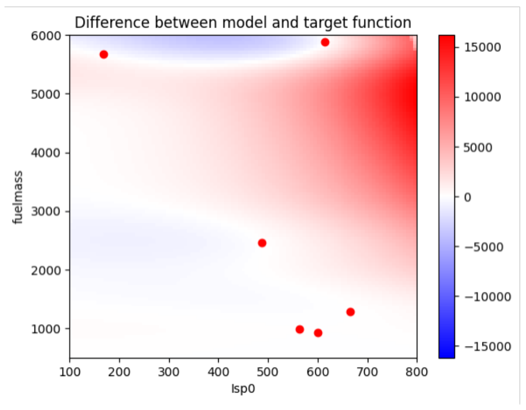


Figure 6: Initial state, 6 randomly selected data points – Heatmap of the difference between the model and target function.

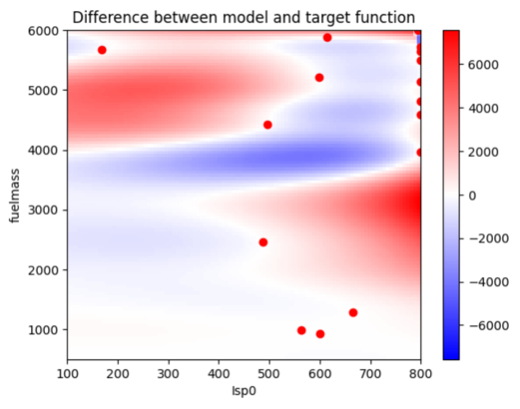


Figure 7: Final state, after 10 iterations of the Bayesian optimisation loop – Heatmap of the difference between model and target function.

We can also note how the strange peak in the top right corner of Figure 4, which is clearly due the truncation of the simulations that would have exceeded the maximum allotted time. The emulator can effectively detect this behaviour and correctly locate the minimum on the drought that precedes this peculiarity.

3.4 Sensitivity analysis

We run sensitivity analysis, as introduced in section 1.2 (and detailed in Appendix A.4), considering 4 features (fuelmass, isp0, drymass and thrust). Since sensitivity analysis aims at investigating the behaviour of the system under scrutiny, we would ideally like to use the most accurate measures we have: the HF simulator. For other problems, this is typically unfeasible due to the long computation time and cost, and so emulators can again come to the rescue.

The sensitivity framework typically expects input distributed uniformly over [0,1], so we rescale our input features to this range. This will also help in preventing the exploding gradient issue.

These are the Monte Carlo approximations of the first-order Sobol indices for our system, computed using both the simulator (blue) and the emulator (orange):

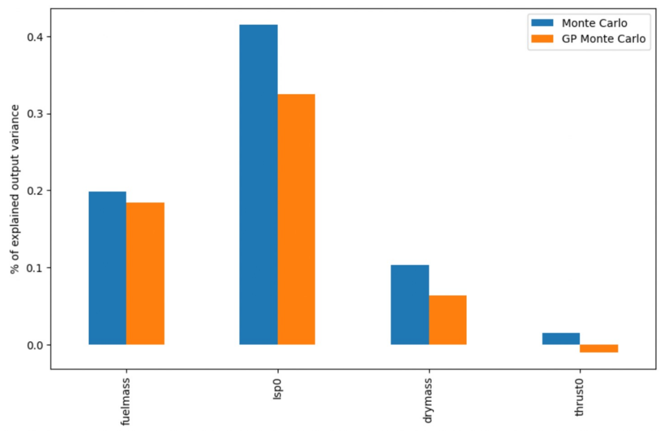


Figure 8: First-order Sobol indices: fuelmass, isp0, drymass, thrust0

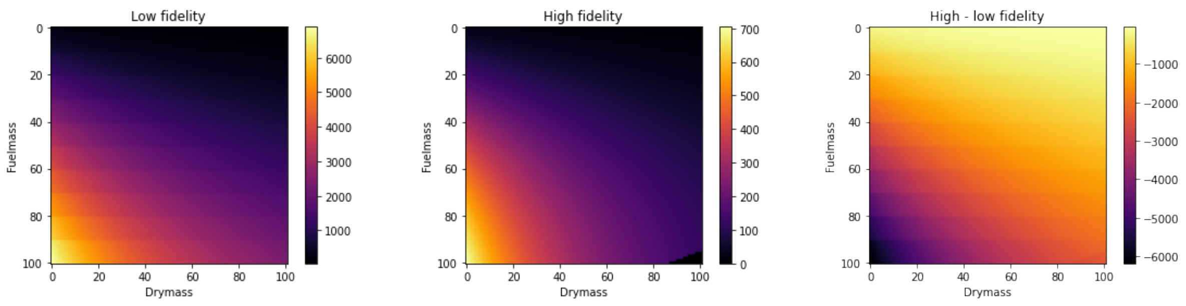
We can first notice that the sums of indices are less than 1 for both the simulator and GP approximation; following the theory of A.4 We may however also notice that the height of bars are quite different for the simulator- and emulator-based approximations. This must be expected since we were not able to get a very good fit to the 4-features model. But the order of responsibility is the same, which is good.
Isp0 has the highest index, and is hence the feature which explains the largest amount of variance in the output - when we don't consider any interaction terms in the model. Similarly, thrust0 explains very little of the output variance. Therefore, if we want to simplify the model, setting thrust0 to a constant value would enable us to keep most of the variety in output, while making the model simpler and hence easier to deal with for further analysis.
This can also be useful information when designing a missile launch; if we need to change the missile's range, the most effective alteration would be isp0, or alternatively adjusting the fuelmass. If we try to change drymass or thrust, the result will probably not change as much.

Another important note is that, since the indices are approximated using Monte Carlo, we never know for sure if we have reached the correct indices. We might get more accurate results by increasing the number of Monte Carlo iterations.

3.5 Multifidelity modelling

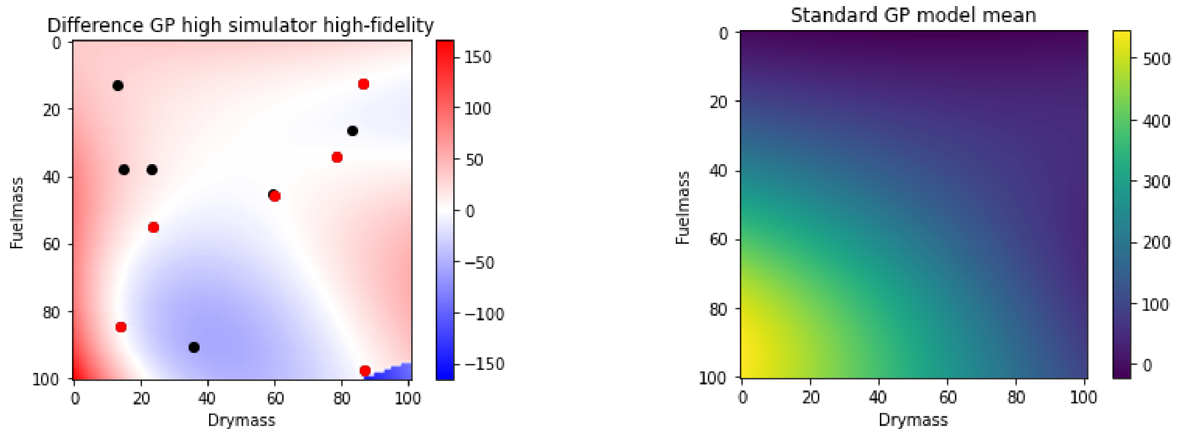
Multifidelity modeling is a way of stitching together information from sources of different degree of exactness [7] [4].

For the given study, the two simulators taken into consideration have varying ranges of uses. One is a high-fidelity simulator that is accurate but slow, which is considered to generate the ground truth. On the other hand we have a low-fidelity simulator that makes a lot of approximations but can return the output value relatively quickly. To visualise the functioning of the two simulators, they were sampled on a 101 x 101 grid. Plotted are the 2 simulators, and the difference between them:



In order to compare information from the simulators, we must look at comparable inputs parameters. Fuelmass, drymass and missile diameter is used in both simulators, and in reference to the real world aspect of a ballistic missile, fuel mass and dry mass were taken into consideration. By using two-dimensional input, we are able to visualise results.

The results will be compared to a reference model, defined as a **simple GP emulator** trained only on six random samples (the ones where we will evaluate the HF simulator in the MF models described below).
To avoid blob-like features in the result, we initialise the hyperparameter length-scale to a high value. This will encourage the optimised lengthscale-value to also be high. The model is similar to the one described in section 3.2 (2 features), since the only difference is that we expect a decrease in range for increased drymass, which should not affect the kernel choice.
The final GP model resulted in a Root Mean Square Error of 30,5.
The plot below illustrates the mean of the final model to the right, and the difference to the HF simulator (i.e. our target), to the left. The 6 red dots indicate the points where the HF simulator was evaluated. As expected we note that the error there is zero.



In the image above, along with the 6 red dots, are reported 6 extra black dots. Indeed, for multi-fidelity we will still evaluate the HF simulator in correspondence of the red dots, while the LF simulator will be evaluated on both these points and the extra black ones (12 points in total).

A **linear MF model** assumes a linear relationship between the LF and HF simulators:

$$f_{high}(x) = f_{err}(x) + \rho f_{low}(x)$$

In emukit two kernels are specified; one for the bias f_{err}, and one for correlation between simulators. The LF simulator resulted to have lines of non-smoothness on top of the general smoothness, hence it would be difficult to fit a GP to the true bias and correlation. Instead we opt to capture the general trends, so the same kernel was used as the pure GP, for both bias and correlation. Again we can fix the noise at both fidelities to zero, to reflect that we observe the exact values of both simulators.
Unfortunately we did not have time to fine-tune for this, so there might exist other kernels that would yield better results.
After optimizing the hyperparameters, the mean posterior of the HF emulator, and the difference to the actual simulator grid were plotted.
The final model has RMSE=45,9.

The result is worse than the pure GP model. This is because the linear MF model cannot capture a non-linear relationship between the LF and HF data. Such relationship can be due to the different assumptions and simplification of forces between the HF and the LF simulator. This is non-linear, and therefore the relationship between the simulators is nonlinear and so hard to capture through a linear kernel. The extra data-guidance given by the LF data points can be more misguiding than helpful, and therefore the results might be worse than the pure GP model.
This non-linearity can be confirmed by plotting the LF data against HF data:

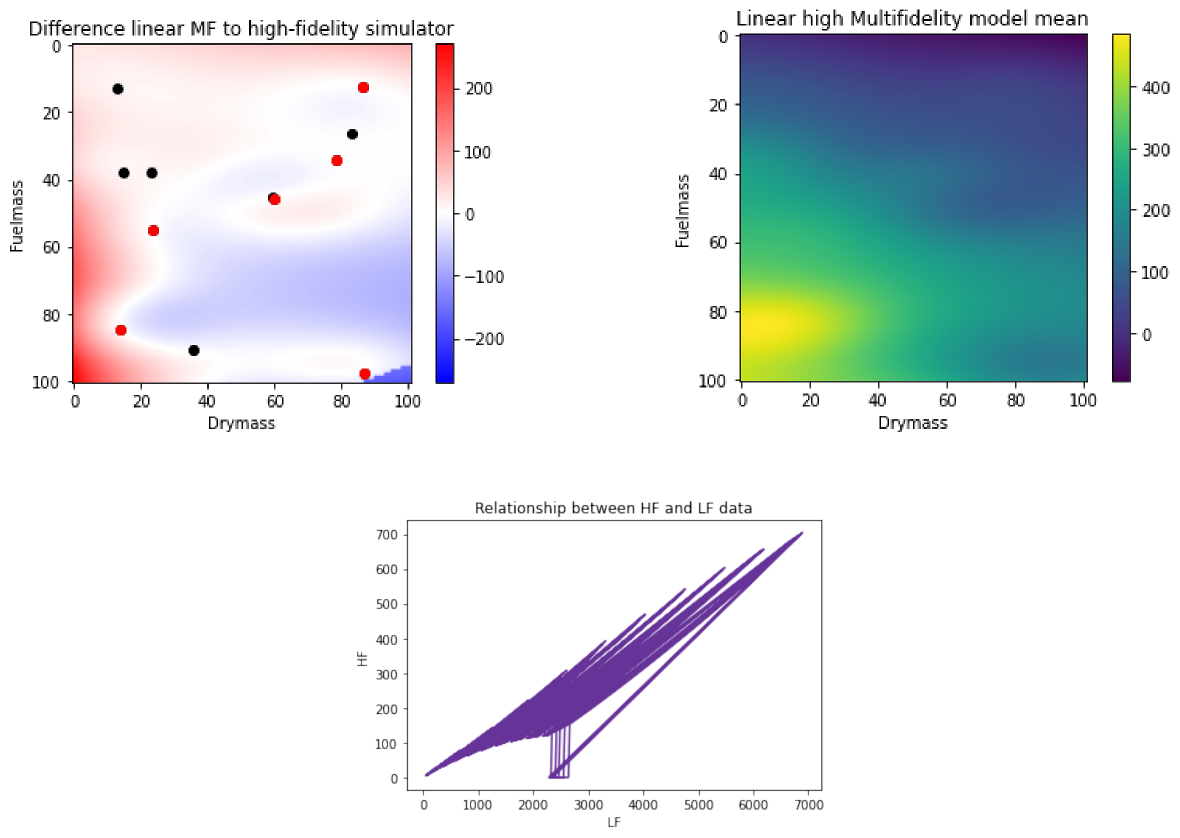


Figure 9: Non-linear relationship between LF and HF data

To capture the non-linear relationship, a **nonlinear multi-fidelity model** is used:

$$f_{high}(x) = \rho(f_{low}(x)) + \delta(x)$$

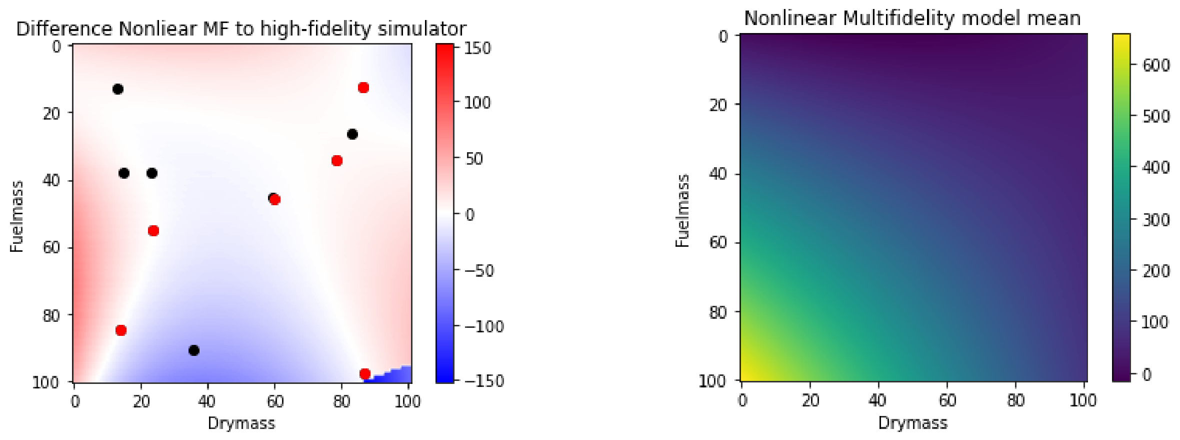
The number of fidelities that is considered while designing this kernel is still 2. The kernel at the base level is defined as

$$k_{base}(x, x') \tag{14}$$

For every subsequent level, the kernel is computed as

$$k_{base}(x, x')k_{base}(y_{i-1}, y'_{i-1}) + k_{base}(x, x') \tag{15}$$

[5]. For this study, the base kernel function taken into consideration was the linear kernel, resulting in second order kernels for the bias and correlation. Also here we set the noise-level to zero. After the optimisation of the hyperparameters, the final model has RMSE=22,7.



Now we are able to model the (nonlinear) relationship between the simulators much better, and can see the benefit of MF modelling; we have our best result yet wrt RMSE. We can also see from the plotted difference, that the top left corner is now a better fit than the pure GP (color closer to white), so the black LF points has given us some valuable information.

4 Conclusion

4.1 Discussion

In conclusion, the analysis presented in this project demonstrates the power of the used simulation-emulation framework, and how GPs can be utilised here to deliver meaningful results.

The document also identifies the challenges that arise when dealing with a high number of features in a system. As the number of features increases, it becomes increasingly difficult to capture the entirety of the system and the relationships between inputs and targets. However, the results presented in this analysis showcase the potential for improvement with additional fine-tuning of the model.

Neural networks have been shown to be effective in a variety of applications, and their use as a substitute for traditional Gaussian Processes is worth exploring. The achieved results of this analysis may seem relatively simple, but the methodologies presented are powerful and can be used for much larger and complex analyses.

To make the data more interpretable and easily visualisable, we chose to present basic examples, which, for example, use 2D space instead of 4D. However, it is important to note that this decision was made purely to simplify the presentation of the analysis and should not be seen as a limitation of the methodology.

Overall, our project highlights the potential of machine learning techniques in the field of data analysis and the importance of advanced fine-tuning of models when dealing with a large number of features.

4.2 Future work

As this project studies the aspects of optimisation of range of a ballistic missile, parallels can be drawn towards optimisation of energy used, least resistance trajectory, or maximum impact force of the missile. This could be achieved by designing custom acquisition functions and reward functions tailored to the needs of the specific use case.

References

[1] A.N. Avramidis and J.R. Wilson. Integrated variance reduction strategies. In *Proceedings of 1993 Winter Simulation Conference - (WSC '93)*, pages 445–454, 1993.

[2] David Duvenaud. Automatic model construction with gaussian processes. 11 2014.

[3] Geoffrey Ethan Forden. Gui_missile_flyout: A general program for simulating ballistic missiles. *Science & Global Security*, 15:133 – 146, 2007.

[4] Alexander I.J Forrester, András Sóbester, and Andy J Keane. Multi-fidelity optimization via surrogate modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 463(2088):3251–3269, 2007.

[5] https://emukit.readthedocs.io/en/latest/tutorials.html. Emukit documentation. *Tutorials - emukit 0.4.10 documentation.*

[6] Neil D. Lawrence. Emukit and experimental design, Nov 2022.

[7] Neil D. Lawrence. Multifidelity modelling, Nov 2022.

[8] David C. Wright Lisbeth Gronlund. Depressed trajectory slbms: A technical evaluation and arms control possibilities. *Science Global Security*, 3(1-2):101–159, 1992.

[9] J. Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards Global Optimization*, 2:117–129, 09 2014.

[10] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, 2006.

[11] I.M Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and Computers in Simulation*, 55(1):271–280, 2001. The Second IMACS Seminar on Monte Carlo Methods.

A Appendix

A.1 High-fidelity simulator conditions

The thrust generated by the missile is affected by the altitude of the missile. The thrust of at any given time-stamp n is given by equation 9.

$$T_{n+1} = T_n * (-0.4339.h_{norm}^3 + 0.6233.h_{norm}^2 - 0.01.h_{norm} + 1.004) \tag{16}$$

$$h_{norm} = \frac{h_n}{160934} \tag{17}$$

The coefficient of drag also changes with respect to the speed of the missile. The relationship is shown in the equations below.

$$C_d = \begin{cases} 0.15 & if\ mach\ >\ 5 \\ -0.03125.mach + 0.30625 & if\ mach\ >\ 1.8\ and\ mach\ \le\ 5 \\ -0.25.mach + 0.7 & if\ mach\ >\ 1.2\ and\ mach\ \le\ 1.8 \\ 0.625.mach - 0.35 & if\ mach\ >\ 0.8\ and\ mach\ \le\ 1.2 \\ 0.15 & if\ mach\ \le\ 0.8 \end{cases} \tag{18}$$

$$mach = \frac{V}{\sqrt{1.4 * 287 * temperature(in\ K)}} \tag{19}$$

$$temperature(in\ {}^0C) = \begin{cases} 15.04 - 0.00649.h & if\ h\ \le 11000 \\ -56.46 & if\ 11000\ <\ h\ \le 25000 \\ -131.21 + 0.00299.h & if\ t\ >\ 25000 \end{cases} \tag{20}$$

The density that is responsible for calculating the drag force on the missile also varies with height. It is given by the cases

$$\rho = \begin{cases} \rho_0.e^{-\frac{h}{8420}} & if\ h\ <\ 19200 \\ \rho_0.(0.857003 + \frac{h}{57947})^{-13.201} & if\ h\ > 19200\ and\ h\ <\ 47000 \\ 0 & if\ h\ \ge\ 47000 \end{cases} \tag{21}$$

Also for the initial 5 seconds of the flight the missile is assumed to be perfectly vertical, else the differential equation 4 results to be not defined at $t = 0$. It is after 5 seconds that change in γ is observed. Also the gravitational forces change with respect to the altitude gained by the missile. This relationship is shown in equation 22.

$$g = g_0.\frac{h^2}{(R_e + h)^2} \tag{22}$$

A.2 The origin and structure of the missile simulator

The original simulator was written in the BASIC programming language by Dr David Wright of MIT in 1992, who wrote it for his paper "Depressed Trajectory SLBMS", published in the "Science and Global Security" journal.

A Python 2.7 version of the code was created by Josh Levinger for GlobalSecurity.org in June 2005.

We have ported the Levinger's code to Python 3.10, preserving the original computational model of the 1992 simulator.

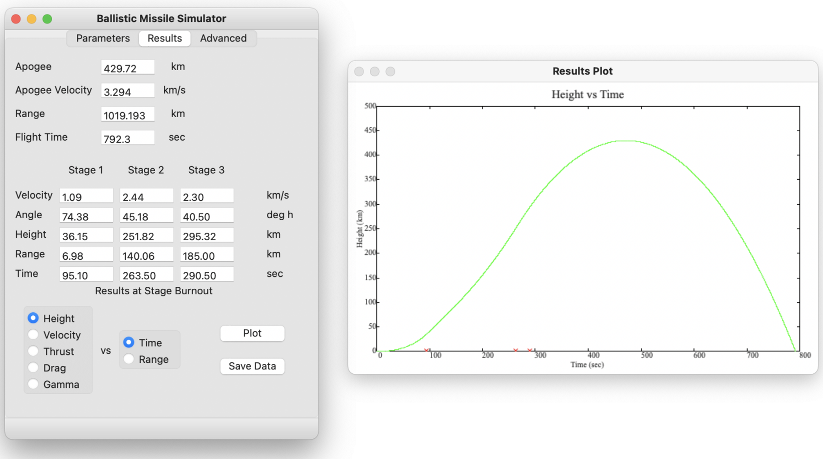
Our version of the simulator consists of the following three Python files:

- *sim.py* – the simulator itself, which can be used as a command line application, or it can be imported as a back end for the UI or notebook.

- *gui.py* – the user interface, developed using the wxWidges cross-platform UI library. It can be directly executed with "python gui.py" (or "pythonw gui.py" when using the Anaconda distribution).

- *plot.py* – the application logic responsible for chart rendering in the UI.

The UI enables the user to input missile parameters and run simulations. Pre-sets for 8 missile types are available for an ease of use. The UI also enables the user to plot the results and export the calculated data.



Ballistic Missile Simulator

Parameters | Results | Advanced

```
Apogee           429.72    km
Apogee Velocity  3.294     km/s
Range            1019.193  km
Flight Time      792.3     sec

          Stage 1   Stage 2   Stage 3
Velocity  1.09      2.44      2.30      km/s
Angle     74.38     45.18     40.50     deg h
Height    36.15     251.82    295.32    km
Range     6.98      140.06    185.00    km
Time      95.10     263.50    290.50    sec
            Results at Stage Burnout
```

[✓ Height] Velocity Thrust Drag Gamma vs [✓ Time] Range

Figure 10: A screenshot of the UI of the simulator, showing the estimated Height vs. Time plot of the "DPRK TD-1" missile.

A.3 Defining the Feature Space

The pseudo code for finding the optimal parameter space is given as below:

```
hash_map = {p: [inf,-inf] for p in parameters}

for parameter in parameters:
    for j in parameter_range:
        range = simulator().get_range(params)
        if range is valid:
            hash_map[parameter][0] = min(j,hash_map[parameter][0])
            hash_map[parameter][1] = max(j,hash_map[parameter][1])
```

The ideal ranges for these values were calculated to be:

Parameters	Initial Parameter Space	Final Parameter Space
Payload	10 - 5,000	10 - 2,410
Missile Diameter	0.1 - 10	0.1 - 10
RV Diameter	0.1 - 10	0.1 - 10
Fuel Mass	2,000 - 20,000	2,000 - 7,000
Dry Mass	1,000 - 5,000	1,000 - 3,000
Specific Impulse	100 - 2,000	100 - 800
Thrust	10,000 - 70,000	10,000 - 70,000

Table 1: Final Parameter Spaces

The important thing to note is that the upper limit were hit for the parameters Missile diameter, RV Diameter and Thrust. Hence it can be assumed that these values can be much higher than they reasonably need to. Regardless this experiment helped in finding out the upper bounds for Payload, Fuel Mass, Dry Mass and Specific Impulse.

A.4 Introduction to sensitivity analysis

Sensitivity analysis is a technique we can use to analyze the variability in a system's output. Local sensitivity looks at the effect of a single input variable while the other variables are set. To understand the variability of the output across the whole input domain, we must look at global sensitivity.

A common measure for the global effect of one or multiple inputs is the Sobol indices:

$$S_l = \frac{var(g_l(x_l))}{var(g(x))}$$

where l may be one or multiple indices. This measures how much responsibility the relevant indices have for the total variance of the output, but does not consider interaction terms between indices if they are not both in l.

To compute this, we need to know the distribution of each input; p(**x**), and assume that the inputs are independent. Typically, we can assume $x_i \sim Uniform(0, 1)$. Then we know that the total variance of our simulator function, g, is

$$var(g(x)) = E[g(x)^2]_{p(x)} - E[g(x)]^2_{p(x)}$$

Furthermore, we use a Hoeffding-Sobol Decomposition of g, where different indices are marginalised out, to get

$$var(g) = \sum_{i=1}^{p} var(g_i(x_i)) + \sum_{i<j}^{p} var(g_{ij}(x_i, x_j)) + \cdots + var(g_{1,\ldots,p}(x_1, \ldots, x_p)) \tag{23}$$

where $g_i(x_i) = E[g(x)]_{p(x_{\sim i})} - g_0$; i is marginalised out.

Then the first-order Sobol index of i is

$$S_i = \frac{var(g_i(x_i))}{var(g(x))}$$

Since we normalise with the total variance, the first-order indices cannot sum to more than 1. If there are any interaction terms creating variance in the output, according to equation 14, the sum of S_i will actually be < 1.

Unfortunately, these integrals are usually infeasible. But we can approximate them, using for example Monte Carlo sampling [11]. This is trivial in emukit, just using the function compute_effects.

A.5 Ballistic missile range maximisation

Range estimation, using experimental design loop with IRV acquisition
Initial RMSE: 3903, final RMSE: 295.

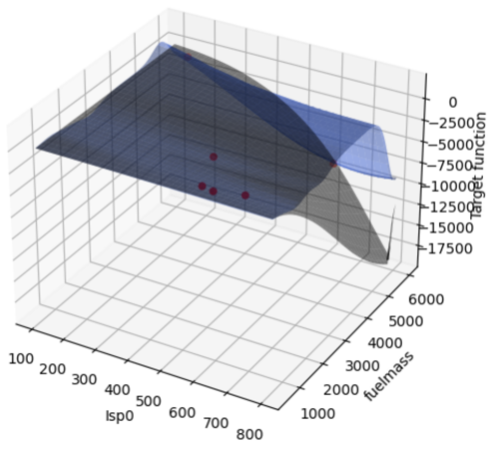


Figure 11: Initial state, 6 randomly selected data points. In black we have the target function, in blue the model mean prediction, and in red the points where the simulations have been run.

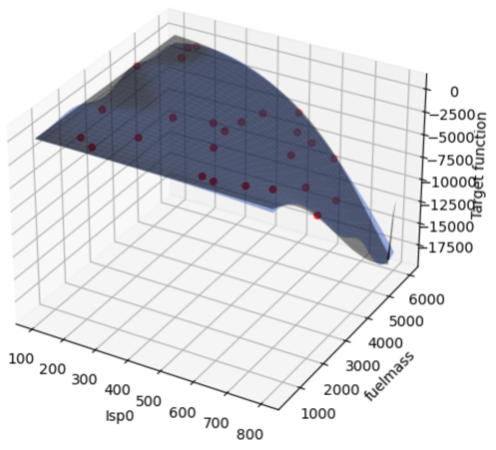


Figure 12: Final state, after 20 iterations of the experimental design loop. In black we have the target function, in blue the model mean prediction and in red the points where the simulations have been run.

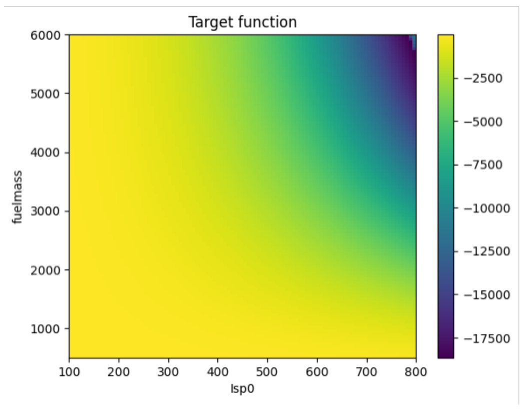


Figure 13: Initial state, 6 randomly selected data points – Heatmap of the target function.

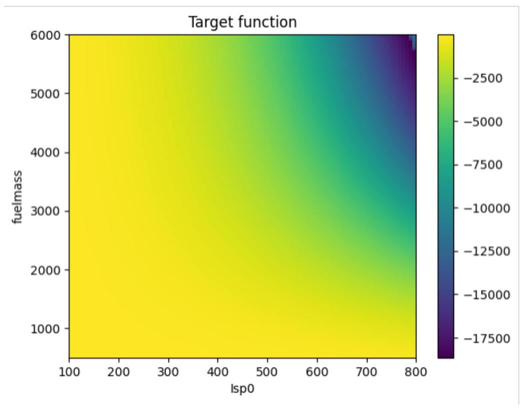


Figure 14: Final state, after 20 iterations of the experimental design loop – Heatmap of the target function.

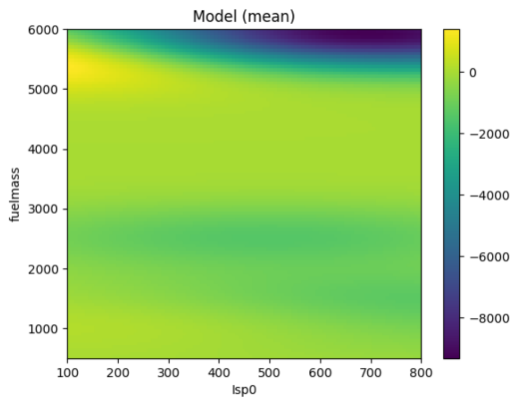


Figure 15: Initial state, 6 randomly selected data points – Heatmap of the model mean prediction.

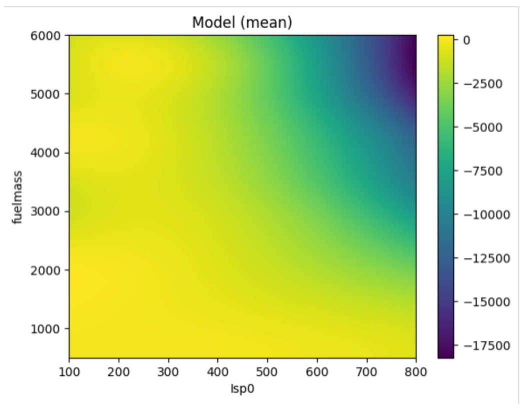


Figure 16: Final state, after 20 iterations of the experimental design loop – Heatmap of the model mean prediction.

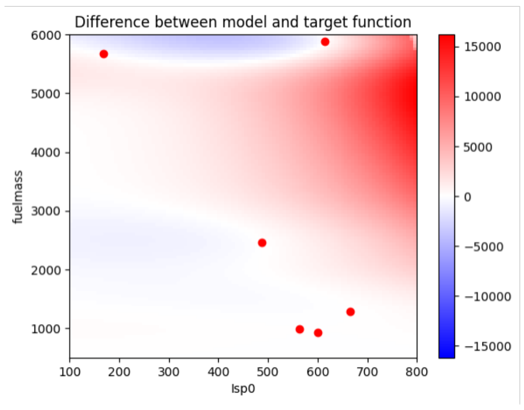


Figure 17: Initial state, 6 randomly selected data points – Heatmap of the difference between the model and target function.

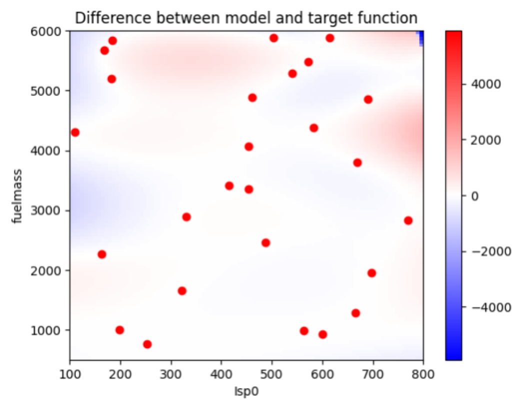


Figure 18: Final state, after 20 iterations of the experimental design loop – Heatmap of the difference between the model and target function.

Range maximization, using Bayesian optimization loop with ExpectedImprovement acquisition
Initial RMSE: 3903, final RMSE: 1897.

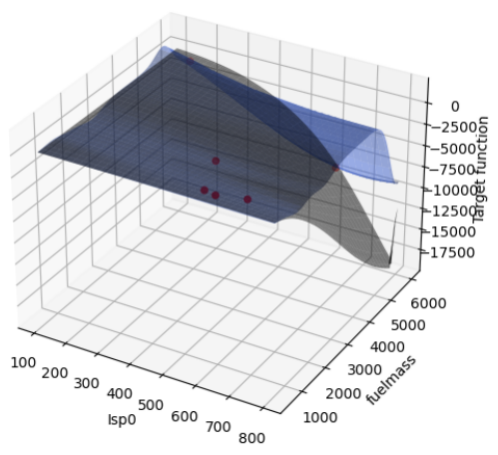


Figure 19: Initial state, 6 randomly selected data points. In black we have the target function, in blue the model mean prediction, and in red the points where the simulations have been run.

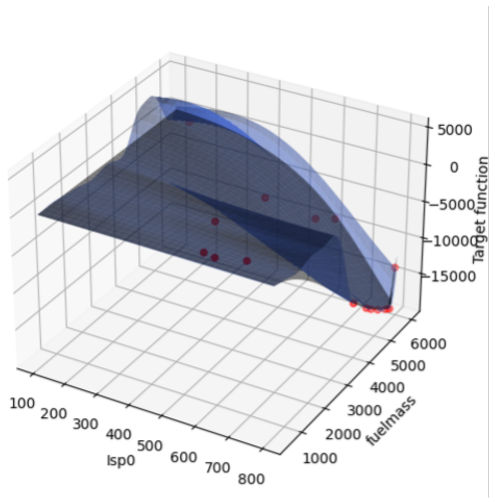


Figure 20: Final state, after 10 iterations of the Bayesian optimisation loop. In black we have the target function, in blue the model mean prediction, and in red the points where the simulations have been run.

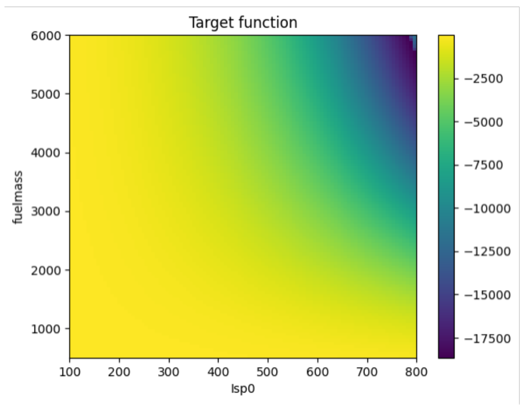


Figure 21: Initial state, 6 randomly selected data points – Heatmap of the target function.

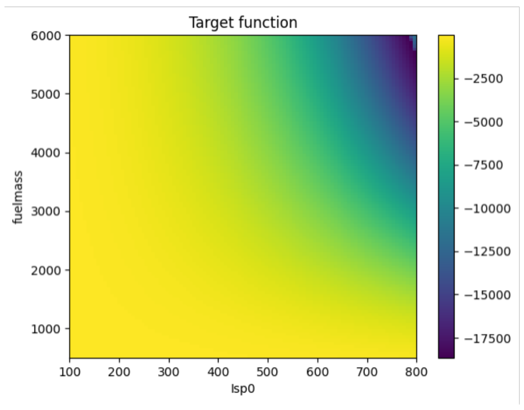


Figure 22: Final state, after 10 iterations of the Bayesian optimisation loop – Heatmap of the target function.

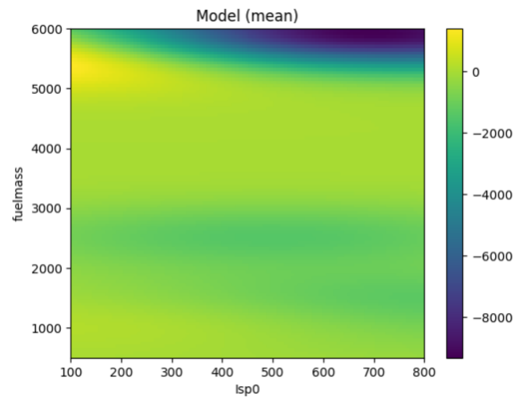


Figure 23: Initial state, 6 randomly selected data points – Heatmap of the model mean prediction.

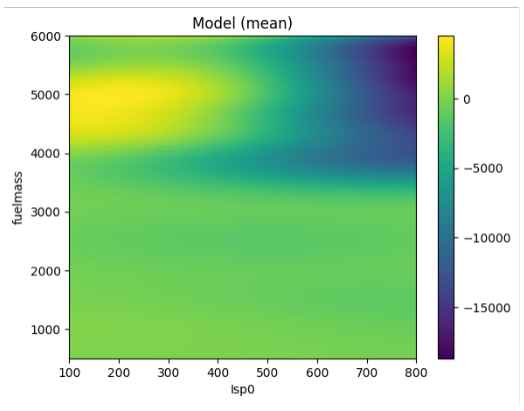


Figure 24: Final state, after 10 iterations of the Bayesian optimisation loop – Heatmap of the model mean prediction.

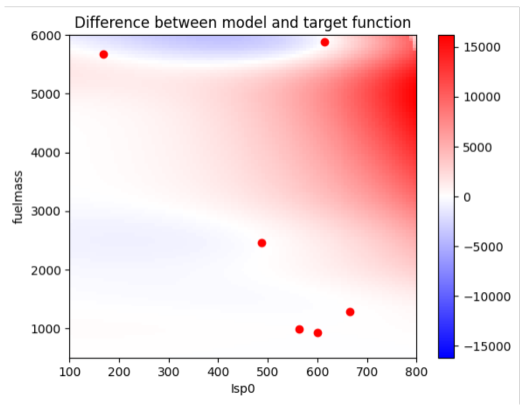


Figure 25: Initial state, 6 randomly selected data points – Heatmap of the difference between the model and target function.

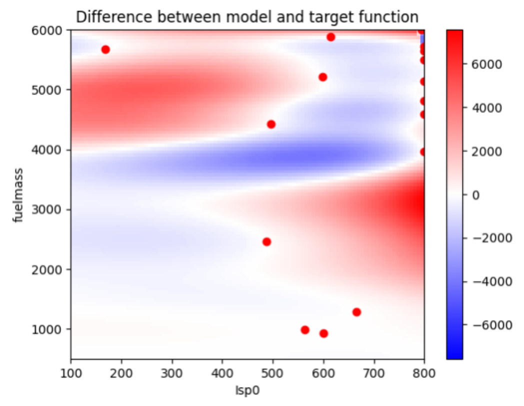


Figure 26: Final state, after 10 iterations of the Bayesian optimisation loop – Heatmap of the difference between the model and target function.