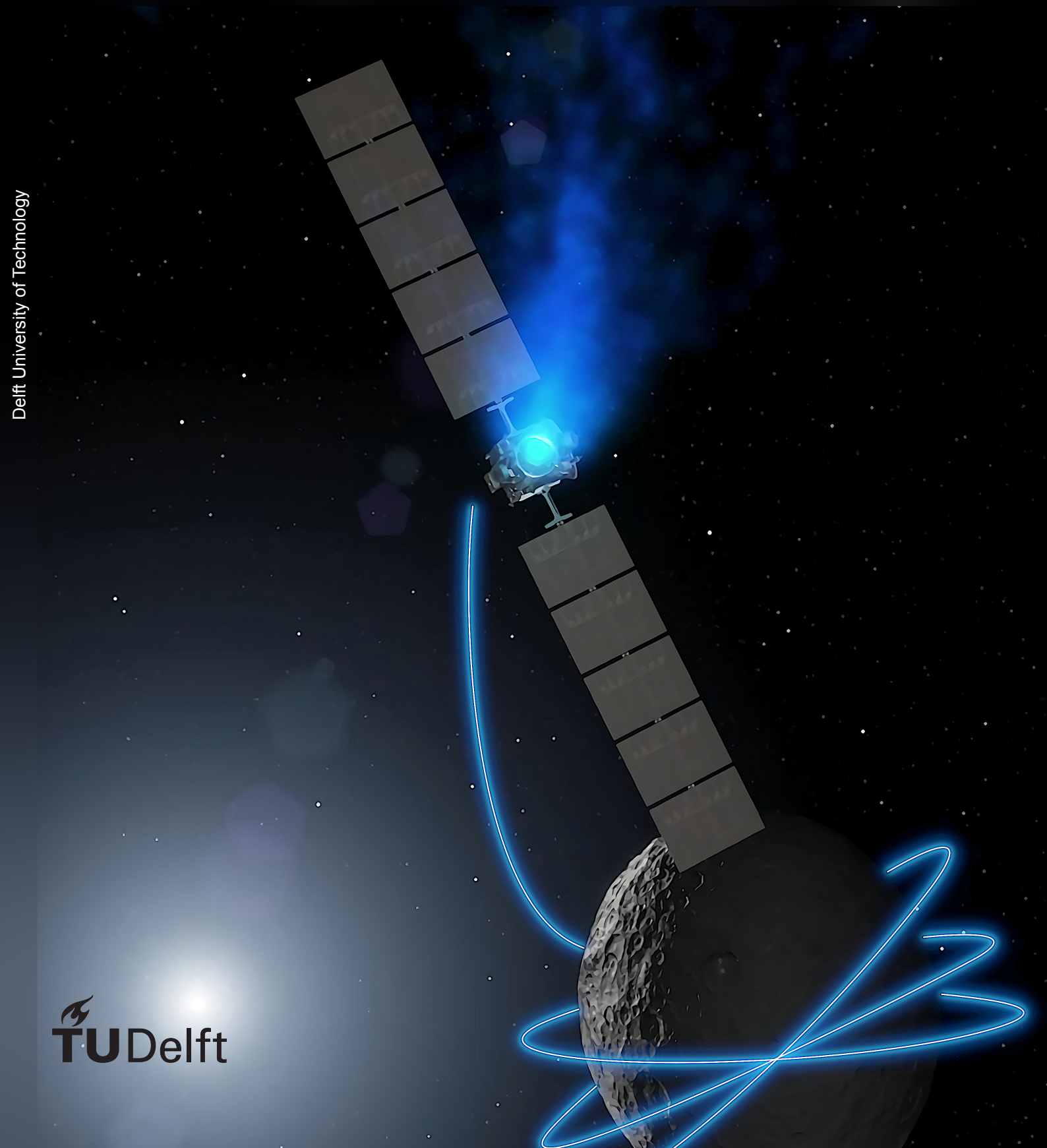
Online Surrogate Models for the Constrained Optimization of Interplanetary Low-Thrust Trajectories

M.Sc. Thesis

Francisco A. Andrade Castanheira



Delft University of Technology

TU Delft

Online Surrogate Models for the Constrained Optimization of Interplanetary Low-Thrust Trajectories

M.Sc. Thesis

by

Francisco A. Andrade Castanheira

to obtain the degree of

Master of Science

at the Astrodynamics and Space Missions Section,
Space Engineering Department, Faculty of Aerospace Engineering,
Delft University of Technology,
to be defended publicly on Friday, 23 September 2022 at 13:00

Student number:	5123562	
Duration:	5 July 2021 – 23 September 2022	
Thesis committee:	Prof. Dr. Ir. P. N. A. M. Visser	Committee chair
	Ir. K. J. Cowan, MBA,	Supervisor
	Dr. S. Speretta,	External examiner

Original Cover Image Taken From:
`https://www.nasa.gov/sites/default/files/thumbnails/image/pia20919.jpg`
(last accessed on 27 March 2021)

TU Delft

Preface

Over a thousand sunsets, and over a thousand sunrises. That is how long it took. From arriving at the land of rain, or the land of biking, or the land of biking in the rain, until delivering this manuscript which marks the end of my academic journey. But the rain is not all that bad. It makes the warm comfort of the house more enjoyable, and the sunny days more appreciated. And even though the rain and I get along well by ourselves, I never felt alone. There are other people under the rain. Other people with whom I shared the grey days, the blue days, and even the white days at times, when not even the rain was brave enough to come out with the cold. To those people, I am very thankful:

Kevin Cowan, for accepting me under his wing, and guiding and motivating me through this thesis, always open to explore new avenues whenever my volatile self decided to pursue something different.

Dominic Dirkx and Geoffrey Garrett for entertaining my way too ambitious ideas in the beginning of this project, and helping me narrowing those ideas down to something that was doable, as well as helping me start to understand where I wanted to take my professional life.

My friends, the ones I lived with, the ones I made in Delft, the ones I visited when coming to Portugal, and the ones who suffered through Propagation and Optimization in Astrodynamics with me. it has been an absolute pleasure.

My grandparents, and in special, my parents, Ana and Luciano. Thank you for giving me this possibility despite how hard it may have felt. And thank you for everything else. Cannot forget about Bruna and Mónica, my dogs, who never fail to lift up my spirits. Finally, a big thank you to Natasha, for always accompanying me despite the rain and being my safe harbor when the storm is too strong.

I don't know if the rain will keep on falling, or if the sun will come out and shine. But I am happy with either, I know where to go, and good things will come.

Francisco Castanheira
Lisbon, July 2022

Abstract

The optimization of interplanetary, low-thrust trajectories is a computationally expensive aspect of preliminary mission design. To reduce the computational burden associated with it, surrogate models can be used as cheap approximations of the original fitness function. Training the surrogate models in a fully online manner can be done to remove the need of having previously generated datasets, which is another source of computational cost. The Sims-Flanagan transcription is used to model an Earth-Mars transfer which is optimized through different optimization routines. The development of a C++ library with machine learning tooling was initiated, containing implementations for Generalized Regression Neural Networks (GRNNs) and Radial Basis Function Networks (RBFNs) that are used in global and local surrogates, respectively, having their hyperparameters tuned through cross-validation. A surrogate model was constructed using Differential Evolution (DE) operators and an uncertainty-based infill criterion for the global search phase, and approximation of the derivative of the original fitness function which is provided to SNOPT (Sparse Nonlinear Optimizer), in the local search phase. An ablation study was performed to assess how each of the components of the surrogate model contribute to the results. It was verified that neither the derivative information nor the local search as a whole led to better results. The surrogate model was also outperformed by the standard optimization strategy found in literature, Monotonic Basin Hopping (MBH). Two new surrogate models incorporating ideas of this strategy were created, with one of them outperforming every other model that was tested. Despite not having performed a full study of the computational effort due to the simulations having been run in a server with a variable load, the new models present better results for similar amounts of fitness function evaluations. A Wilcoxon rank-sum test was performed to assess whether the results have statistical significance, leading to the conclusion that a surrogate model can be used to improve the optimization of low-thrust trajectories modeled with the Sims-Flanagan transcription when inserted in a monotonic basin hopping optimization scheme.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations and Acronyms

AAS	American Astronautical Society
AIAA	American Institute of Aeronautics and Astronautics
ANN	Artificial Neural Network
AU	Astronomical Unit
CPU	Central Processing Unit
DE	Differential Evolution
FE	Function Evaluations
GRNN	Generalized Regression Neural Network
LHS	Latin Hypercube Sampling
MAE	Mean Absolute Error
MBH	Monotonic Basin Hopping
mjd2000	Modified Julian Date 2000
NLP	Nonlinear Programming
RAM	Random Access Memory
RBF	Radial Basis Function
RBFN	Radial Basis Function Network
RBFND	Radial Basis Function Network with Derivatives
SA	Simulated Annealing
SPICE	Spacecraft, Planet, Instrument, “C-matrix”, Events
STD	Standard Deviation
ToF	Time of Flight
Tudat	TU Delft Astrodynamics Toolbox

Symbols

CR	crossover ratio	-
D	amount of dimensions	-
E	expected value	multiple units
f	general function or fitness function	multiple units
F	scaling factor	-
g	probability density function	multiple units^{-1}

g_0	standard gravity	m s^{-2}
H	Hessian matrix	multiple units
$\mathcal{H}$	statistical hypothesis	-
I_{sp}	spacecraft specific impulse	s
k	number of cross-validation folds	-
l	RBFN basis function coefficient	-
$\mathbf{l}$	RBFN basis function coefficient vector	-
m	spacecraft mass	kg
N	amount of	-
p	RBFN polynomial coefficient	-
$\mathbf{p}$	RBFN polynomial coefficient vector	-
P	population	-
r_1, r_2, r_3	random integers	-
R	sum of the ranks in a population	-
$\mathbf{S}$	spacecraft state vector	m, m s^{-1}
t	time	s
T	spacecraft thrust magnitude	N
$\mathbf{u}$	mutant vector	multiple units
U	Wilcoxon rank-sum test statistic	-
V	velocity magnitude	m s^{-1}
$\mathbf{V}$	velocity vector	m s^{-1}
`viol`	constraint violation vector	multiple units
$\mathbf{x}$	decision vector	multiple units
$\mathbf{w}$	vector random variable	multiple units
$\mathbf{W}$	observation of the random variable $\mathbf{w}$	multiple units
z	scalar random variable	multiple units
Z	observation of the random variable z	multiple units
Δr	magnitude of the difference in spacecraft position	m
Δv	magnitude of the difference in spacecraft velocity	m s^{-1}
$\mathbf{\Delta V}$	vector of the change in velocity	m s^{-1}
ΔV	magnitude of the change in velocity	m s^{-1}
μ	gravitational parameter	m^3 s^{-2}
∇	gradient	multiple units
ϕ	basis function	-

$\mathbf{\Phi}$	distance matrix	-
φ	polar coordinate	rad
σ	smoothing parameter	-
τ	throttle	-
θ	azimuthal coordinate	rad

Common Subscripts

arr	arrival
c	center points
C	cubic
`C++`	generated with `tudat-learn`
dep	departure
G	Gaussian
i	index
I	impulsive shots
in	input
j	index
k	index
lb	lower bound
`matrix`	implementation with matrix multiplications
max	maximum
mb	match point from backward propagation
mf	match point from forward propagation
`nested`	implementation with nested for-loops
out	output
P	population individuals
`python`	generated with `python`
s	samples
ub	upper bound
v	validation samples
x, y, z	Cartesian components
∞	orbit insertion

Common Superscripts

T	transposition
-	infinitesimally before
+	infinitesimally after
$\hat{}$	approximation

1

Introduction

Low-thrust propulsion systems have grown in popularity since their first use in 1998, during the Deep Space 1 mission [1]. This is evidenced by their presence in other missions such as Hayabusa [2], Dawn [3], and BepiColombo [4], as well as the upcoming DESTINY$^+$ [5]. The rocket engines used for low-thrust propulsion are attractive due to their efficiency in terms of propellant mass, as electric propulsion systems can have a specific impulse of around 10 times that of a traditional chemical propulsion system.

However, in comparison to chemical propulsion, designing a low-thrust trajectory brings more challenges. In order to achieve the required ΔV for a certain trajectory, the spacecraft thrusters have to be operated in an almost continuous fashion, due to the low magnitude of the thrust that they produce. Three common approaches to the design of a low-thrust trajectory are shape-based, indirect, and direct methods. Shape-based methods, such as hodographic-shaping [6] or exponential sinusoids [7], revolve around assuming that the trajectory respects a certain shape that can be described in a fully analytical manner, and guaranteeing that it respects a set of boundary conditions. Indirect methods are based on optimal control theory, using calculus of variations to determine the inputs to a dynamical system that optimize a certain performance index while satisfying designated constraints on the motion of said system [8]. Finally, direct methods take the low-thrust trajectory optimization problem and parametrize it, using Nonlinear Programming (NLP) [9], to optimize one or more objective functions subject to a set of constraints, through the adjustment of a set of variables.

This work was conducted using a direct method, in particular, the Sims-Flanagan model [10], which is frequently used in literature [11–18] as a medium-fidelity approach. This model divides a low-thrust trajectory leg in segments of equal duration, and approximates the low-thrust propulsion system through the application of impulsive manoeuvres halfway through each segment. Orbital elements are computed after each impulsive shot and the state of the spacecraft is propagated in the respective Keplerian orbit. This transcription results in a nonlinear optimization problem which can be solved using a local optimization algorithm, such as the Sparse Nonlinear Optimizer (SNOPT), provided that the algorithm is given a good initial guess. Such initial guesses can be found by searching the decision space using a metaheuristic with a global scope, such as an evolutionary algorithm or Monotonic Basin Hopping (MBH), which is frequently used in literature in combination with the Sims-Flanagan transcription and SNOPT.

However, the search over the whole decision space often involves a large computational cost, associated with the very large number of fitness function evaluations that need to be computed. Various attempts at reducing this computational burden have been made, mostly through the approximation of said fitness computations. Ampatzis and Izzo [19] use an Artificial Neural Network (ANN) alongside the original trajectory model during the optimization process without degrading the quality of the final trajectory. Li et al. [20] utilize ANNs to estimate optimal transfer costs of transfer times, fuel consumption and total ΔV of low-thrust trajectories. More examples of using ANNs and other machine learning algorithms can be found in literature [21–24]. However, many of these works involve the creation of a dataset prior to the optimization process, which often involves a large computational effort.

Recent domain-agnostic literature [25–31] has focused on surrogate-assisted optimization as a way to reduce the computational burden associated with expensive optimization processes. These surrogate models often comprise computationally cheap regression or interpolation functions that attempt to

approximate the fitness function of the original problem. The approximations can then be used during the optimization process to make a more informed decision on which decision vectors to evaluate with the exact function. Moreover, it is common to do so in a fully online manner, that is, by not spending computational resources on generating datasets in advance and instead building a dataset out of the fitness evaluations that nevertheless need to be computed during the optimization process. This project follows the work of Wang et al. [27] in that it uses Generalized Regression Neural Networks (GRNN) and Radial Basis Function Networks (RBFN) to build surrogate models for global and local search phases, respectively. The GRNN was chosen as it can efficiently be fitted to large datasets and can, in theory, converge to any underlying regression surface. Finally, the choice of the RBFN was due to how well it scales for inputs of large dimensions, according to the study by Díaz-Manríquez et al. [32], which is very much necessary with the Sims-Flanagan model, as decision spaces ranging from around 30 to 90 dimensions were studied in the current work.

1.1. Research Objective and Question

Based on the knowledge gathered while scouring through literature, a research gap was identified. As it was mentioned in the previous section, this gap mainly concerns the usage of online surrogate models for trajectory optimization in the context of preliminary mission design, or lack thereof. In an attempt to fill that gap, a **research objective** was delineated:

- *To improve the optimization of interplanetary, low-thrust trajectories based on the Sims-Flanagan transcription through the use of online surrogate models.*

To guide the process of reaching the objective, a **research question** has been formulated:

- *How can online GRNN and RBFN surrogates improve the optimization of interplanetary, low-thrust trajectories based on the Sims-Flanagan model?*

The main question sprouts a number of **secondary questions** that complement it and that make it easier to gradually achieve the aforementioned objective:

- *How can evolutionary and local optimization algorithms be used in combination with online surrogates to perform constrained, single-objective, high-dimensional optimization of interplanetary, low-thrust trajectories?*
- *Can a surrogate-assisted optimization approach lead to better results than the monotonic basin hopping techniques used in literature?*
- *How does each component of a surrogate model contribute to the optimization of interplanetary, low-thrust trajectories?*

1.2. Outline of the Report

The remainder of this report is composed of three chapters and four appendices. Chapter 2 contains the bulk and main findings of the research that was conducted in the form of a paper manuscript. Limitations about some aspects of said research can be found in Chapter 3. Chapter 4 includes extended conclusions and recommendations for future work. Relevant information about the methods that were used during the research process and that did not make the paper manuscript can be found in Appendix A. Said appendix contains information on GRNNs, first- and second-order derivatives of the RBFN model, and small descriptions of cross-validation, latin hypercube sampling, monotonic basin hopping, the Wilcoxon rank-sum test, and how some of the plots were generated. Appendices B and C contain the verification and validation of the software tools and models that were used during this project. Finally, Appendix D contains class diagrams of the tool that was developed for the project, `tudat-learn`, which implements machine learning tooling, such as estimators, datasets, scalers, and samplers.

2

Paper

The paper presented in the coming pages is written in the format specified for the Astrodynamics Specialist Conference, hosted by the American Astronautical Society (AAS) and cohosted by the American Institute of Aeronautics and Astronautics (AIAA).

(Preprint) AAS XX-XXX

ONLINE SURROGATE MODELS FOR THE CONSTRAINED OPTIMIZATION OF INTERPLANETARY LOW-THRUST TRAJECTORIES

Francisco Castanheira* **and Kevin Cowan†**

The optimization of interplanetary, low-thrust trajectories is a computationally expensive aspect of preliminary mission design. To reduce the computational burden associated with it, surrogate models can be used as cheap approximations of the original fitness function. Training the surrogate models in a fully online manner can be done to remove the need of having previously generated datasets, which is another source of computational cost. The Sims-Flanagan transcription is used to model an Earth-Mars transfer which is optimized through different optimization routines. The development of a C++ library with machine learning tooling was initiated, containing implementations for generalized regression neural networks and radial basis function networks that are used in global and local surrogates, respectively, having their hyperparameters tuned through cross-validation. Surrogate models were constructed with differential evolution operators and derivative information. However, the best-performing model did not use a local search phase and incorporated a surrogate in a monotonic basin hopping optimization strategy which, by itself, is the most common approach in literature. A Wilcoxon rank-sum test helped confirming that the aforementioned surrogate model performed the best, having its median solution outperform the median solutions from every other model by, at least, approximately 400 m s^{-1}, in terms of total ΔV. Despite not having conducted a full study of the computational effort due to the simulations having been run in a server with a variable load, the best-performing model presents the best results for similar amounts of fitness function evaluations. It was concluded that a surrogate model can be used to improve the optimization of low-thrust trajectories modeled with the Sims-Flanagan transcription when inserted in a monotonic basin hopping optimization scheme.

INTRODUCTION

In recent years, low-thrust propulsion systems such as the ones that use electric propulsion have increased in popularity, as evidenced by past and upcoming interplanetary missions.[1–3] Having a specific impulse of around 10 times that of a chemical propulsion system, electric propulsion can be more efficient in terms of propellant mass. However, as the name indicates, these propulsion systems are only able to deliver low amounts of thrust, which results in the need for long thrusting times to achieve high exhaust velocities. This leads to additional challenges when designing a trajectory.

When modelling a low-thrust trajectory, the parametrization introduced by Sims and Flanagan[4], hereinafter designated as the Sims-Flanagan model, is a common choice in literature for preliminary mission design.[5–8] Being a direct method, it poses an optimization problem that can be solved by local optimization algorithms, one of them being the Sparse Nonlinear Optimizer (SNOPT), which

*MSc Student, Faculty of Aerospace Engineering, Delft University of Technology, The Netherands
†Education fellow + Lecturer, Faculty of Aerospace Engineering, Delft University of Technology, The Netherands, k.j.cowan@tudelft.nl

is recommended by Sims and Flanagan[4] and widely used in literature. However, as the decision space grows, these local methods tend to converge to suboptimal solutions or even not converge if one fails to provide them with a good initial guess.

Being able to explore a large decision space is crucial in the context of preliminary mission design, as multiple launch windows can be more easily examined and compared. When using the Sims-Flanagan method, multiple global search strategies can be found in literature. Yam et al.[5] compare the performance of Differential Evolution (DE) and Simulated Annealing (SA) on an Earth-Mars rendezvous mission, while Yam et al.[6] also compare SA but with Monotonic Basin Hopping (MBH) and multistart strategies. Both of these works use SNOPT as a local search algorithm at some point in the optimization process. Ellison et al.[7] point out that the MBH and SNOPT perform the best, both in terms of the likelihood of finding a feasible solution and in terms of the likelihood of converging to the optimal solution. Ulibarrena and Cowan[8] also make use of MBH but employ sequential least-squares programming in the local search phase.

Despite the Sims-Flanagan method resulting in computationally cheap fitness evaluations, that is, the evaluation of the objective and constraint functions, getting it to converge requires the fitness to be computed a large amount of times. When accounting for wanting to explore a large decision space, the computational cost of the optimization process escalates rapidly. Recent domain-agnostic literature has focused on surrogate-assisted optimization as a way to reduce the computational burden associated with expensive optimization processes.[9–13] These surrogate models often comprise computationally cheap regression or interpolation functions that attempt to approximate the fitness function of the original problem. Even though it is possible to use an existing dataset to train the surrogate models with, in literature it is common to start with an empty dataset and add more decision vectors and the respective objective and constraint values to it as the optimization progresses, in an online fashion. This reduces the cost of creating an initial dataset and encourages the use of surrogates that have fast training processes.

Lim et al.[9] mention that the surrogate models have two major goals: first, to mitigate the *curse of uncertainty*, and second, to benefit from the *blessing of uncertainty*. The former refers to the negative consequences that are introduced by the approximation error associated with the chosen models, while the latter alludes to the benefits achieved by smoothing rugged fitness landscapes to prevent the search from getting stuck in local optima. To attain these goals, Wang et al.[10] use a Generalized Regression Neural Network (GRNN) to construct a global surrogate, and a Radial Basis Function Network (RBFN) to construct a local surrogate. The GRNN is a regression model which does not model the target function in an exact manner but gets as close as possible to the training data on average, which aims to benefit from the blessing of uncertainty by smoothing out the fitness landscape. Contrasting with the GRNN, the RBFN is an interpolation model and passes through each instance of the training data, being able to approximate the fitness landscape more accurately and alleviating the curse of uncertainty. As it will be seen, this paper adopts a very similar approach.

The work of Wang et al.[10] is particularly interesting due to the fact that it deals with surrogate-assisted optimization of single-objective problems with inequality constraints, something that can be applied to the Sims-Flanagan problem. It does so by employing a global search using DE mutation and crossover operators on a global surrogate and a local search that performs local optimization on a local surrogate. Miranda-Varela and Mezura-Montes[11] performed a study on the performance on different constraint-handling techniques when used to compare candidate solutions in surrogate-assisted optimization of problems with equality and inequality constraints. Other interesting work, albeit regarding unconstrained optimization includes the paper by Chen et al.,[12] which employs a local and a global surrogate alongside a DE algorithm to optimize high-dimensional problems with decision spaces ranging from 20 to 100 dimensions. Finally, Liu et al.[13] developed a novel surrogate-assisted evolutionary algorithm with an uncertainty grouping based criterion to select which of the candidate vectors generated by the surrogate model to evaluate with the exact function, providing yet another tool for the optimization not to get stuck on local optima.

The goal of this work is to improve the preliminary design of interplanetary, low-thrust trajectories by reducing the computational effort associated with their optimization, through the use of online surrogate models. The key idea is to build a dataset of fitness evaluations during the optimization process and using it to train estimators to predict the exact fitness values at a lower computational cost. It should be noted that these *exact* fitness values correspond to evaluations of the fitness function yielded by the Sims-Flanagan problem, which the surrogates attempt to approximate. GRNN and RBFN surrogates are used for global and local searches, respectively, with the latter being able to provide an estimate of the first-order derivatives of the original function, which can be used by the local optimization algorithm, SNOPT. Naturally, the Sims-Flanagan model is used to describe the low-thrust trajectories.

Various tools have been used leading up to this paper. The original astrodynamics models and tooling have been taken from TU Delft Astrodynamics Toolbox* (`Tudat`) and later modified to the existing needs. The same has been done with the DE algorithm, which was taken from `pagmo`, a parallel global multiobjective framework for optimization, developed by the Advanced Concepts Team of the European Space Agency.[14] An MBH algorithm as well as the interface for SNOPT† and the whole optimization framework were also taken from `pagmo` but used without modifications. At last, development of a tool aimed at bringing machine learning tools and techniques to `Tudat` was started. The tool was named `tudat-learn`‡, was written entirely in C++ with maintainability, modularity, and scalability in mind, that, at the time of writing contains implementations of RBFNs, GRNNs, datasets, cross-validation procedures, and other machine-learning related tools.

The remainder of this paper is structured as follows: First, the trajectory model used to represent the low-thrust transfer is presented. Then, the surrogate model is introduced, followed by the optimization approach. Finally, the results obtained during the optimization process are exposed and examined, with conclusions being drawn.

TRAJECTORY MODEL

Even though there are various options when it comes to choosing a model for an interplanetary low-thrust trajectory, the Sims-Flanagan model[4] has been shown to be robust within the context of preliminary mission design.[7,15] An implementation of this model that is common in literature[5,6] exists in `Tudat` and was used in this work, being presented below and illustrated in Figure 1. It is also important to mention that `ECLIPJ2000` was the reference frame used in this work, which is centered in the Solar System barycenter and whose xy-plane is aligned with the Earth mean ecliptic and equinox of the epoch J2000.

The trajectory is divided into legs that begin and end with a control node, which are usually associated with planets or small bodies, but can simply be free points in space. Low-thrust arcs on each leg are modelled as sequences of impulsive manoeuvres, $\mathbf{\Delta V}_i$, connected by conic section arcs, where i indicates the i^{th} impulsive manoeuvre. Each leg is divided in N_I segments of equal duration, with the impulsive shots being applied halfway through the respective segment, time-wise. The magnitude of the $\mathbf{\Delta V}_i$ at each segment should not exceed a maximum magnitude, $\Delta V_{\max,i}$, which corresponds to the velocity change accumulated by the spacecraft when it is operated at full thrust throughout that segment:

$$\Delta V_{\max,i} = \frac{T_{\max}}{m_{i-1}} \frac{t_{\text{arr}} - t_{\text{dep}}}{N_I}, \tag{1}$$

where $T_{\max}$ denotes the maximum thrust of the low-thrust engine, and t_{dep} and t_{arr} denote the times at departure from and arrival at the control nodes, respectively. Moreover, m_{i-1} indicates the mass

*`https://github.com/tudat-team/tudat-bundle` (last accessed on 10 May 2022)
†SNOPT was used with an academic license provided by the University of California, San Diego.
‡`https://github.com/tudat-team/tudat-learn` (last accessed on 10 May 2022)

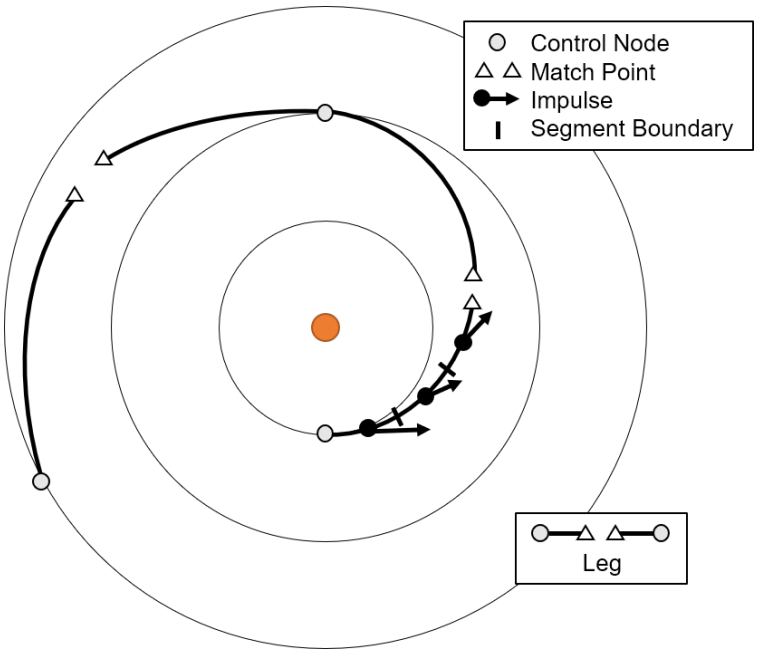


Figure 1. Impulsive ΔV transcription of a low-thrust trajectory, according to the Sims-Flanagan model.[4]

of the spacecraft after the $(i-1)^{\text{th}}$ impulsive shot, with the initial mass being the mass at departure, $m_0 = m_{\text{dep}}$.

After every impulsive shot, the mass of the spacecraft is propagated using Tsiolkovsky's rocket equation:[16]

$$m_{i+1} = m_i \exp\left(-\frac{\Delta V_i}{g_0 I_{\text{sp}}}\right), \tag{2}$$

where $g_0 = 9.80665$ m s^{-2} denotes the acceleration due to Earth's gravity at sea level and I_{sp} corresponds to the specific impulse of the low-thrust engine.

These impulsive shots are applied within a duration assumed to be negligible and naturally cause a change in the state of the spacecraft. Let $\boldsymbol{\tau}_i$ denote the throttle vector of the i^{th} impulsive shot, with a magnitude τ_i between 0 and 1. Using the azimuthal and polar directions of the throttle, θ_i and φ_i, respectively, the throttle vector in the Cartesian reference frame can be computed as shown below:

$$\boldsymbol{\tau}_i = \tau_i \begin{pmatrix} \cos\theta_i \sin\varphi_i \\ \sin\theta_i \sin\varphi_i \\ \cos\varphi_i \end{pmatrix}, \tag{3}$$

which can then be used to compute the ΔV_i, which is used alongside the velocity before the impulsive shot, $\mathbf{V}_i^-$ to compute the velocity after it is applied, $\mathbf{V}_i^+$:

$$\boldsymbol{\Delta V}_i = \frac{T_{\max}}{m_{i-1}} \frac{t_{\text{arr}} - t_{\text{dep}}}{N_I} \boldsymbol{\tau}_i, \qquad \mathbf{V}_i^+ = \mathbf{V}_i^- + \boldsymbol{\Delta V}_i. \tag{4}$$

With the expression on the left, it is possible to compute the changes in velocity associated with every impulsive shot before any propagation, by interleaving the computation of said expression with Equation (2). The former uses the mass of the spacecraft to compute a change in velocity, while the latter uses that change to compute the new mass of the spacecraft.

In the context of preliminary mission design, it is common to evaluate the quality of a trajectory by examining the amount of propellant it utilizes. Naturally, lower propellant quantities can lead to greater science returns, for instance by making it possible for the spacecraft to carry more instruments. Quantifying the propellant usage is often done with the final mass of the spacecraft or with the total ΔV used for the trajectory, which is the approach taken in this work. To compute it, the ΔV_i for each of the impulsive shots are added up, as done below:

$$\Delta V = \sum_{i=1}^{N_I} \Delta V_i = \sum_{i=1}^{N_I} \frac{T_{\max}}{m_{i-1}} \frac{t_{\text{arr}} - t_{\text{dep}}}{N_I} \tau_i. \tag{5}$$

At a single leg, the trajectory is propagated forwards from the departure control node and backwards from the arrival control node, with half of the total impulsive shots being applied during each of those half-legs. After the propagation, the half-legs should meet at the match point or the state mismatch at the match point, shown below, should be less than a tolerance to have a feasible trajectory:

$$\mathbf{S_{mf}} - \mathbf{S_{mb}} = \left[\Delta r_x, \Delta r_y, \Delta r_z, \Delta v_x, \Delta v_y, \Delta v_z\right], \tag{6}$$

with $\mathbf{S_{mf}}$ and $\mathbf{S_{mb}}$ corresponding to the spacecraft state vectors at the match point, originated by the forwards and backwards propagations, respectively. Additionally, r denotes the position of the spacecraft, v denotes its velocity, and the subscripts x, y, and z indicate the corresponding Cartesian component. It is also important to note that, in the backwards propagated half leg, the right-hand side expression in Equation (4) has to be rearranged and $\mathbf{V}_i^+$ is used to compute $\mathbf{V}_i^-$, as one is going from an instant in time after the impulsive shot to the one before it happens.

In addition to using Sims-Flanagan as a model for the low-thrust trajectory, it is necessary to describe the orbit insertion and rendezvous processes. The medium fidelity nature of Sims-Flanagan is coherent with the assumption that the spheres of influence of the departure and arrival bodies are negligible, as their size is small in comparison to the scale of interplanetary trajectories. Therefore, the departure and arrival positions of the spacecraft are assumed to coincide with the center of mass of the departure and arrival bodies, respectively. Regarding velocities, a rendezvous is required at arrival and despite it being possible to make a small velocity correction when that happens, it was decided to set the spacecraft to arrive at the body of choice with no difference in velocity to it, for simplicity. However, the same does not happen in the departure, due to the orbit insertion process. It is assumed that the burn time of the launch vehicle is short, resulting in the approximation of it being applied as an impulsive shot in the departure position. Let the launch velocity be denoted by V_∞, and its azimuthal and polar directions be denoted by θ_∞ and φ_∞, respectively. The velocity of the spacecraft at departure, $\mathbf{V_{dep}}$, can be computed in the Cartesian reference frame through the expressions that follow:

$$\mathbf{V}_\infty = V_\infty \begin{pmatrix} \cos\theta_\infty \sin\varphi_\infty \\ \sin\theta_\infty \sin\varphi_\infty \\ \cos\varphi_\infty \end{pmatrix}, \qquad \mathbf{V_{dep}} = \mathbf{V_p} + \mathbf{V}_\infty, \tag{7}$$

with $\mathbf{V}_\infty$ being the vector of the launch velocity and $\mathbf{V_p}$ the velocity of the planet at departure.

Modeling the Low-Thrust Leg as an Optimization Problem

The Sims-Flanagan method originates a single-objective, box-bounded optimization problem subject to six equality constraints. A concrete enunciation of the problem is presented below:

$$\underset{\mathbf{x}}{\text{minimize}} \quad \Delta V(\mathbf{x}), \tag{8}$$

$$\text{subject to} \quad \left(\mathbf{S_{mf}}(\mathbf{x}) - \mathbf{S_{mb}}(\mathbf{x})\right)_j = 0, \quad j = 1, \ldots, 6, \tag{9}$$

$$\mathbf{x_{lb}} \leq \mathbf{x} \leq \mathbf{x_{ub}}, \tag{10}$$

where j designates the different individual components of the state mismatch, $\mathbf{x}$ denotes the decision vector, and $\mathbf{x_{lb}}$ and $\mathbf{x_{ub}}$ correspond to its upper and lower bounds.

In this work, the decision vector and its bounds correspond to the following set of variables:

- the departure epoch, t_{dep},
- the time of flight, ToF,
- the orbit insertion velocity, V_∞,
- the azimuthal direction of the orbit insertion velocity, θ_∞,
- the polar direction of the orbit insertion velocity, φ_∞,
- the throttle magnitudes at each segment, τ_i,
- the azimuthal direction of the throttles at each segment, θ_i,
- and the polar direction of the throttles at each segment, φ_i,

which adds up to $(5 + 3 \times N_I)$ decision variables.

Various works in literature define the throttle vectors as x, y, and z components that vary between -1 and 1, highlighting the percentage of the maximum thrust applied in that specific direction.[5–7] However, when multiple components approach a unit value for a particular impulsive shot, its norm can be greater than one, which makes it larger than the $\Delta V_{\text{max},i}$, from Equation (1). This results in having to introduce N_I inequality constraints on the norm of the throttle vectors, adding further complexity to the optimization problem. This paper follows the approach taken by Ulibarrena and Cowan,[8] who prevent it by defining the throttle vectors as a magnitude and two directions, encoding the limitation on the throttle magnitude in the bounds of the problem.

The Sims-Flanagan model is often paired with a nonlinear programming (NLP) problem solver, with SNOPT being the most common choice in literature.[5–7] When using SNOPT and a suitable initial guess, the Sims-Flanagan model is fast and robust, being a common choice for a medium fidelity model. Among SNOPT's various settings[17], perhaps the most relevant is the "Major feasibility tolerance", ϵ, that dictates which solutions SNOPT considers to be feasible:

$$\underset{j}{\text{maximum}} \quad \frac{\texttt{viol}_j}{||\mathbf{x}||} \leq \epsilon, \tag{11}$$

where $\texttt{viol}_j$ is the violation of the j^{th} constraint and $||\mathbf{x}||$ the Euclidean norm of the decision vector. In literature, various values can be found for ϵ, Yam et al.[5] use 10^{-6}, Ellison et al.[7] use 10^{-5}, and Ozimek et al.[18] use 10^{-8} for a generalized version of the Sims-Flanagan model. Ellison et al.[7] also mention that the likelihood of SNOPT both finding a feasible solution and converging to the optimal solution is best when the decision variables, objective function, and constraints are scaled to the same order of magnitude. Following that directive, the decision variables are scaled between 0 and 1, the position mismatch is scaled with a characteristic length unit, which can be the Astronomical Unit (AU) in the case of an Earth-Mars transfer, and the ΔV as well as the velocity mismatch are scaled with a characteristic velocity unit, $\sqrt{\mu/\text{AU}}$, where μ is the gravitational parameter of the central body.

Finally, regarding the number of impulsive shots, Englander and Conway[19] mention that the larger values result in more realistic estimates but at a high computational cost, as SNOPT estimates the gradient of the function it is optimizing, which grows quadratically with the dimension of the problem. The authors mention that it is common to have at least 10 impulsive shots per revolution about the central body.

SURROGATE MODELS

While there are many regression and interpolation functions that a surrogate can be built from, this paper follows the work of Wang et al.[10] in the use of GRNNs and RBFNs to construct global and local surrogates, respectively.

Generalized Regression Neural Network

Proposed by Specht,[20] the GRNN is a memory-based neural network that provides estimates of continuous variables and can, in theory, converge to any underlying regression surface. The GRNN is a one-pass learning algorithm, as opposed to other neural networks that commonly use backpropagation, which results in a cheap approximation that is suitable for dealing with expensive optimization problems. The properties of GRNNs are perfect to have it as the global surrogate, smoothing out the fitness landscape and benefiting from the blessing of uncertainty. Let f correspond to the function being approximated, $\mathbf{x} = \{x_1 \dots \ x_{D_{\text{in}}}\}^T$ to a decision vector of dimension D_{in}, and $\mathbf{c}_i = \{c_{i1} \dots \ c_{iD_{\text{in}}}\}^T$ to the N_{c} decision vectors or center points that the GRNN is fitted to, with $i = 1, \dots, N_{\text{c}}$. The GRNN computes the approximation of f, $\hat{f}$, as follows:

$$\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{N_{\text{c}}} f(\mathbf{c}_i) \cdot \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma^2}\right)}{\sum_{i=1}^{N_{\text{c}}} \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma^2}\right)}, \tag{12}$$

where $\|\cdot\|$ denotes the standard Euclidean norm. Naturally, f represents any of the objective or constraint functions that are being approximated, and a separate approximation must be done for each of those functions.

Radial Basis Function Network

The RBFN is an interpolation model that has been applied to different scientific and engineering fields, and it was chosen to construct the local surrogates during the optimization process. The study conducted by Díaz-Manríquez et al.[21] compared the performance and efficiency of multiple metamodeling techniques. The study concluded with RBFN being the most robust model, especially as the number of input dimensions increased. Even though the study only examined problems up to 50 dimensions, which can quickly fall short as the number of impulsive shots chosen for Sims-Flanagan increases, it made the RBFN the most sensible choice.

Using the notation introduced for the GRNN, the RBFN approximates a function f as follows:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{N_{\text{s}}} l_i \phi(\|\mathbf{x} - \mathbf{c_i}\|) + p_0 + \sum_{k=1}^{D\text{in}} p_k x_k \tag{13}$$

where ϕ is a basis function that takes the Euclidean distance between the input vector and a center point as its own input. Furthermore, l_i, p_0, and p_k are coefficients determined during the training process, with the latter two being zero- and first-order polynomial coefficients, respectively. Even though it is common to use RBFNs without polynomial coefficients, Flyer et al.[22] state that their presence tends to improve the accuracy of the approximation at domain boundaries and the accuracy of derivative approximations through the whole domain. Similarly to the GRNN, Equation (13) approximates a single objective or constraint. Therefore, a different set of $\mathbf{l}$ and $\mathbf{p}$ coefficients has to be computed for each objective and constraint. The coefficients can be obtained by solving the linear system below:

$$\left[\begin{array}{ccc|cccc} & & & 1 & c_{11} & \cdots & c_{1D_{\text{in}}} \\ & \mathbf{\Phi} & & \vdots & \vdots & \vdots & \vdots \\ & & & 1 & c_{N_{\text{c}}1} & \cdots & c_{N_{\text{c}}D_{\text{in}}} \\ - & - & - & - & - & - & - \\ 1 & \cdots & 1 & & & & \\ c_{11} & \cdots & c_{N_{\text{c}}1} & & \mathbf{0} & & \\ \vdots & \cdots & \vdots & & & & \\ c_{1D_{\text{in}}} & \cdots & c_{N_{\text{c}}D_{\text{in}}} & & & & \end{array}\right] \cdot \begin{bmatrix} l_1 \\ \vdots \\ l_{N_{\text{c}}} \\ - \\ p_0 \\ p_1 \\ \vdots \\ p_{D_{\text{in}}} \end{bmatrix} = \begin{bmatrix} f(\mathbf{c}_1) \\ \vdots \\ f(\mathbf{c}_{N_{\text{c}}}) \\ - \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{14}$$

where $\mathbf{0}_{(D_{\text{in}}+1)\times(D_{\text{in}}+1)}$ is a null matrix and $\mathbf{\Phi}$, the distance matrix, is computed as follows:

$$\mathbf{\Phi} = \begin{bmatrix} \phi(\|\mathbf{c}_1 - \mathbf{c}_1\|) & \phi(\|\mathbf{c}_1 - \mathbf{c}_2\|) & \cdots & \phi(\|\mathbf{c}_1 - \mathbf{c}_{N_c}\|) \\ \phi(\|\mathbf{c}_2 - \mathbf{c}_1\|) & \phi(\|\mathbf{c}_2 - \mathbf{c}_2\|) & \cdots & \phi(\|\mathbf{c}_2 - \mathbf{c}_{N_c}\|) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\|\mathbf{c}_{N_c} - \mathbf{c}_1\|) & \phi(\|\mathbf{c}_{N_c} - \mathbf{c}_2\|) & \cdots & \phi(\|\mathbf{c}_{N_c} - \mathbf{c}_{N_c}\|) \end{bmatrix}. \tag{15}$$

Multiple forms of basis functions can be used with RBFNs. In this paper, both the cubic form $\phi_C(\|\mathbf{x} - \mathbf{c}_i\|) = \|\mathbf{x} - \mathbf{c}_i\|^3$ and the Gaussian form $\phi_G(\|\mathbf{x} - \mathbf{c}_i\|) = \exp(-\|\mathbf{x} - \mathbf{c}_i\|^2/(2\sigma^2))$ are used as they are common in surrogate-assisted optimization literature.[10,12,13]

This model also has the advantage of there being the possibility to compute its derivatives analytically, which can help approximate the derivatives of the original function, an aspect that is particularly useful when the latter are not available. Various local solvers, such as SNOPT require gradient information and can approximate the gradient of the function being optimized via finite differentiation. However, if SNOPT is used to perform a local search on a surrogate model, it may be beneficial to provide the analytical derivatives of said surrogate. Regarding the Sims-Flanagan model, Ellison et al.[7] derive analytical expressions for many entries of the problem gradient resulting in improved performance. In this work, the analytical derivatives of the Sims-Flanagan model are not used, but the ones of the RBFN surrogate models are provided to SNOPT when performing a local search on the surrogate.

The gradient $\nabla\hat{f}$ of the function $\hat{f}$ at the point $\mathbf{x}$ is defined below:

$$\nabla\hat{f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial\hat{f}}{\partial x_1} & \cdots & \frac{\partial\hat{f}}{\partial x_{D_{\text{in}}}} \end{bmatrix}^T, \tag{16}$$

with each of the partial derivatives being given by

$$\frac{\partial\hat{f}}{\partial x_j}(\mathbf{x}) = \sum_{i=1}^{N_c} l_i \frac{\partial\phi}{\partial x_j}(\|\mathbf{x} - \mathbf{c}_i\|) + p_j. \tag{17}$$

Taking into account that

$$\|\mathbf{x} - \mathbf{c}_i\| = \left[(x_1 - c_{i1})^2 + ... + (x_D - c_{iD_I})^2\right]^{\frac{1}{2}}, \tag{18}$$

the first-order partial derivatives of the cubic and Gaussian basis functions can be computed as follows:

$$\frac{\partial\phi_C}{\partial x_j}(\|\mathbf{x} - \mathbf{c}_i\|) = 3 \cdot (x_j - c_{ij}) \cdot \|\mathbf{x} - \mathbf{c}_i\|. \tag{19}$$

$$\frac{\partial\phi_G}{\partial x_j}(\|\mathbf{x} - \mathbf{c}_i\|) = \phi_G(\|\mathbf{x} - \mathbf{c}_i\|) \cdot \left[-\frac{(x_j - c_{ij})}{\sigma^2}\right], \tag{20}$$

A derivation for equations (17), (19), and (20) can be found in Appendix A.

Some Observations on the Surrogate Models

As mentioned, the global surrogate will be constructed out of a GRNN, while the local surrogate will comprise an RBFN. Each surrogate is composed of seven GRNNs or RBFNs, one which approximates the objective and six that approximate each of the constraints. From here onwards, using the terms GRNN model or RBFN model will refer to that set of seven separate approximations. Furthermore, **RBFND** will be used to refer to an RBFN surrogate model in which the first-order derivatives are implemented, and **RBFN** will indicate that said derivatives are not available in the model and that the input vector is slightly different, which will be explained in the coming paragraphs.

In terms of what constitutes the center points that the models are fitted to, RBFND is the simplest. It uses the same variables as the optimization problem derived from the Sims-Flanagan model and given to SNOPT: A decision vector consisting of departure epoch, time of flight, orbit insertion velocity (magnitude and directions), and the throttle variables for each of the impulsive shots, all scaled between 0 and 1, according to the problem bounds. Both RBFND and the other surrogate models attempt to approximate the objective and constraint values scaled in the same way as provided to SNOPT. It is common for many regression and interpolation algorithms to perform best when the domain and the values they are trying to approximate all lie within the same ranges. Here is no different, as both models depend on the Euclidean distance between some center points and input decision vectors. Having variables within different ranges contributing to the computation of an Euclidean distance can easily lead to one of them dominating that computation due to differences in their orders of magnitude. That was verified in early simulations, where the departure date, given as a Modified Julian Date 2000 (mjd2000) in seconds and the time of flight, also given in seconds, were the main contributors to the Euclidean distance measured between any two decision vectors.

The GRNN and RBFN models are built with center points that are different than the ones used to train the RBFND model. Following the approach of Stubbig and Cowan[23] , the departure epoch and the orbit insertion velocity parameters are not used as input variables. Instead, the departure epoch and the time of flight are used to query the departure and arrival state of the departure and arrival bodies. Afterwards, the orbit insertion velocity is used to compute the departure state of the spacecraft, according to Equation (7). Finally, the departure and arrival states are also scaled between 0 and 1. The limits of this scaling procedure are computed by querying the departure and arrival states within the departure and arrival windows, at an approximate rate of 365×10^4 queries per year in those windows. The minimum (most negative) and maximum (most positive) values for each of the state variables are stored, with the minimum and maximum possible orbit insertion velocities being added to the respective state variables.

While it is important to reduce the size of the input vector as much as possible for the optimization process to avoid redundant inputs, the same is not necessarily true for the surrogate model. The surrogate model may benefit from having more physical information about the problem, i.e., departure and arrival states, while the optimization process cannot even perform searches directly on these variables, as they are dependent on others. Within the field of machine learning, this process of selecting and manipulating data into the right input variables, or features, is referred to as *feature engineering*.

The reason for not applying these changes in the input vectors for the RBFND model is that SNOPT requires the derivatives of the fitness vector with regard to each of the decision variables. The RBFN model described in the previous paragraphs could approximate the derivatives of the fitness vector with regard to the spacecraft's departure and arrival states, but not with regard to the departure epoch and the orbit insertion parameters, which is what SNOPT requires.

Hyperparameter Tuning

Hyperparameters are parameters of the estimator that are not directly learned during the training or fitting process. In this paper, the hyperparameters that will be tuned are the choice of basis function in the RBFN and RBFND models, the smoothing parameter σ if the Gaussian basis function is chosen as well as the one in the GRNN model, and finally, the size of the training dataset to which the RBFN and RBFND models are fitted. While the time it takes to construct a GRNN is a linear function of the size of the training dataset, regarding the RBFN and RBFND models, it is a function of the size of the training dataset cubed. Therefore, it is convenient to select an appropriate dataset size that balances accuracy and efficiency.

In order to make an informed selection of the hyperparameters, a cross-validation routine was ran for each of the three models. Datasets were generated by applying the description of the optimization

problem in Equations (8), (9), and (10) to an Earth-Mars rendezvous mission. Following the work of Yam et al.,[5] this paper runs the optimization with 10, 20, and 30 impulsive shots, with a dataset of 3000 instances being generated for each of those amounts, using Latin Hypercube Sampling (LHS). The actual portion of the datasets that is used during cross-validation starts at 200 instances and increases in increments of 100 up to the full 3000 instances. The size of the validation set is fixed at 100, resulting in a k-fold cross validation where $k =$ (size of dataset/100). The reason for this unconventional choice of a variable number of folds is due to one of the figures of merit chosen to evaluate the models with. Taken from the work of Stubbig and Cowan[23] , that figure of merit is the *sorting accuracy*. The sorting accuracy is defined using a "Top 25 out of 100" measure, meaning that it corresponds to the fraction of the 25 best validation instances that the model correctly predicts as belonging to that group. This figure of merit is derived from the model's application, which states that in order to guide a population-based algorithm, it is more important for the surrogate model to distinguish good from bad candidate solutions than to accurately predict the correct objective and constraint values.

It is, however, necessary to define how different validation instances are compared. Taking into account that a validation instance is a decision vector taken from the validation dataset and the respective respective objective and constraint values, the best out of two instances is selected according to the *feasibility rules*, enunciated below:

1. Any feasible solution is preferred over any infeasible one.
2. Among two feasible solutions, there is preference for the one with the best objective function value.
3. Among two infeasible solutions, there is preference for the one yielding a smaller constraint violation.

Equation (11) is used to decide whether or not a solution is feasible, and the maximum constraint violation is used as the criterion to compare the amount of constraint violation between multiple solutions.

Even though the main purpose of the surrogate models is to guide the optimization process towards better optima, it was deemed important to have a measure of accuracy of said models. The chosen measure is the Mean Absolute Error (MAE). For a validation set of size N_v and a D_out-dimensional fitness vector, MAE is defined as follows:

$$\mathrm{MAE} = \frac{1}{N_\mathrm{v}} \sum_{i=1}^{N_\mathrm{v}} \left(\frac{1}{D_\mathrm{out}} \sum_{j=1}^{D_\mathrm{out}} \left| f_{ij} - \hat{f}_{ij} \right| \right) \tag{21}$$

where f_{ij} and $\hat{f}_{ij}$ correspond to the j^{th} component of the exact and the approximated fitness vectors, respectively, of the i^{th} validation instance.

The cross-validation procedure was run for four values of $2\sigma^2$ (1, 2, 25, and 100) in addition to the cubic basis function for the RBFN and RBFND models. The results can be found in Appendix B. For the GRNN model, the choice is straightforward, with $2\sigma^2 = \frac{1}{2}$, as it is the value for which the model presents the highest sorting accuracy and lowest MAE for most training dataset sizes, as shown in Figures B.1 and B.2. With the RBFND model, a maximum size of 1500 samples was chosen, as it is the number for which the sorting accuracy is maximum, with 30 impulsive shots, as it can be seen in Figure B.3. In the same figure, it can be argued that $2\sigma^2 = 2$ either outperforms or has the joint best performance for every amount of impulsive shots, resulting in its choice for the RBFND model. The same value shows better performance in the RBFN model for 10 impulsive shots, as shown in Figure B.5. However, for 20 and 30 impulsive shots, the cubic basis function was chosen for this model, as it seems to have the best and joint best performance for 20 and 30 impulsive shots, respectively, in terms of sorting accuracy. The mean absolute error for 20 and 30 impulsive shots was not considered for the choice of hyperparameters due to most of the basis functions and dataset sizes yielding similar error values, as shown in Figures B.4 and B.6.

OPTIMIZATION APPROACH

In this paper, both population-independent and population-based approaches are used, as the former is usually found in research concerning the use of the Sims-Flanagan model, and the latter is common within surrogate-assisted optimization literature.

Population-Independent Approach

To serve as a benchmark for the results, MBH will be used in conjunction with SNOPT, since together they constitute one of, if not the most common and well-performing optimization strategy in literature regarding the Sims-Flanagan Model.[6,7,19] MBH, or Iterated Local Search,[24] is a metaheuristic algorithm that samples candidate decision vectors in a neighbourhood of the existing decision vectors, instead of searching the whole decision space, for a faster convergence. This algorithm starts with a randomly generated decision vector to which a perturbation is applied. The perturbed decision vector is provided as an initial guess to SNOPT, which yields an optimized decision vector. This optimized decision vector substitutes the original one if it has better fitness. There is a counter which gets reset every time a decision vector with better fitness is found, and incremented whenever the new decision vector is worse than the one who originated it. In case the counter goes over a certain user-defined number, a new decision vector is independently sampled from the decision space and the process keeps going as long as the maximum number of function evaluations is not exceeded. Yam et al.[6] set that limit at 500 iterations, and it is used in this work as well. Furthermore, this work also uses the perturbations defined by the aforementioned authors, which are described below:

1. For each decision variable x_i, and its lower and upper bounds $x_{\mathrm{lb},i}$ and $x_{\mathrm{ub},i}$, respectively, add to x_i a value uniformly chosen in the interval $[-0.05(x_{\mathrm{ub},i} - x_{\mathrm{lb},i}), 0.05(x_{\mathrm{ub},i} - x_{\mathrm{lb},i})]$.
2. With a probability of 10%, shift the departure epoch, either forwards, or backwards with equal probability, by an amount of time corresponding to the synodic period between the arrival and departure bodies.

To guarantee that the problem bounds are respected, the perturbations follow some rules. The first perturbation can only increase or decrease the decision variables up to their upper or lower bounds, respectively. The second perturbation has its direction chosen in a way such that the bounds are never violated, being shifted forwards if the departure epoch is too close to its lower bound and backwards if it is too close to the upper bound.

Population-Based Approach

The population-based approach taken in this paper is rather similar to the one taken in the work of Wang et al.[10] It comprises global and local search phases in which GRNN and RBFN-based models, respectively, are constructed as surrogates at every iteration. A flow chart illustrating the optimization process can be found in Figure 2. Said process starts with a population of size N_P being initialized, which is done by generating decision vectors using Latin Hypercube Sampling (LHS), that are then provided as initial guesses to SNOPT for optimization. The optimized decision and fitness vector pairs are stored both in the population and in the dataset, with the latter containing all the of the distinct vector pairs yielded by SNOPT during the optimization process, when SNOPT is used to optimize the Sims-Flanagan problem. It is important to mention that new instances are only included in the dataset if they are not duplicates, which is assessed using fuzzy comparisons. Two decision vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$, consisting of double-precision floating-point numbers, are considered approximately equal if:

$$\|\mathbf{x}_1 - \mathbf{x}_2\| \leq 10^{-12} \cdot \min(\|\mathbf{x}_1\|, \|\mathbf{x}_2\|). \tag{22}$$

Finally, the number of Function Evaluations (FE) of the Sims-Flanagan problem, computed while SNOPT runs, is saved and what remains of the optimization process keeps being executed while that number stays below a maximum number of function evaluations $\mathrm{FE}_{\max}$.

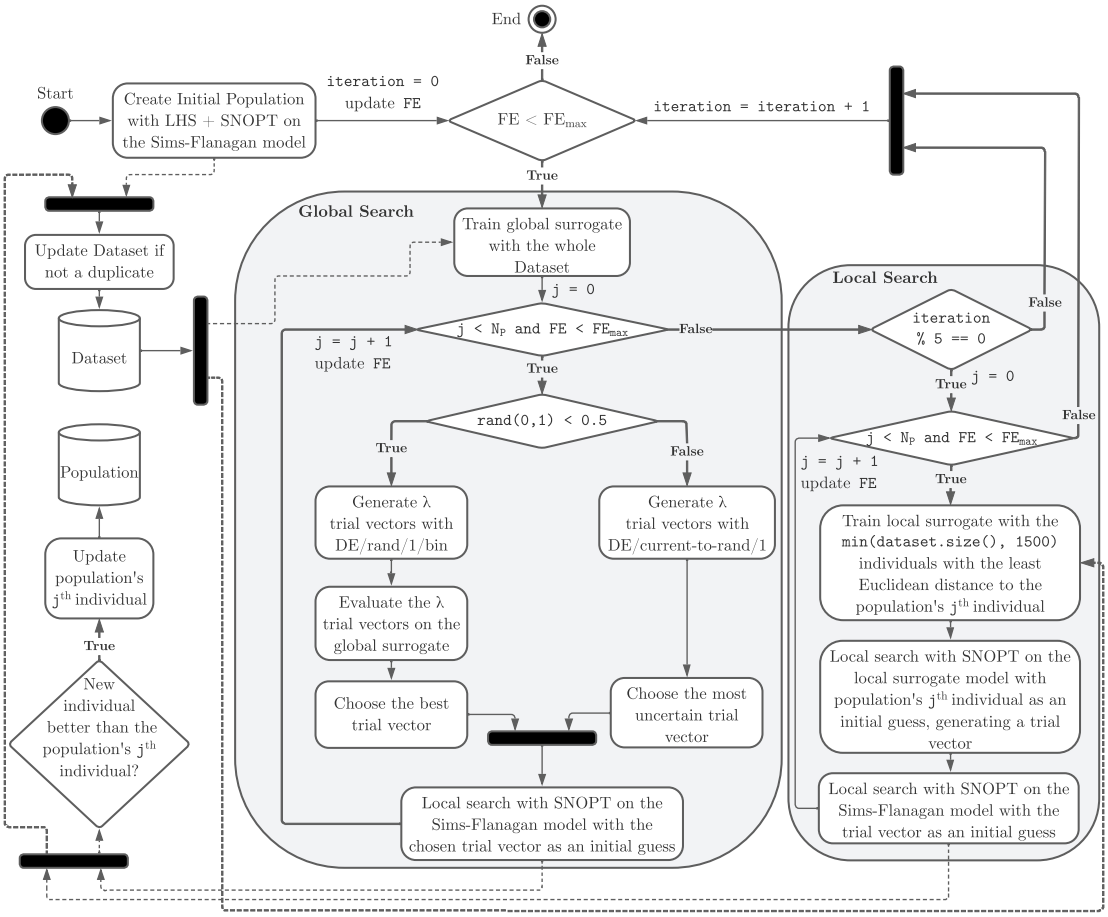


Figure 2. Flow diagram of the optimization process.

Each iteration of the optimization process starts with a **global search** phase. In that phase, a GRNN model is trained with the whole dataset. Two operators from the algorithm proposed by Storn and Price,[25] differential evolution, are used to generate candidate vectors that can serve as initial guesses for SNOPT. The first operator is denoted by "DE/rand/1/bin", which corresponds to a mutation using solely randomly selected individuals, followed by binary crossover. The second operator, denoted by "DE/current-to-rand/1" simply consists of mutation but besides using randomly selected individuals, the mutated decision vector must have information about the specific individual it could replace in the population. Let $P = \{\mathbf{x}_1 \dots \mathbf{x}_{N_P}\}^T$ denote the population of size N_P and $\mathbf{x}_i = \{x_{i1} \dots x_{iD_{\text{in}}}\}^T$ a D_{in}-dimensional decision vector of that population, a mutant vector $\mathbf{u}_i = \{u_{i1} \dots u_{iD_{\text{in}}}\}^T$ can be computed according to the expressions below:

- DE/rand/1/bin

$$u_{ij} = \begin{cases} x_{r_1 j} + F \times (x_{r_2 j} - x_{r_3 j}), & \text{if } rand_{ij} \leq CR \text{ or } j = j_{rand}, \\ x_{ij}, & \text{otherwise,} \end{cases} \tag{23}$$

- DE/current-to-rand/1

$$u_{ij} = x_{ij} + F \times (x_{r_1 j} - x_{ij}) + F \times (x_{r_2 j} - x_{r_3 j}), \tag{24}$$

where $i = 1, \ldots, N_P$ and $j = 1, \ldots, D_{\text{in}}$. Additionally, r_1, r_2, and r_3 are three mutually different integers uniformly generated from $\{1, \ldots, N_P\}$, $rand_{ij}$ is a random number uniformly drawn from the interval [0, 1], and j_{rand} is an integer uniformly selected from $\{1, \ldots, D_{\text{in}}\}$. Finally, F corresponds to the scaling factor, and CR to the crossover ratio. Similarly to what happens with MBH, the perturbations are limited to ensure that the decision variables stay within their respective bounds.

For each individual in the population, one of the two operators is chosen with equal probability to generate λ trial vectors. If DE/rand/1/bin is chosen, the GRNN surrogate is used to evaluate the λ new trial vectors and the best candidate is selected according to the feasibility rules. However, if DE/current-to-rand/1 is chosen, the most uncertain vector is chosen as the best candidate. Taking the recommendation of Jin[26] , the Euclidean distance can be used as a measure of uncertainty when the surrogate model does not have an inherent way to compute it, which is the case with the GRNN model. Therefore, the chosen vector will be the one whose minimum distance (measured in the decision space) from all the vectors in the dataset is maximal (out of the λ trial vectors). The most uncertain vector is chosen in an attempt to explore the decision space in regions that have not been searched previously, where the surrogates are expected to have lower accuracy. After a new individual is provided as an initial guess to SNOPT, the resulting fitness vector is compared to the individual in the population that gave origin to the initial guess. If the new fitness is better, the new individual takes the place of the old one in the population. In any case, the new decision vector and respective fitness are added to the dataset.

Every five global searches, a **local search** is performed. Early test simulations showed that the local search did not introduce many individuals in the population after the first five to ten iterations, which may be due to the value set for the feasibility tolerance resulting in constraints that are relatively easy to satisfy. Another set of early simulations with stricter feasibility tolerance values had the local search contributing more throughout the optimization process. As such strictness was not necessary, it was decided to reduce the frequency of the exploitation mechanism by performing a local search every five global searches, with the latter, the exploration mechanism, happening more often.

During the local search, for each of the N_P individuals in the population, a separate RBFN-based surrogate is built. The minimum between 1500 (from the hyperparameter tuning results) and the total size of the dataset is chosen as the amount of training instances each local surrogate is built with. Wang et al.[10] compute the maximum as $(D_{\text{in}} + 1)(D_{\text{in}} + 2)/2$, but treat problems with, at most, $D_{\text{in}} = 30$. In this work, D_{in} can go up to around 100, which would make training an RBFN-based model on an amount of instances dictated by the previous expression extremely expensive. As each of the population's individuals will be provided as an initial guess for a local search on the surrogate model, it is desired that the surrogate is as accurate as possible within a neighborhood of the corresponding individual. Therefore, the dataset instances chosen to build each surrogate with, are the ones whose decision vector is closest to the population's individual being considered, in terms of Euclidean distance. This, of course, from the point that the dataset starts having more than 1500 instances. After each of the surrogates is built, the corresponding individual is provided as an initial guess for SNOPT to perform local optimization on that surrogate. If the RBFND model is being used, SNOPT can make use of derivatives that are computed analytically. Afterwards, the decision vector that results from local optimization on the surrogate model is provided as an initial guess to SNOPT a second time, but for SNOPT to perform local optimization on the exact, Sims-Flanagan model. Similarly to the global search phase, the resulting decision and fitness vector pair are added to the dataset, and the population is updated if the new individual is better than the one that originated it.

As previously mentioned, this paper uses a very similar optimization approach to the one taken by Wang et al.,[10] with the major differences being worth enumerating. In this work, the surrogates attempt to only approximate the decision and respective fitness vector pairs that have been yielded by SNOPT optimization, instead of any possible pair of vectors. In the global search, the Euclidean distance between the decision vectors is used as a measure of uncertainty. The local search phase

also has differences, with the local optimization on the surrogate making use of the analytical derivatives provided in the RBFND model, it not being done at every iteration, and the local surrogate having a problem-independent upper bound on the amount of training instances, due to the higher-dimensional decision spaces in this work. Finally, although equality and inequality constraints can be treated in a similar way, this work deals with the former, whereas Wang et al.[10] solve problems with the latter.

Ablation Study and Model Variants

In an attempt to understand how the different components of the surrogate model influence the result of the optimization process, an *ablation study* was performed, where said components are removed, one at a time, and the results are compared.

Two variants of a surrogate model using aspects taken from MBH were also built and evaluated against the rest. The first variant removes the local surrogate and applies the MBH and synodic perturbations in the global search phase, instead of the DE operators, always choosing to evaluate the best predicted individual. The other variant, as long as the dataset has at least 125 instances, also generates λ new individuals using the perturbations, evaluates them in a GRNN surrogate trained on the entire dataset, and chooses the best one according to the feasibility rules to be provided as an initial guess to SNOPT. This second variant uses the counter that keeps track of how many iterations MBH goes through without improvements, and if it goes over the aforementioned 500 iterations, randomly generates a new individual from the entirety of the problem bounds. The justification for these models will come from the results themselves, and it will be presented in the coming section.

Below is a list with the description of each of the models with which simulations were run, and their respective designation in the remainder of the paper, in bold:

- **full**: The optimization is run with global and local search phases, and with derivative information, given by the RBFND model. Population-based.
- **just_local**: The optimization is ran without the global search phase. Population-based.
- **just_global**: The optimization is ran without the local search phase. Population-based.
- **no_global_no_local**: The optimization is ran without both the global and local surrogates, essentially comprising DE operators and SNOPT. Population-based.
- **no_derivatives_RBFND**: The optimization is ran on the RBFND model, without derivative information. Population-based.
- **no_derivatives_RBFN**: The optimization is ran on the RBFN model, that does not have derivative information, but has a more descriptive decision vector with the inclusion of the departure and arrival states. Population-based.
- **mbh**: The optimization is ran with the regular MBH scheme. Not population-based.
- **global_mbh_perturbation:** The optimization is ran without the local search phase, and with the perturbations used in MBH instead of the DE operators, in the global search phase. Population-based.
- **mbh_global_surrogate:** The optimization is ran with a global surrogate in the regular MBH scheme, with a counter for maximum iterations without improvement. Not population-based.

EXPERIMENTAL STUDY

The Optimization Problem

In this work, an Earth-Mars rendezvous mission is considered, as it is an example problem already present in literature.[5,8] The bounds of the optimization problem can be found in Table 1, whereas the parameters concerning the spacecraft, surrogate models, and optimization algorithms are presented in Table 2. Many of these parameters are in accordance to the ones used in the works of Yam et al.[5] and Wang et al.[10]. Optimization runs are performed for 10, 20, and 30 impulsive shots, with

the maximum number of function evaluations of the Sims-Flanagan problem's fitness being set at 80×10^6, 100×10^6, and 120×10^6, respectively. As it is expected for the complexity and difficulty of the optimization problem to increase for larger decision vectors, the amount of allowed function evaluations also increases with the number of impulsive shots, in an attempt to improve convergence.

Table 1. Unscaled problem bounds.

Decision Variable	Unit	Lower Bound	Upper Bound
Departure epoch	mjd2000	5478.5 (Jan. 1, 2015)	9131.5 (Jan. 1, 2025)
Time of Flight, ToF	day	300	1000
Orbit insertion velocity, V_∞	m s^{-1}	0	3000
Azimuthal direction of V_∞, θ_∞	rad	0	2π
Polar direction of V_∞, φ_∞	rad	0	π
Throttle magnitudes, τ_i	-	0	1
Azimuthal direction of τ_i, θ_i	rad	$-\pi$	π
Polar direction of τ_i, φ_i	rad	$-\pi$	π

Table 2. Spacecraft, surrogate, and optimization parameters

Parameter	Value
Spacecraft Parameters	
Initial mass	1500 kg
Specific impulse	3000 s
Thrust	0.135 N
Surrogate Parameters	
GRNN $2\sigma^2$	$\frac{1}{2}$
RBFND basis function	Gaussian, $2\sigma^2 = 2$
RBFN basis function ($N_I = 10$)	Gaussian, $2\sigma^2 = 2$
RBFN basis function ($N_I = 20, 30$)	Cubic
Amount of individuals generated with DE, λ	100
Maximum training dataset size of local surrogate	1500
Population-Based Optimization Parameters	
Population Size, N_P	80
Scaling factor for DE/rand/1/bin, F	0.8
Scaling factor for DE/current-to-rand/1, F	0.4
Crossover ratio, CR	0.4
SNOPT major feasibility tolerance	10^{-6}
MBH-Based Optimization Parameters	
Maximum iterations without improvement	500
Perturbation of the decision vector	0% to $\pm$5% of the decision variable range
Synodic Perturbation Probability	10%
Earth-Mars Synodic Period	779.94 julian days

Simulation Setup

As mentioned in the previous section, alongside the main optimization run with GRNN and RBFND surrogates, five other runs were conducted for an ablation study, as well as an MBH run, as a way to provide means of comparison with the methods commonly used in literature. Furthermore, two runs incorporating MBH and surrogate aspects together were conducted, in order to do a preliminary analysis beyond the population-based surrogate-models. This led to a total

of nine optimization runs. A 95%-confidence Wilcoxon rank-sum test,[27] which is commonly used in surrogate-assisted optimization literature[10,11,13] , was performed to evaluate the quality of the results, with some statistical significance being provided by each of those runs being conducted with 20 different random seeds. The Wilcoxon rank-sum test was chosen since it evaluates whether two sets of measurements are drawn from the same distribution without making any assumptions over said distribution, which allows for useful comparisons between the different models. The optimization runs were executed within Delft University of Technology's Eudoxos server*, with an Anaconda environment being used to manage dependencies†, and `tudat-learn` being installed from source from the develop branch as of 28 April 2022.

Results

The results obtained during the simulations and a thorough analysis of said results are presented in the coming pages. This includes the evolution of the populations over the course of the simulations for the population-based models, the statistical results yielded by the Wilcoxon rank-sum test, and the best solutions obtained among all of the optimization runs.

Figures 3 and 4 contain plots of the mean constraint violation and mean feasible objective, respectively, of the population over the number of exact evaluations of the fitness function originated by the Sims-Flanagan model. The mean constraint violation corresponds to the constraint violation of each individual, computed according to Equation (11), averaged over the 80 individuals of the population and afterwards, averaged over the 20 independent runs. The mean feasible objective or mean objective of the feasible individuals corresponds to the average ΔV value of the individuals in the population whose fitness vector does not have any of its constraints violated. Again, this is averaged over the 20 independent runs.

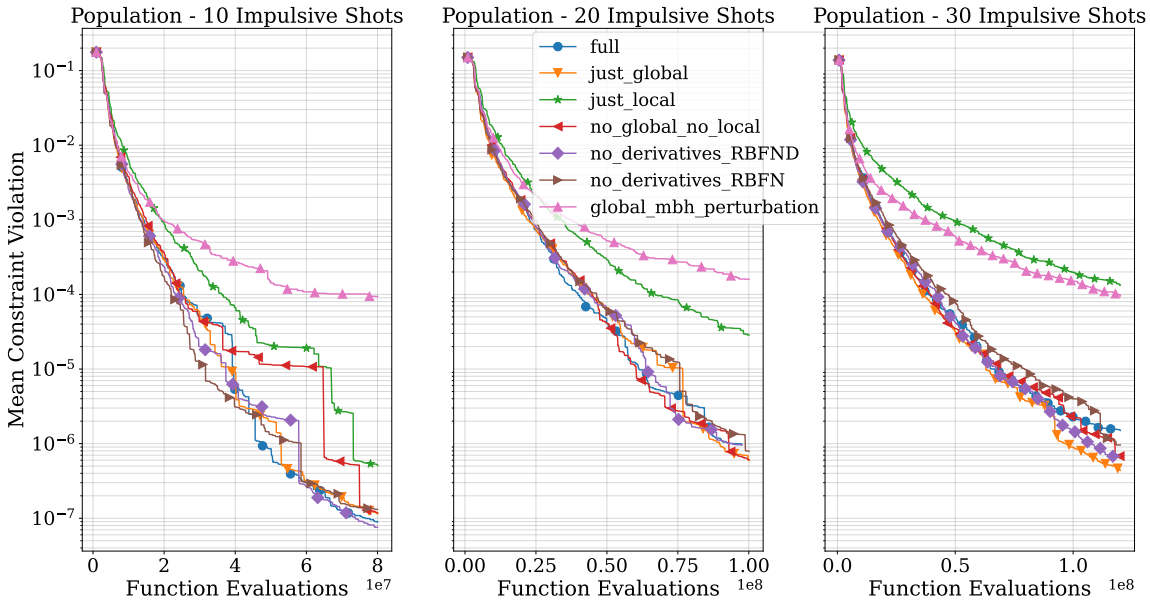


Figure 3. Mean of the constraint violation of the individuals in the population, over the simulation, for the different surrogate variants. Averaged over the 20 random seeds.

*4×Intel® Xeon™ E5-2683 v3 CPUs (Central Processing Unit) and around 260GB of RAM (Random Access Memory). Ten single-threaded simulations were run at a time, in parallel, with less than 1% of the total memory being used at all times.

†`Tudat` 2.9.0, `pagmo` 2.16.1, and `pagmo_plugins_nonfree` 0.22 to manage SNOPT 7.6.

In Figure 3, it can be seen that the average constraint violation value of the individuals in the population is monotonically non-increasing, which happens since new individuals are only introduced to the population if their constraint violation value is less than or equal to the one of the individual whose place they would be taking. The same cannot be said about Figure 4, where the plotted curves are not necessarily monotonic. The increase in the mean of the objective function or total ΔV of the feasible individuals can happen when not every individual in the population is feasible and a new feasible individual is found, whose ΔV value is larger than the current mean.

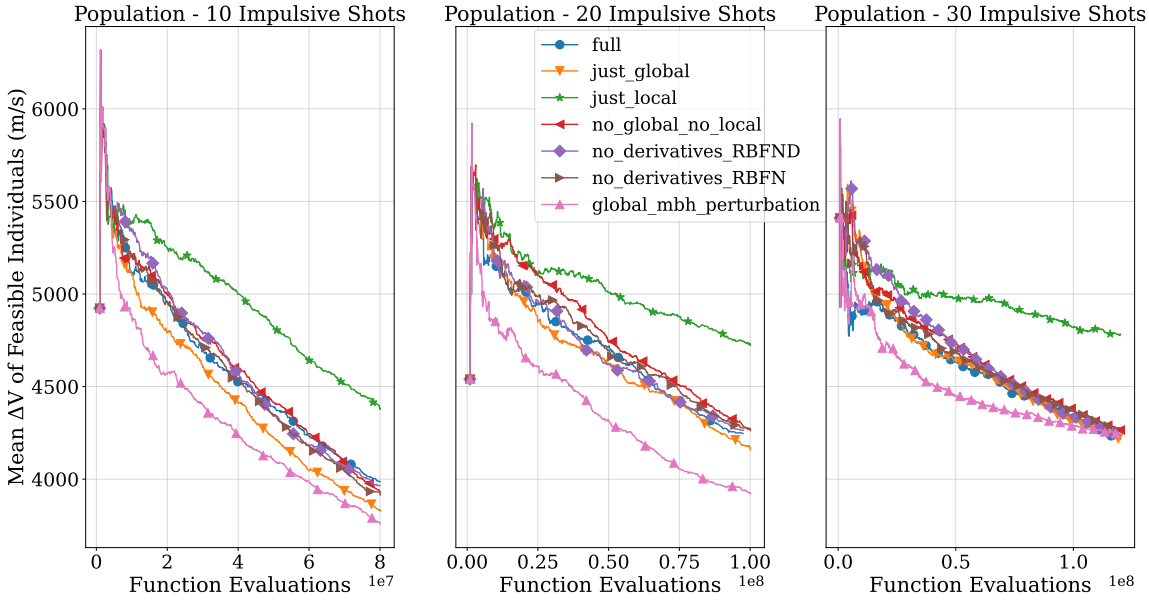


Figure 4. Mean of the objective function of the feasible individuals in the population, over the simulation, for the different surrogate variants. Averaged over the 20 random seeds.

Regarding the performance of the surrogate models, from Figure 4, it can be seen that none of the models that use DE operators (the first six) for their perturbations clearly has a better performance over the others, except maybe for the **just_global** model, when there are only 10 impulsive shots. It is also evident that **just_local** is the worst performing model, meaning that, when the global search is removed, the local search alone is likely unable to explore a large enough portion of the search space, resulting in a slower convergence. There is even a point to be made against having a local search at all. It may be the case that SNOPT, as a local optimization algorithm, performs enough of a local search that the challenge lies in tuning the global search phase.

In Figures 3 and 4, the values are also plotted for the **global_mbh_perturbation** model, which is an attempt at improvement by completely removing the local search and having the perturbations used in the **mbh** model instead of the DE operators in the global search phase. The choice of the MBH perturbation comes from the results in Tables 3, 4, 5, which will be explained and examined shortly. It is clear that, while the mean constraint violation for this model is larger than the others', meaning that there are fewer feasible individuals in the population, the mean objective value of said individuals is always the lowest for both 10 and 20 impulsive shots, and also almost always the lowest for the majority of the duration of the simulations with 30 impulsive shots.

Even though these figures provide a good way to visualize the evolution of the population through the optimization process, it is extremely important to evaluate every solution that was generated to correctly compare all the models, preferably with a statistical method. This is necessary since in the current setup for population-based optimization, an individual is only introduced in the population

if it is better than the individual that generated it, even if it is better than other individuals in the population. Furthermore, since some of the said models are not population-based, it is not possible to visualize how they perform using metrics that are dependent on the existence of a population.

Table 3. Statistical results for the simulations with 10 impulsive shots.

Simulation	Best (m s^{-1})	Median (m s^{-1})	STD (m s^{-1})	Count
full	2621.9	4739.0	1859.2	5363
just_local	2622.4	5423.6 (+)	1798.1	3862
just_global	2619.5	4534.9 (−)	1825.8	5662
no_global_no_local	**2618.7**	4926.0 (+)	1856.9	5224
no_derivatives_RBFND	2622.0	4712.9 (=)	1845.6	5384
no_derivatives_RBFN	2625.8	4628.4 (−)	1850.0	5548
mbh	2619.8	4118.1 (−)	1975.7	5474
global_mbh_perturbation	2619.2	3835.5 (−)	1700.9	6301
mbh_global_surrogate	2619.0	**3007.8** (−)	**1441.7**	**7564**

(+): significant difference favoring the **full** surrogate model; (−): significant difference favoring the model being compared against **full**; (=): no significant difference between the two models being compared.

The 95%-confidence Wilcoxon rank-sum test was applied using the same approach as the one taken by Miranda-Varela and Mezura-Montes[11], with the **full** surrogate model being taken as the base algorithm for the test. The variants are compared against the base algorithm, with the required ΔV of *all* the solutions, that is, the feasible individuals, obtained through the 20 independent runs being collected and serving as the basis of comparison. The Wilcoxon rank-sum test tests the null hypothesis that two sets of measurements are drawn from the same distribution, with the alternative hypothesis being that the values in one of the samples are more likely to be larger than the values in the other sample. Tables 3, 4, and 5 contain the statistical results obtained for the runs with 10, 20, and 30 impulsive shots, respectively. Those tables contain, for each of the models, over the 20 independent runs, the ΔV of the best solution, the median ΔV and its standard deviation among all the solutions that were found, as well as the amount of solutions that were found. For each of those quantities, the best value is identified in **bold**. Finally, (+), (−), and (=) identifiers are used to represent the result of the Wilcoxon rank-sum test. (+) indicates that there is a significant difference favoring the base algorithm with respect to the variant it is being compared with, (−) indicates that there is also a significant difference but that it favors the variant, and (=) indicates that there is no significant difference between the algorithms that are being compared.

Table 4. Statistical results for the simulations with 20 impulsive shots.

Simulation	Best (m s^{-1})	Median (m s^{-1})	STD (m s^{-1})	Count
full	2627.4	4402.0	1719.4	3216
just_local	2627.7	5206.0 (+)	1766.5	2114
just_global	2635.4	4360.5 (=)	1723.9	3480
no_global_no_local	2631.1	4654.8 (+)	1773.5	3279
no_derivatives_RBFND	2643.7	4390.3 (=)	1766.0	3299
no_derivatives_RBFN	2626.1	4391.0 (=)	1766.8	3349
mbh	2622.0	3913.9 (−)	1963.3	3270
global_mbh_perturbation	2622.2	3761.1 (−)	1482.0	4559
mbh_global_surrogate	**2621.4**	**3012.4** (−)	**1195.7**	**6434**

(+): significant difference favoring the **full** surrogate model; (−): significant difference favoring the model being compared against **full**; (=): no significant difference between the two models being compared.

From the tables, it can be seen that between the **full** surrogate, the other variants from the ablation study, and the **mbh** model, the latter always has the statistically best results, with the

median of its solutions being the lowest for every amount of impulsive shots. This is what led to experimenting with the **global_mbh_perturbation** and the **mbh_global_surrogate** models, as these would incorporate aspects of MBH with surrogate models. It is also important to reiterate that **mbh** and **mbh_global_surrogate** are not present in Figures 3 and 4 since they are not population based methods.

While the graphical performance of the **global_mbh_perturbation** model has already been evaluated, the results of Tables 3, 4, and 5, show that it is statistically better than the original **mbh** model, and than all the surrogate models that use DE operators. However, the model that displays the absolute best performance is the one denoted by **mbh_global_surrogate**. Except for the best solution for 10 impulsive shots, this model presents the best results for every figure of merit presented in the tables. Notable examples include the approximately 50 m s^{-1} difference from its best solution to the best solution of any of the other models, in Table 5, or the difference of at least 700 to 800 m s^{-1} in the median solutions with regard to the other models, evidenced by Tables 3 and 4. This is particularly impressive taking into account that it also yields the largest amount of solutions, that is, individuals with feasible fitness vectors. It is also interesting that the standard deviation of the solutions yielded by the **mbh_global_surrogate** model is lower from all the models, evidencing less disparity between said solutions.

Table 5. Statistical results for the simulations with 30 impulsive shots.

Simulation	Best (m s^{-1})	Median (m s^{-1})	STD (m s^{-1})	Count
full	2750.0	4490.0	1231.6	3697
just_local	2762.6	5019.8 (+)	1177.7	1112
just_global	2768.2	4477.8 (=)	1227.6	4299
no_global_no_local	2742.6	4665.6 (+)	1232.1	3571
no_derivatives_RBFND	2752.2	4523.6 (=)	1257.5	3709
no_derivatives_RBFN	2728.5	4488.0 (=)	1244.0	3601
mbh	2752.7	3885.6 (−)	1253.8	3827
global_mbh_perturbation	2744.5	3665.4 (−)	1226.6	6484
mbh_global_surrogate	**2698.1**	**3237.8** (−)	**978.2**	**7700**

(+): significant difference favoring the **full** surrogate model; (−): significant difference favoring the model being compared against **full**; (=): no significant difference between the two models being compared.

Table 6. ΔV and state mismatch of the best solutions found by the mbh_global_surrogate model, for 10, 20, and 30 impulsive shots.

Impulses	ΔV (m s^{-1})	Δr_x (m)	Δr_y (m)	Δr_z (m)	Δv_x (m s^{-1})	Δv_y (m s^{-1})	Δv_z (m s^{-1})
10	2619.0	113170	76147	130.61	3.9727×10^{-2}	2.9667×10^{-2}	1.6138×10^{-3}
20	2621.4	5201.4	467890	1.3248	7.1386×10^{-2}	8.8710×10^{-4}	2.9243×10^{-3}
30	2698.1	9424.1	60321	1431.1	8.3776×10^{-3}	3.2251×10^{-5}	6.6059×10^{-4}

Table 6 contains the fitness vectors of the best trajectories found by the full surrogate model, and Figure 5 depicts those trajectories. The three trajectories utilise the maximum magnitude for the orbit insertion velocity, 3000 m s^{-1}, resulting in a total ΔV slightly over the 5600 m s^{-1} mark, which corresponds to the approximate ΔV value of a Hohmann transfer between Earth and Mars. Being slightly over the most ΔV-efficient transfer between these two bodies makes it evident that the best trajectory found by the full surrogate is a cheap trajectory.

CONCLUSION

The development of `tudat-learn`, a library with machine-learning tooling was started, and said library was used alongside `Tudat`, `pagmo`, and SNOPT to perform surrogate-assisted surrogate op-

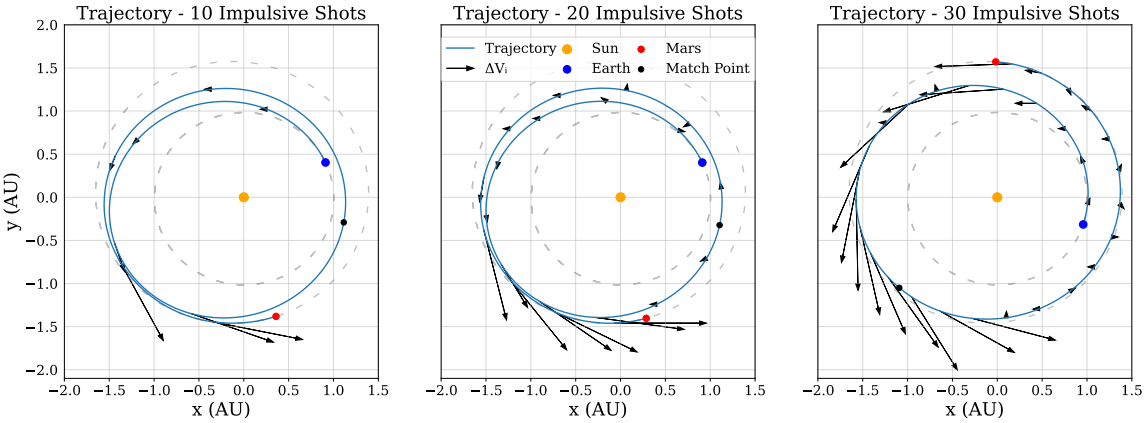


Figure 5. Feasible trajectories with the lowest ΔV found by the mbh_global_surrogate model.

timization of low-thrust, interplanetary trajectories modeled by the Sims-Flanagan transcription in an Earth-Mars transfer. GRNN and RBFN surrogates were employed in global and local searches, respectively, with the former using DE operators for the perturbations of the existing individuals and the latter providing an approximation of the derivatives of the problem being optimized. The hyperparameters of the surrogate models were tuned through k-fold cross-validation. Each of the optimization runs was done with 20 different random seeds, for 10, 20, and 30 impulsive shots. An ablation study evaluated the contribution of the different components of the surrogate model, concluding that the local surrogate, with or without derivative information, did not improve the optimization results. The MBH approach that is standard in literature outperformed the surrogates that were using DE operators, so two new surrogate models without DE operators were created. The model that used a regular MBH optimization scheme with a global surrogate to select which decision vector to evaluate next was the best performing model, verified by a 95%-confidence Wilcoxon rank-sum test. The median of the solutions yielded by this model was, at least, 400 m s^{-1} lower than the ones yielded by other models. The best solutions of each model were very similar, except for the case with 30 impulsive shots, in which the aforementioned best-performing model had around a 50 m s^{-1} advantage.

It is concluded that the inclusion of an online global surrogate alongside MBH and SNOPT can improve the optimization of interplanetary low-thrust trajectories modelled by the Sims-Flanagan transcription. However, it is also concluded that local surrogates or global surrogates in a DE framework should not be utilized to optimize such trajectories. Further investigation on the use of surrogate models alongside MBH, SNOPT, and the Sims-Flanagan model, as well as a thorough study on balancing the computational effort are recommended as directions for future work.

REFERENCES

[1] M. D. Rayman, P. A. Chadbourne, J. S. Culwell, and S. N. Williams, "Mission design for deep space 1: A low-thrust technology validation mission," *Acta Astronautica*, Vol. 45, No. 4, 1999, pp. 381–388. Third IAA International Conference on Low-Cost Planetary Missions.

[2] M. D. Rayman, T. C. Fraschetti, C. A. Raymond, and C. T. Russell, "Dawn: A mission in development for exploration of main belt asteroids Vesta and Ceres," *Acta Astronautica*, Vol. 58, No. 11, 2006, pp. 605–616.

[3] O. Çelik, D. A. Dei Tos, T. Yamamoto, N. Ozaki, Y. Kawakatsu, and C. H. Yam, "Multiple-Target Low-Thrust Interplanetary Trajectory of DESTINY+," *Journal of Spacecraft and Rockets*, 2021, pp. 1–18.

[4] J. Sims and S. Flanagan, "Preliminary design of low-thrust interplanetary missions," *AAS/AIAA Astrodynamics Specialist Conference, AAS paper 99-338*, Girdwood, Alaska, August 1999.

[5] C. H. Yam, F. Biscani, and D. Izzo, "Global optimization of low-thrust trajectories via impulsive Delta-V transcription," *27th International Symposium on Space Technology and Science*, 2009.

[6] C. H. Yam, D. D. Lorenzo, and D. Izzo, "Low-thrust trajectory design as a constrained global optimization problem," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, Vol. 225, No. 11, 2011, pp. 1243–1251.

[7] D. H. Ellison, J. A. Englander, and B. A. Conway, "Robust Global Optimization of Low-Thrust, Multiple-Flyby Trajectories," *AAS/AIAA Astrodynamics Specialist Conference, Hilton Head, SC*, 2013.

[8] V. S. Ulibarrena and K. Cowan, "Low-thrust interplanetary trajectory optimization using pre-trained artificial neural network surrogates.," AAS/AIAA Astrodynamics Specialist Conference, AAS Preprint, 2021.

[9] D. Lim, Y. Jin, Y.-S. Ong, and B. Sendhoff, "Generalizing surrogate-assisted evolutionary computation," *IEEE Transactions on Evolutionary Computation*, Vol. 14, No. 3, 2009, pp. 329–355.

[10] Y. Wang, D.-Q. Yin, S. Yang, and G. Sun, "Global and local surrogate-assisted differential evolution for expensive constrained optimization problems with inequality constraints," *IEEE transactions on cybernetics*, Vol. 49, No. 5, 2018, pp. 1642–1656.

[11] M.-E. Miranda-Varela and E. Mezura-Montes, "Constraint-handling techniques in surrogate-assisted evolutionary optimization. An empirical study," *Applied Soft Computing*, Vol. 73, 2018, pp. 215–229.

[12] G. Chen, Y. Li, K. Zhang, X. Xue, J. Wang, Q. Luo, C. Yao, and J. Yao, "Efficient hierarchical surrogate-assisted differential evolution for high-dimensional expensive optimization," *Information Sciences*, Vol. 542, 2021, pp. 228–246.

[13] Q. Liu, X. Wu, Q. Lin, J. Ji, and K.-C. Wong, "A novel surrogate-assisted evolutionary algorithm with an uncertainty grouping based infill criterion," *Swarm and Evolutionary Computation*, Vol. 60, 2021, p. 100787.

[14] F. Biscani and D. Izzo, "A parallel global multiobjective framework for optimization: pagmo," *Journal of Open Source Software*, Vol. 5, No. 53, 2020, p. 2338, 10.21105/joss.02338.

[15] J. A. Sims, P. A. Finlayson, E. A. Rinderle, M. A. Vavrina, and T. D. Kowalkowski, "Implementation of a Low-Thrust Trajectory Optimization Algorithm for Preliminary Design," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Citeseer, 2006.

[16] K. F. Wakker, "Fundamentals of astrodynamics," 2015.

[17] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM review*, Vol. 47, No. 1, 2005, pp. 99–131.

[18] M. T. Ozimek, J. F. Riley, and J. Arrieta, "The Low-Thrust Interplanetary Explorer: A Medium-Fidelity Algorithm For Multi-Gravity Assist Low-Thrust Trajectory Optimization," *AAS Space Flight Mechanics Conference, No. AAS*, 2019, pp. 19–348.

[19] J. A. Englander and B. A. Conway, "Automated solution of the low-thrust interplanetary trajectory problem," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 1, 2017, pp. 15–27.

[20] D. F. Specht, "A general regression neural network," *IEEE transactions on neural networks*, Vol. 2, No. 6, 1991, pp. 568–576.

[21] A. Díaz-Manríquez, G. Toscano, and C. A. C. Coello, "Comparison of metamodeling techniques in evolutionary algorithms," *Soft Computing*, Vol. 21, No. 19, 2017, pp. 5647–5663.

[22] N. Flyer, B. Fornberg, V. Bayona, and G. A. Barnett, "On the role of polynomials in RBF-FD approximations: I. Interpolation and accuracy," *Journal of Computational Physics*, Vol. 321, 2016, pp. 21–38.

[23] L. Stubbig and K. Cowan, "Improving the evolutionary optimization of interplanetary low-thrust trajectories using a neural network surrogate model - AAS 20-658 - Preprint," 08 2021.

[24] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," *Handbook of metaheuristics*, pp. 320–353, Springer, 2003.

[25] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, Vol. 11, No. 4, 1997, pp. 341–359.

[26] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, Vol. 1, No. 2, 2011, pp. 61–70.

[27] F. Wilcoxon, "Individual comparisons by ranking methods," *Breakthroughs in statistics*, pp. 196–202, Springer, 1992.

APPENDIX A: RBFN DERIVATIVE DERIVATION

Equation (17) is derived as follows:

$$\begin{aligned}\frac{\partial \hat{f}}{\partial x_j}(\mathbf{x}) &= \frac{\partial}{\partial x_j}\left(\sum_{i=1}^{N_\mathrm{s}} l_i \phi(\|\mathbf{x}-\mathbf{c_i}\|) + p_0 + \sum_{k=1}^{D\mathrm{in}} p_k x_k\right) \\ &= \sum_{i=1}^{N_\mathrm{s}}\left[\frac{\partial l_i}{\partial x_j}\phi(\|\mathbf{x}-\mathbf{c_i}\|) + l_i \frac{\partial \phi}{\partial x_j}(\|\mathbf{x}-\mathbf{c_i}\|)\right] + \frac{\partial p_0}{\partial x_j} + \sum_{k=1}^{D\mathrm{in}}\left[\frac{\partial p_k}{\partial x_j}x_k + p_k\frac{\partial x_k}{\partial x_j}\right] \\ &= \sum_{i=1}^{N_\mathrm{c}} l_i \frac{\partial \phi}{\partial x_j}(\|\mathbf{x}-\mathbf{c}_i\|) + p_j. \end{aligned} \tag{25}$$

The partial derivatives of the cubic basis function, in Equation (19), are derived below:

$$\begin{aligned}\frac{\partial \phi_C}{\partial x_j}(\|\mathbf{x}-\mathbf{c}_i\|) &= \frac{\partial}{\partial x_j}(\|\mathbf{x}-\mathbf{c}_i\|^3) \\ &= \frac{\partial}{\partial x_j}\left(\left[(x_1-c_{i1})^2 + ... + (x_D - c_{iD_I})^2\right]^{\frac{3}{2}}\right) \\ &= \frac{3}{2}\cdot 2\cdot (x_j - c_{ij})\left[(x_1-c_{i1})^2 + ... + (x_D - c_{iD_I})^2\right]^{\frac{3}{2}-1} \\ &= 3\cdot(x_j - c_{ij})\cdot\|\mathbf{x}-\mathbf{c}_i\|. \end{aligned} \tag{26}$$

The partial derivatives of the squared distance between the input vector and a specific center point are derived below:

$$\begin{aligned}\frac{\partial}{\partial x_j}(\|\mathbf{x}-\mathbf{c}_i\|^2) &= \frac{\partial}{\partial x_j}\left((x_1-c_{i1})^2 + ... + (x_D - c_{iD_I})^2\right) \\ &= 2\cdot(x_j - c_{ij}), \end{aligned} \tag{27}$$

which, in turn, aids in the derivation of Equation (20):

$$\begin{aligned}\frac{\partial \phi_G}{\partial x_j}(\|\mathbf{x}-\mathbf{c}_i\|) &= \frac{\partial}{\partial x_j}\exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma^2}\right) \\ &= \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma^2}\right)\cdot\frac{\partial}{\partial x_j}\left[-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma^2}\right] \\ &= \phi_G(\|\mathbf{x}-\mathbf{c}_i\|)\cdot\left[-\frac{2\cdot(x_j-c_{ij})}{2\sigma^2}\right] \\ &= \phi_G(\|\mathbf{x}-\mathbf{c}_i\|)\cdot\left[-\frac{(x_j-c_{ij})}{\sigma^2}\right]. \end{aligned}$$

APPENDIX B: HYPERPARAMETER TUNING RESULTS

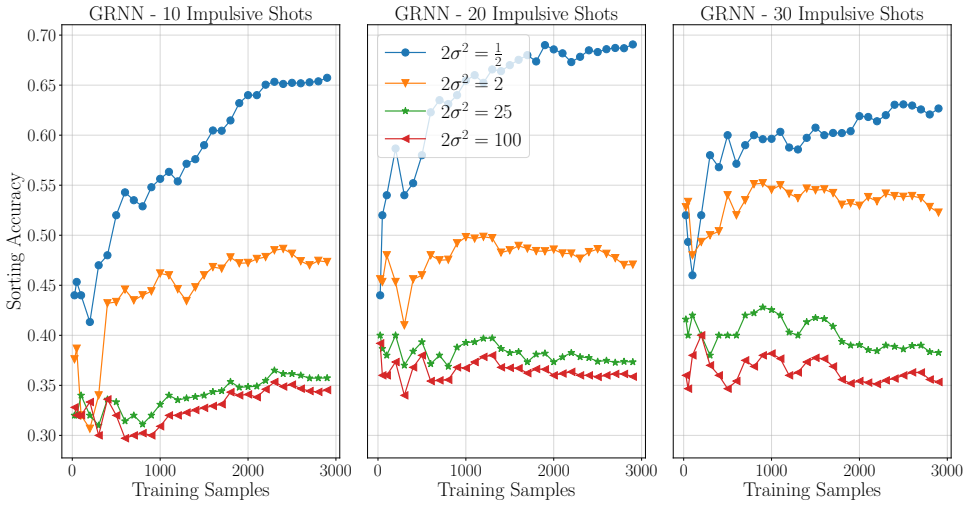


Figure B.1. Sorting accuracy of the GRNN model obtained with cross-validation for the different impulsive shot amounts, $2\sigma^2$ values, and training dataset sizes.

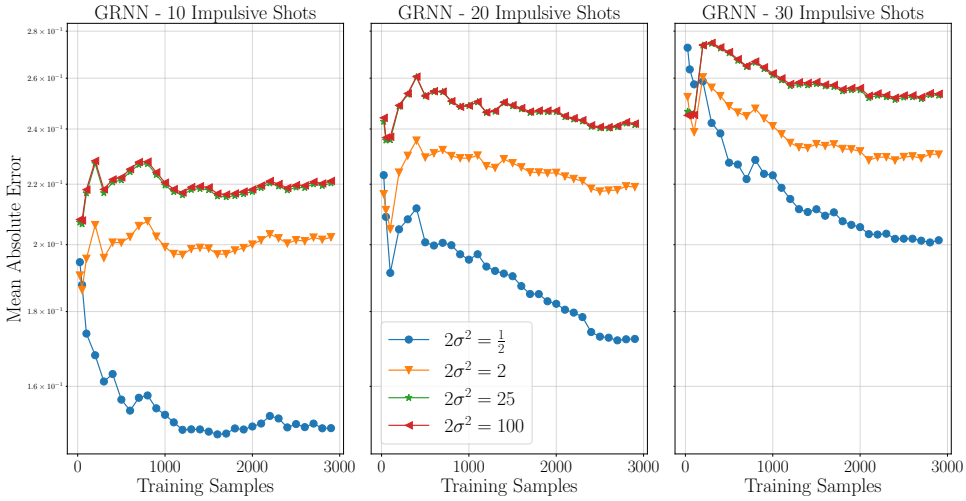


Figure B.2. Mean absolute error of the GRNN model obtained with cross-validation for the different impulsive shot amounts, $2\sigma^2$ values, and training dataset sizes.

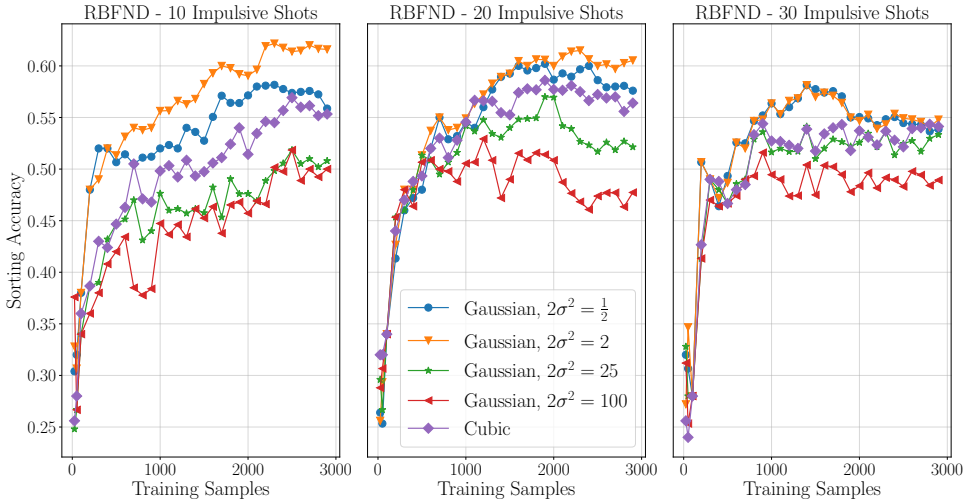


Figure B.3. Sorting accuracy of the RBFND model obtained with cross-validation for the different impulsive shot amounts, basis functions, and training dataset sizes.

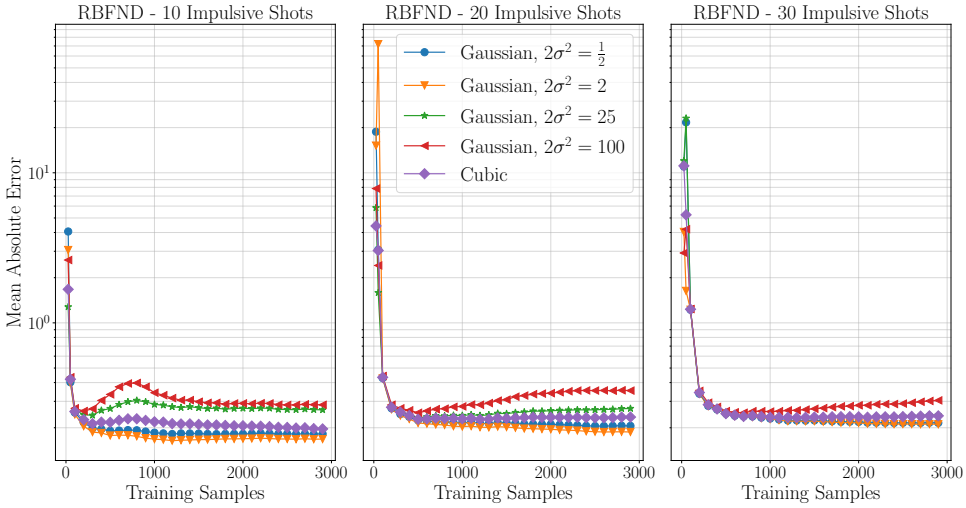


Figure B.4. Mean absolute error of the RBFND model obtained with cross-validation for the different impulsive shot amounts, basis functions, and training dataset sizes.

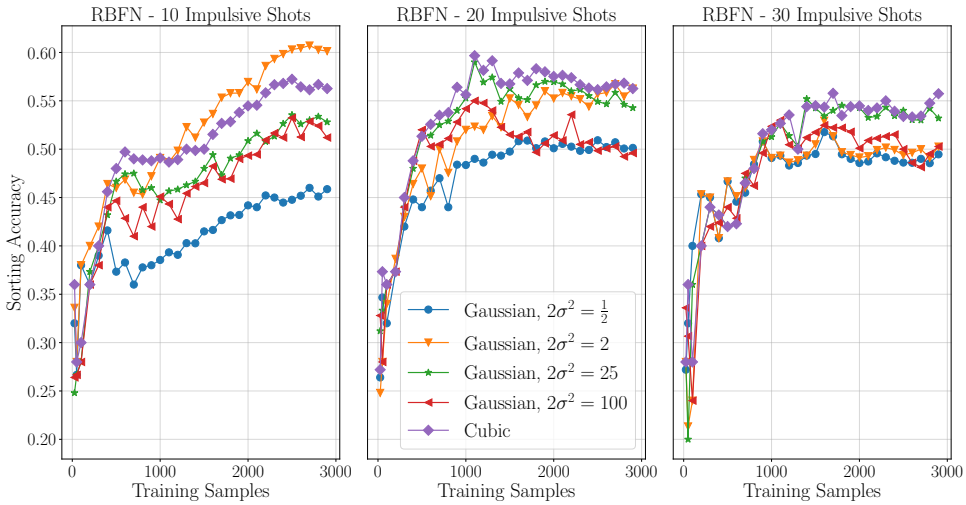


Figure B.5. Sorting accuracy of the RBFN model obtained with cross-validation for the different impulsive shot amounts, basis functions, and training dataset sizes.

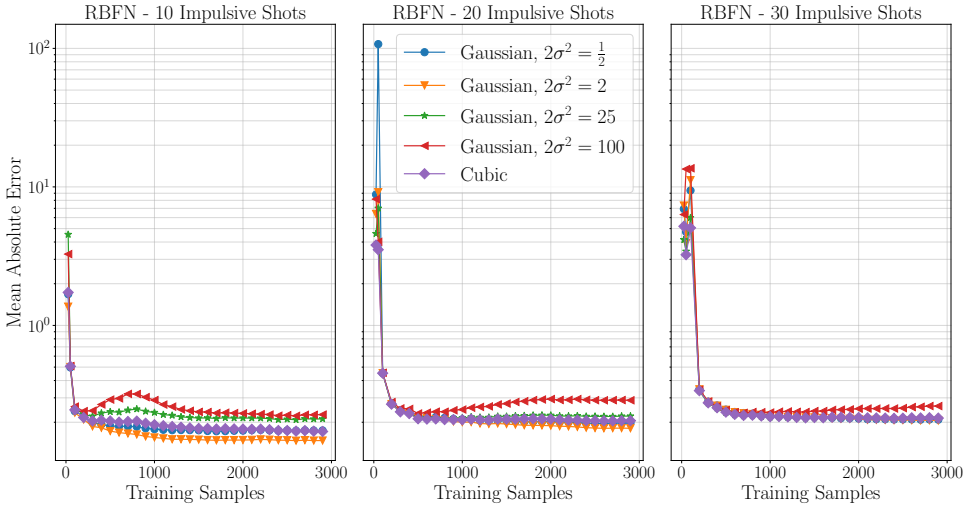


Figure B.6. Mean absolute error of the RBFN model obtained with cross-validation for the different impulsive shot amounts, basis functions, and training dataset sizes.

3

Limitations on Run Time and Performance

The software that was used has some limitations that influence the run time and performance of the algorithms that were used. They will be listed below:

- *Parabolic orbits:*

  As mentioned in the paper, the implementation of the Sims-Flanagan model was taken from `Tudat`. That includes a Keplerian propagator that, at the time of writing, does not have support for parabolic orbits. This is a problem during the optimization runs, as orbital parameters of parabolic orbits are found multiple times, raising exceptions. When that happens during the generation of the initial population, a new individual is sampled from within the problem bounds until a parabolic orbit is not found during propagation. However, when it happens during a global or local search, that individual is ignored and not considered to be introduced in the population. Naturally, this may result in not finding solutions to the problem that could have been better than the ones that were obtained.

- *SNOPT querying dates outside of problem boundaries:*

  The positions of the celestial bodies are queried from SPICE (Spacecraft, Planet, Instrument, "C-matrix", Events) kernels [33] through `Tudat`. A `pagmo::problem` wrapping `Tudat`'s implementation of the Sims-Flanagan model is provided to SNOPT for optimization. Even though this problem includes the problem bounds, multiple times during the optimization runs, exceptions are thrown when SNOPT is running due to the positions of the relevant celestial bodies being queried at a date not covered by the kernel. Note that the kernel covered the entire problem bounds, and these queries were not only outside of those bounds, but also outside of the bounds of the kernel itself. Even though what causes this phenomenon was not investigated, the first guess goes to the fact that SNOPT estimates the Hessian matrix of the problem through finite differencing, which may lead to it needing to query the function outside of problem bounds to get its derivatives at a point within said bounds. These exceptions were handled in the same way as the ones thrown due to the occurrence of parabolic orbits, in addition to resetting the kernels, an action required to keep on using them without restarting the run.

- *Cluster load makes it hard to perform a fair analysis on the computational effort:*

  The simulations were run on Delft University of Technology's Eudoxos server, which is concurrently used by multiple students, having a volatile load. When the load is high, the duration of the simulations can be longer, and it can shorter when the load is low. Since the load varies and cannot be predicted, the duration of the simulations which was recorded is of little meaning. Figure 3.1 contains the run times for every simulation. As an example of this issue with the cluster load, one can see that the simulation with the **mbh_global_surrogate** model often takes much less time to run than the one with the **mbh** model, which is essentially the same as the

mbh_global_surrogate model, but without the surrogate operations, which should translate to a lower computational effort and lower run time for the **mbh** model.

Despite this issue, it is still possible to do a basic comparison of the performance of the different models by making the assumption that the time spent in computing the fitness function yielded by the Sims-Flanagan optimization problem is what dominates the total run time of the simulations. Under this assumption, the different models can be compared for the same amount of fitness function evaluations, as it was done in the paper.

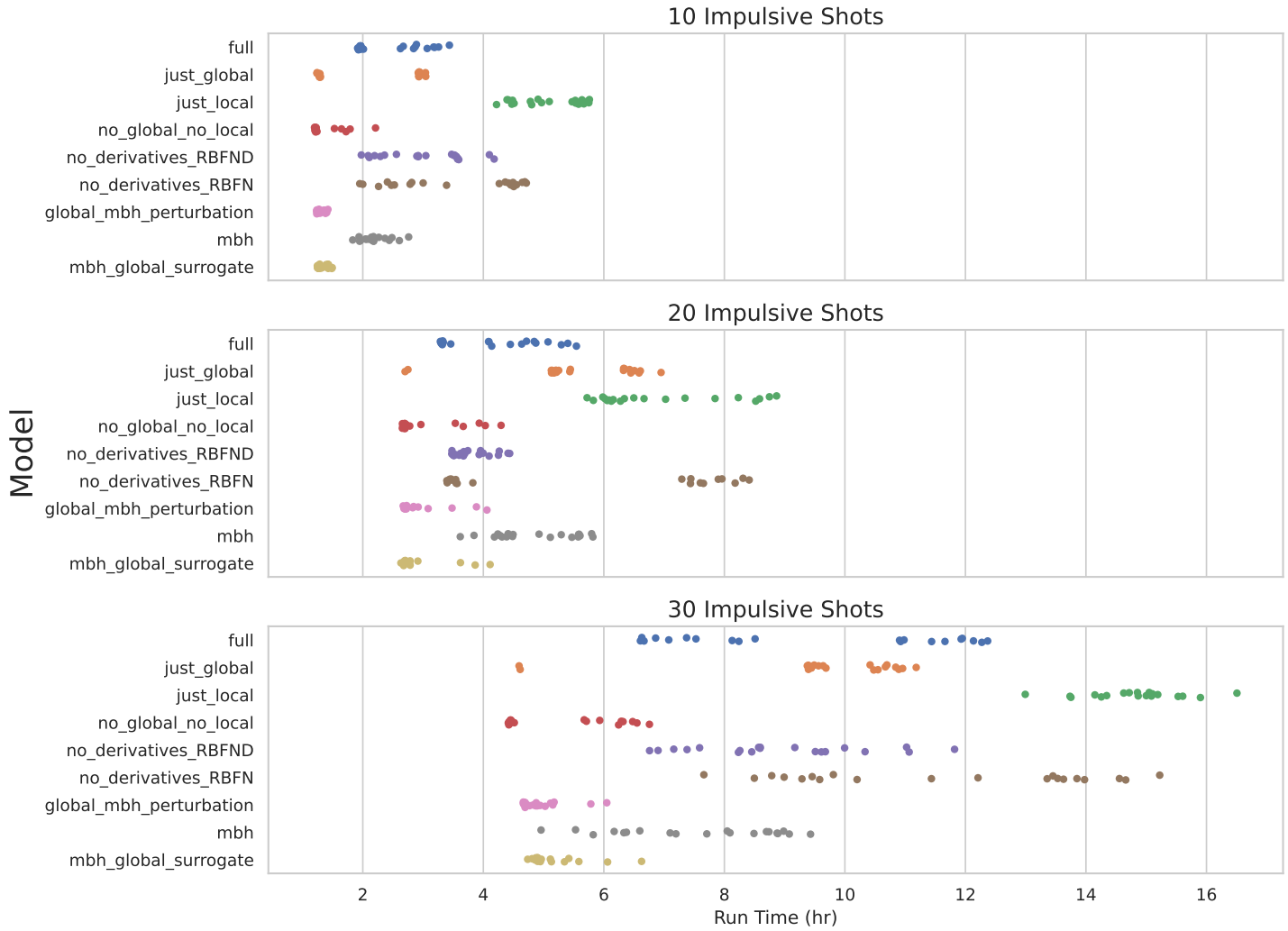


Figure 3.1: Run times of the optimization routines with different models for 10, 20, and 30 impulsive shots. Each dot represents a different random seed.

4

Conclusions and Recommendations

In this chapter, detailed answers to the research questions are presented as the conclusions of the study, accompanied by recommendations for future work directions.

- *How can online GRNN and RBFN surrogates improve the optimization of interplanetary, low-thrust trajectories based on the Sims-Flanagan model?*

  In this work, GRNN and RBFN functions were only used to build global and local surrogates, respectively. While the global surrogates seemed to improve the results of the optimization, the same can not be said of the local surrogates. It is believed that the local surrogates are not beneficial to the optimization process due to SNOPT inherently searching the decision space in a region close to the initial guess that it gets as an input. This searching aspect associated with SNOPT may make the local surrogate redundant, worsening the performance of the models that use said local surrogate. Therefore, it is inferred that, with trajectories modelled by the Sims-Flanagan transcription, it is beneficial to use GRNNs in global surrogates, but that to beat the performance of MBH, which is the method that often yields the best results in literature, the surrogates should be included in that optimization strategy.

- *How can evolutionary and local optimization algorithms be used in combination with online surrogates to perform constrained, single-objective, high-dimensional optimization of interplanetary, low-thrust trajectories?*

  The results make it clear that it is possible to use different kinds of surrogates to perform optimization of interplanetary, low-thrust trajectories in an evolutionary, population-based setting. These surrogates can be global, local, they can provide derivative information or not do it at all, and they can also work with various input sets, and various input dimensions. All of these surrogate models were able to produce solutions that respected the constraints imposed in the optimization problem.

- *Can a surrogate-assisted optimization approach lead to better results than the monotonic basin hopping techniques used in literature?*

  In terms of the best possible solution, both the MBH and the surrogate models were able to reach very similar results. However, it is different in terms of the overall quality of the solutions, which was evaluated by comparing the median solution of the different models as well as their distribution, with a Wilcoxon rank-sum test. This test evidenced that the quality of the solutions found by the standard MBH approach was higher than the one of the solutions yielded by the surrogate models employing DE operators as perturbations. However, it was possible to beat the MBH approach by employing the perturbation that is used there in two different ways: First, by having it serve as the perturbation in a population-based setting, taking the place of the DE operators; and second, with even better results, by employing a global surrogate in the standard MBH approach, just to choose which decision vector is provided to SNOPT as an initial guess.

- *How does each component of a surrogate model contribute to the optimization of interplanetary, low-thrust trajectories?*

Regarding the choice of basis function and hyperparameters, a careful approach is advised. In case of the GRNN, too large $2\sigma^2$ values lead to the model having the same prediction with every input. Low enough values lead to divisions by zero, which is also not desirable. In case one chooses the Gaussian basis function in the RBFN, the $2\sigma^2$ values also need to be within a certain range, to keep the distance matrix values from simply being ones or zeros, respectively, for too large or too little $2\sigma^2$ values. The cubic basis function is a good choice if less hyparameter tuning is desired. Both GRNN and RBFN highly benefit from or even require normalizing the input variables and having the output variables in the same range for a good performance.

As it has been previously mentioned, the local surrogate did not bring many benefits to the optimization process. Additionally, there was no significant difference when removing the derivative information. It is concluded that in the current optimization scheme, SNOPT is enough for exploitation, and a local search phase is not necessary. Regarding exploration, the global surrogate does its job well, contributing to the good quality of the solutions, especially when a good perturbation is chosen. The global surrogate, being constructed out of a GRNN, has another advantage over the local surrogate, as it can be trained with the whole dataset without a great increase in computational effort, which scales linear with its size. Oppositely, the fitting time for an RBFN scales with the size of the dataset cubed, making it impractical to go over certain amounts, depending on the use-case.

It is considered that the goal of *improving the optimization of interplanetary, low-thrust trajectories based on the Sims-Flanagan transcription through the use of online surrogate models* was successfully achieved with this work. However, it does raise some questions that can be taken as starting points for future work:

- As shown by the results, the surrogate model incorporated in the MBH approach was the one that led to the best results. Since most of the duration of this research project was occupied with examining surrogate models using differential evolution operators, it would be interesting to have a more detailed analysis on surrogate models working alongside MBH, experimenting with models extended from the ones developed here, such as incorporating a local search in the process. Furthermore, it would be beneficial to expand from the single Earth-Mars transfer to a multiple leg mission with gravity assists and deep space manoeuvres with multiple levels of surrogate-assisted optimization, such as inner loops to optimize the thrust profiles of each leg and outer loops that focus on the general mission parameters. This would bring more value to mission planning as the objective is often to maximize its the scientific return, by visiting multiple bodies in the solar system.

- In this work, the surrogate models were only used to try to approximate fitness vectors from the respective decision vector. Furthermore, they were only trained with decision and fitness vector pairs that were yielded by SNOPT after it was ran for local optimization. Perhaps, it is possible to successfully use the surrogate models in a different way, such as to approximate the decision vector that SNOPT would output from another decision vector that would have been provided to it as an initial guess. If done successfully, it would reduce a large part of the computational effort, as it would not be necessary to run SNOPT as often during the optimization process.

- Although a statistical analysis was done on the results, it would be interesting to have an analysis focused on visualization, from a data analytics perspective. As an example, visualizing how the solutions found by each surrogate model are distributed over the decision space, for different optimization problems may result in it being easier to choose a particular model for a particular problem, based on the information extracted from the data.

- As it was mentioned in the limitations, this study does not assess the change in computational effort introduced by the surrogate models due to the simulations being ran in a server with variable load. In addition to that, the run time is also highly dependent on the implementation, with it being possible for two different implementations having their run times multiple orders of magnitude apart. After settling on particular implementations of the surrogate models, optimization algorithms, and astrodynamics models, a thorough analysis of the run time should be done to verify if, for the same computational effort, it is possible to obtain better solutions with surrogate-assisted optimization. Naturally, it is important for the environment in which the simulations are

run to be the similar across all runs. This includes monitoring some aspects during the process to assure that they lie within an acceptable range. Such aspects include the computational load (system only working on simulations, without other concurrent tasks) and the temperature of the room that the system is in. A run time and space complexity analysis should also be performed, as a means of comparison that is independent on the hardware, being solely derived from the implementation.

A

Complementary Information on the Chosen Methods

A.1. GRNN

Proposed by Specht [34], the GRNN is a memory-based neural network that provides estimates of continuous variables and can, in theory, converge to any underlying regression surface. GRNNs are one-pass learning algorithm, as opposed to other neural networks that commonly use backpropagation, which results in a cheap approximation that is suitable for dealing with expensive optimization problems. The properties of a GRNN are perfect to have it as the global surrogate, smoothing out the fitness landscape and benefiting from the blessing of uncertainty. Assuming that $g(\mathbf{w}, z)$ is the known joint continuous probability density function of a vector random variable, $\mathbf{w}$, and a scalar random variable, z. Let $\mathbf{W}$ correspond to a particular measured value of the random variable $\mathbf{w}$, the regression of z on $\mathbf{W}$ is given by

$$E[z|\mathbf{W}] = \frac{\int_{-\infty}^{+\infty} z g(\mathbf{W}, z) dz}{\int_{-\infty}^{+\infty} g(\mathbf{W}, z) dz}. \tag{A.1}$$

When the density $g(\mathbf{w}, z)$ is not known, it can be estimated from a sample of observations of $\mathbf{w}$ and z, $\mathbf{W}_i$ and Z_i, respectively. Let N_s be the number of samples, D_in the dimension of $\mathbf{w}$, and σ a user-defined smoothing parameter, the estimation of the joint probability density function, $\hat{g}(\mathbf{W}, Z)$, can be done by the nonparametric estimators proposed by Parzen [35]:

$$\begin{aligned}\hat{g}(\mathbf{W}, Z) = &\frac{1}{2\pi^{(D_\mathrm{in}+1)/2} \cdot \sigma^{(D_\mathrm{in}+1)}} \\ &\cdot \frac{1}{N_\mathrm{s}} \sum_{i=1}^{N_\mathrm{s}} \exp\left[\frac{(\mathbf{W}-\mathbf{W}_i)^T \cdot (\mathbf{W}-\mathbf{W}_i)}{2\sigma^2}\right] \exp\left[\frac{(Z-Z_i)^2}{2\sigma^2}\right].\end{aligned} \tag{A.2}$$

By combining Equations (A.1) and (A.2), the estimation of Z, $\widehat{Z}$, for a particular $\mathbf{W}$ can be computed as follows:

$$\widehat{Z}(\mathbf{W}) = \frac{\sum_{i=1}^{N_\mathrm{s}} Z_i \exp\left(-\frac{\|\mathbf{W}-\mathbf{W}_i\|^2}{2\sigma^2}\right)}{\sum_{i=1}^{N_\mathrm{s}} \exp\left(-\frac{\|\mathbf{W}-\mathbf{W}_i\|^2}{2\sigma^2}\right)}, \tag{A.3}$$

where $\|\cdot\|$ denotes the standard Euclidean norm. Therefore, a set of decision vectors $\mathbf{x}_i$ and their corresponding ΔV_i can be used to approximate the ΔV associated with a certain decision vector $\mathbf{x}$ through the equation above in the form of $\widehat{\Delta V}(\mathbf{x})$. Naturally, any of the constraints can be approximated in place of the total change in velocity.

A.2. RBFN Derivatives

A.2.1. First-Order Derivatives of the Gaussian Basis Function

The partial derivatives of the Gaussian basis function that are presented in the paper are derived for a basis function of the form:

$$\phi_G = \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma^2}\right), \tag{A.4}$$

which is the form that commonly found in literature. In `tudat-learn`, however, the Gaussian basis function was implemented as

$$\phi_G = \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{\sigma^2}\right), \tag{A.5}$$

where the factor 2 was removed, with the intent to simplify the implementation. In retrospect, a simpler result would have been obtained without removing the aforementioned factor, however, it was implemented in that way, so that is how it is presented here.

The first-order partial derivatives become slightly different, as presented below:

$$\begin{aligned}\frac{\partial \phi_G}{\partial x_j}(\|\mathbf{x}-\mathbf{c}_i\|) &= \frac{\partial}{\partial x_j}\exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{\sigma^2}\right)\\ &= \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{\sigma^2}\right)\cdot\frac{\partial}{\partial x_j}\left[-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{\sigma^2}\right]\\ &= \phi_G(\|\mathbf{x}-\mathbf{c}_i\|)\cdot\left[-\frac{2\cdot(x_j-c_{ij})}{\sigma^2}\right].\end{aligned} \tag{A.6}$$

A.2.2. Second-Order Derivatives

Even though SNOPT does not make use of user-provided second-order derivatives, other local optimization algorithms can use them, hence they were implemented as well. The Hessian matrix H of the function $\hat{f}$ which approximates f is defined as follows:

$$H_{\hat{f}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 \hat{f}}{\partial x_1 \partial x_1}(\mathbf{x}) & \cdots & \frac{\partial^2 \hat{f}}{\partial x_1 \partial x_{D_I}}(\mathbf{x}), \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 \hat{f}}{\partial x_{D_I} \partial x_1}(\mathbf{x}) & \cdots & \frac{\partial^2 \hat{f}}{\partial x_{D_I} \partial x_{D_I}}(\mathbf{x}) \end{bmatrix}, \tag{A.7}$$

with the second-order partial derivatives being presented below:

$$\begin{aligned}\frac{\partial^2 \hat{f}}{\partial x_k \partial x_j}(\mathbf{x}) &= \frac{\partial}{\partial x_k}\left(\sum_{i=1}^{N_c} l_i \frac{\partial \phi}{\partial x_j}(\|\mathbf{x}-\mathbf{c}_i\|) + p_j\right)\\ &= \sum_{i=1}^{N_c} l_i \frac{\partial^2 \phi}{\partial x_k \partial x_j}(\|\mathbf{x}-\mathbf{c}_i\|).\end{aligned} \tag{A.8}$$

The second-order derivatives of the radial basis functions are also derived in the following expressions:

$$\begin{aligned}\frac{\partial^2 \phi_C}{\partial x_k \partial x_j}(\|\mathbf{x}-\mathbf{c}_i\|) &= \frac{\partial}{\partial x_k}\left(3\cdot(x_j-c_{ij})\cdot\|\mathbf{x}-\mathbf{c}_i\|\right)\\ &= 3\cdot(x_j-c_{ij})\cdot\frac{\partial}{\partial x_k}(\|\mathbf{x}-\mathbf{c}_i\|)+\|\mathbf{x}-\mathbf{c}_i\|\cdot\frac{\partial}{\partial x_k}\left(3\cdot(x_j-c_{ij})\right)\\ &= 3\cdot(x_j-c_{ij})\cdot\frac{1}{2}\cdot 2\frac{(x_k-c_{ik})}{\|\mathbf{x}-\mathbf{c}_i\|}+\begin{cases}3\cdot\|\mathbf{x}-\mathbf{c}_i\|, & \text{if } i=j,\\ 0, & \text{if } i\neq j\end{cases}\\ &= \frac{3(x_j-c_{ij})(x_k-c_{ik})}{\|\mathbf{x}-\mathbf{c}_i\|}+\begin{cases}3\cdot\|\mathbf{x}-\mathbf{c}_i\|, & \text{if } i=j,\\ 0, & \text{if } i\neq j\end{cases},\end{aligned} \tag{A.9}$$

$$\begin{aligned}\frac{\partial^2 \phi_G}{\partial x_k \partial x_j}(\|\mathbf{x}-\mathbf{c}_i\|) &= \frac{\partial}{\partial x_k}\left(\phi_G(\|\mathbf{x}-\mathbf{c}_i\|)\cdot\left[-\frac{2\cdot(x_j-c_{ij})}{\sigma^2}\right]\right)\\ &= \left[-\frac{2\cdot(x_j-c_{ij})}{\sigma^2}\right]\cdot\frac{\partial\phi_G}{\partial x_k}(\|\mathbf{x}-\mathbf{c}_i\|)+\phi_G(\|\mathbf{x}-\mathbf{c}_i\|)\cdot\frac{\partial}{\partial x_k}\left[-\frac{2\cdot(x_j-c_{ij})}{\sigma^2}\right]\\ &= \left[-\frac{2\cdot(x_j-c_{ij})}{\sigma^2}\right]\cdot\phi_G(\|\mathbf{x}-\mathbf{c}_i\|)\cdot\left[-\frac{2\cdot(x_k-c_{ik})}{\sigma^2}\right]+\begin{cases}\phi_G(\|\mathbf{x}-\mathbf{c}_i\|)\cdot\left(\frac{2}{\sigma^2}\right), & \text{if } i=j,\\ 0, & \text{if } i\neq j\end{cases}\\ &= \frac{4}{\sigma^4}\cdot(x_j-c_{ij})\cdot(x_k-c_{ik})\cdot\phi_G(\|\mathbf{x}-\mathbf{c}_i\|)+\begin{cases}\phi_G(\|\mathbf{x}-\mathbf{c}_i\|)\cdot\left(\frac{2}{\sigma^2}\right), & \text{if } i=j,\\ 0, & \text{if } i\neq j\end{cases}.\end{aligned} \tag{A.10}$$

A.3. Cross-Validation

Cross-validation is a statistical method used to evaluate and compare learning algorithms through the division of data in two segments: one used to train a model and the other used to validate it. The most common form of cross-validation is k-fold cross-validation, in which the training and validation sets must cross over in successive rounds in a way such that each instance of data has a chance to be validated against [36].

In k-fold cross validation, which is the method used in this work, the dataset is divided in k portions, called folds. One fold is chosen as the validation set, and the classifier is trained on the remaining $k-1$ folds, with its performance being evaluated on the aforementioned single fold. The process is performed a total of k times, with each of the folds being the validation set at some point. By averaging the performance over the k folds, an accurate performance estimation on unseen data can be obtained [36].

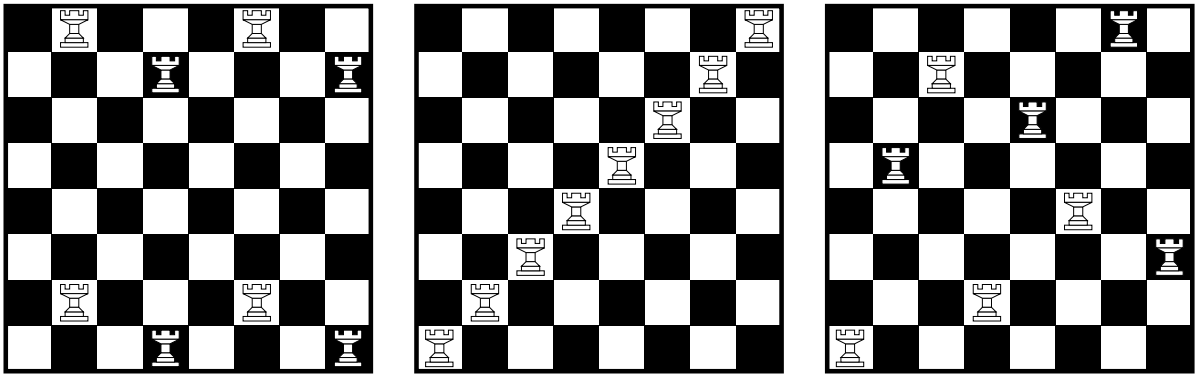
In this work, cross-validation is used to perform hyperparameter tuning for the surrogate models. Since one of the metrics that were used is the sorting accuracy, which relies on the validation set having 100 instances, a different number of folds was used for the different training dataset sizes, to ensure that the validation set stayed with the correct size. Table A.1 contains the size of the training and validation sets, as well as the number of folds for some of the dataset sizes used during the hyperparameter tuning procedure.

Table A.1: Training and validation set sizes as well as amount of folds for different dataset sizes used during cross-validation.

Dataset Size	Training Set Size	Validation Set Size	Number of Folds
200	100	100	2
300	200	100	3
500	400	100	5
1000	900	100	10
1500	1400	100	15
2000	1900	100	20
3000	2900	100	30

A.4. Latin Hypercube Sampling

One of the most common [26–28, 31, 37, 38] ways to attempt to generate a uniform population is Latin Hypercube Sampling (LHS). It was first introduced by McKay et al. [39] with sound mathematical foundation, but it can be described in simple terms. One starts by dividing the range of each decision variable in N_P intervals, where N_P corresponds to the number of individuals in the population or the amount of points being sampled. Afterwards, for each of the decision variables, an interval is selected and a value is sampled from that interval. The process is repeated for each of the individuals taking into account that each interval can only be selected once. A good analogy for the two-dimensional Latin Hypercube is thinking about the N_P intervals as rooks placed on an N_P by N_P chess board without threatening each other, as it can be seen in Figures A.1 (b) and (c) [40].



(a) (b) (c)

Figure A.1: Chessboard-Rook LHS analogy: (a) Does not represent a Latin Hypercube as the rooks are threatening each other; (b) Badly distributed Latin Hypercube as there is a correlation of 1 between the two dimensions; (c) Well distributed Latin Hypercube [40].

A.5. Monotonic Basin Hopping

In this section, the description of Monotonic Basin Hopping according to Yam et al.[15] is presented, with slight modifications being made to better suit the problem at hand. Algorithm 1 contains the description of the routine, which requires further definition of four procedures:

1. `g( )` is a procedure that randomly generates a decision vector uniformly drawn from a box limited by the problem bounds.
2. `s(x)` is a procedure that, given a decision vector `x`, computes a local minimizer of the fitness function, using `x` as an initial guess. Naturally, SNOPT is the local minimizer being used.
3. `Best(x, y)` is a procedure that, given two solutions, `x` and `y`, returns the best one according to a user-defined rule. In this work, it corresponds to feasibility rules enunciated in the paper.
4. `update(f_e)` is a procedure that, given a variable that keeps track of the function evaluations, `f_e`, updates it according to the function evaluations performed during procedure `s`.

Algorithm 1: Monotonic Basin Hopping as given by Yam et al.[15], modified to the needs of the problem at hand. The individual with the best fitness, according to the feasibility rules, is denoted by `x_best`.

```
x_best = g( )
f_e = 0
while f_e < MAX_FUNCTION_EVALUATIONS do
    x = g( )
    x_s = s(x)
    update(f_e)
    k = 0
    while k < MAX_WITHOUT_IMPROVEMENT and function_evaluations <
     MAX_FUNCTION_EVALUATIONS do
        y = p(x_s)
        y_s = s(y)
        update(f_e)
        if Best(x_s, y_s) == y_s then
            x_s = y_s
            k = 0
        end
        else
            k = k + 1
        end
    end
    x_best = Best(x_best, x_s)
end
```

A.6. Wilcoxon Rank-Sum Test

The Wilcoxon rank-sum test [41], sometimes also called Mann-Whitney U test [42] is a frequentist, non-parametric test [43] that was created to compare outcomes between two independent groups [40]. In this test, samples are drawn from two populations and ranked according to their values. In this work, the samples correspond to all the feasible individuals found during the 20 independent runs from the various models. The populations comprise all the feasible individuals that would have been found if the simulations had been run with an infinite amount of random seeds, for each of the models. The null and alternative hypotheses $\mathcal{H}_0$ and $\mathcal{H}_1$, respectively, are as follows [42]:

- $\mathcal{H}_0$: The two populations are equal (The performance of the two models is the same).
- $\mathcal{H}_1$: The two populations are not equal (The performance of the two models is not the same).

After drawing the samples, they are ranked according to their values. For instance, in the Earth-Mars example problem, the lower ΔV values would come first and the larger ones would come last, that is, the best solution would have a rank equal to 1, while the worst would have a rank equal to the total number of samples. In case of a tie, the mean rank for the tied values is given instead. Let R_1 and R_2 correspond to the sum of the ranks given to populations 1 and 2, respectively, the statistic U is given by the lower value between U_1 and U_2, which are given by the following expressions [42]:

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - T_1 \tag{A.11}$$

$$U_2 = n_2 n_1 + \frac{n_2(n_2+1)}{2} - T_2 \tag{A.12}$$

where n_1 and n_2 correspond to the number of samples drawn from each population.

As it was mentioned above, the statistic U has the same value as the lower value between U_1 and U_2. With this, and after selecting a level of significance α, which is often set to 0.05 [26, 27, 29, 30], it is possible to know if $\mathcal{H}_0$ is rejected by comparing U to the critical value, which depends on α, n_1 and

n_2. If the critical value is larger than or equal to U, there is statistically significant evidence at α to reject the null hypothesis. Said critical value is tabulated[1] for common significance levels.

Regarding the optimization algorithms or the different surrogate models, if the null hypothesis is not rejected, it means that their performance is similar. In case it is rejected, the best performing algorithm is the one whose U value is higher. Finally, the rank-sum test was conducted using `scipy`'s `scipy.stats.ranksums` implementation.

A.7. Plotting the Mean Constraint Violation and Mean Feasible Objective

Among the results presented in the paper, Figures 2.3 and 2.4 contain plots of the mean constraint violation and mean feasible objective, respectively, over the amount of fitness function evaluations. These means are computed over the 80 individuals in the population and then over the 20 independent runs. The former is simple to conduct, one just needs to compute the intended average over the population after every time that the population is saved to a file. However, the latter is more laborious.

The issue is that whenever SNOPT is used to optimize a problem with a certain initial guess, it needs to compute the fitness function multiple times until it either finds a solution or fails to converge. That amount of fitness function evaluations is very rarely the same between SNOPT optimization calls with different initial guesses. Therefore, for a certain amount of impulsive shots, and a certain optimization model, it is virtually impossible for the 20 independent runs to have their populations stored at the same amounts of fitness function evaluations. This makes it impossible to find a true average over the 20 independent runs.

The alternative, which was what was done here, is to provide something similar to a running mean. To do so, a **min-heap** [44] data structure such as the one in Figure A.2 is used. This data structure is a binary tree having all of its levels, except possibly the lowest, completely filled. Furthermore, a min-heap respects the property that the lowest value is at its root and the property that each of its sub-trees are min-heaps as well.

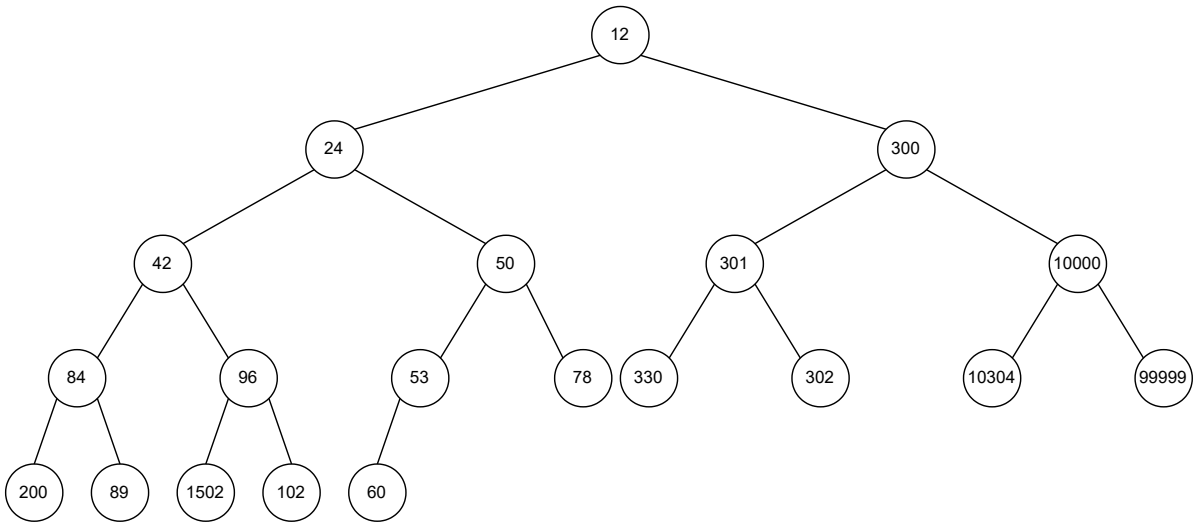


Figure A.2: A min-heap with 20 nodes.

The amount of fitness function evaluations after the first population has been saved is then pushed onto the heap, for each of the 20 independent runs. The nature of the heap guarantees that its root will contain the lowest number of function evaluations from all the 20 independent runs. That number is popped from the heap, and the corresponding constraint or feasible objective values are used to update the running mean, which naturally starts by being an average of a single simulation. Afterwards, the number of function evaluations of that same independent run after the second population is saved is pushed on to the heap and the process repeats. This way, as long as there are data points to process,

[1] `http://ocw.umb.edu/psychology/psych-270/other-materials/RelativeResourceManager.pdf` (last accessed on 10 June 2022)

the heap maintains a sorted ordering of the ”next amount of fitness function evaluations” for the 20 independent runs. As some simulations have all their populations processed, the amount of elements in the heap decreases, as no more amounts of fitness function evaluations are pushed onto it.

B

Verification

The basis functions, GRNNs, RBFNs and respective derivatives, as well as the linear scaler and the LHS were all implemented from scratch in `tudat-learn`. The software verification was done in the form of unit tests in the original version control repository. The results obtained with `tudat-learn` were not compared to results found in literature, but to implementations provided by other libraries. Some results were also implemented in a more efficient way, being verified with a less efficient, more verbose, implementation. Single-precision, floating-point numbers are used here as they are shorter, making it easier for them to be written down in a clear way. Differences in the results are highlighted in **bold**.

B.1. Radial Basis Functions

The input vector $\mathbf{x}$ and the center point $\mathbf{c}$ were generated using `Python`'s `random` module, always using a random seed set to 0. The values computed with `tudat-learn` are denoted by the `C++` subscript and are compared to a `Python` implementation. The gradients and Hessian matrices that were hard-coded in `tudat-learn` were evaluated and compared to symbolic differentiated expressions, through the use of the `sympy` package.

B.1.1. Inputs

$$\mathbf{x} = \begin{bmatrix} 0.84442185 \\ 0.75795440 \\ 0.42057158 \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} 0.25891675 \\ 0.51127472 \\ 0.40493414 \end{bmatrix}, \qquad \sigma = 0.7837989. \tag{B.1}$$

B.1.2. Gaussian Basis Function

The evaluation of the Gaussian basis function at the input $\mathbf{x}$ and center point $\mathbf{c}$, with the "true value", computed with `Python`, on the left, and the value obtained with `tudat-learn`, on the right:

$$(\phi_G)_{\text{python}}(\mathbf{x}, \mathbf{c}) = 0.51815949, \qquad (\phi_G)_{\text{C++}}(\mathbf{x}, \mathbf{c}) = 0.51815949. \tag{B.2}$$

The evaluation of the gradient of the Gaussian basis function at the input $\mathbf{x}$ and center point $\mathbf{c}$, with the "true value", computed with `Python`, on the left, and the value obtained with `tudat-learn`, on the right:

$$(\nabla\phi_G)_{\text{python}}(\mathbf{x}, \mathbf{c}) = \begin{bmatrix} -0.98767754 \\ -0.41611931 \\ -0.0263785\mathbf{1} \end{bmatrix}, \qquad (\nabla\phi_G)_{\text{C++}}(\mathbf{x}, \mathbf{c}) = \begin{bmatrix} -0.98767754 \\ -0.41611931 \\ -0.0263785\mathbf{0} \end{bmatrix}. \tag{B.3}$$

Finally, the evaluation of the Hessian matrix of the Gaussian basis function at the input $\mathbf{x}$ and center point $\mathbf{c}$, with the "true value", computed with `Python`, followed by the value obtained with `tudat-`

learn:

$$(H_{\phi_G})_{\texttt{python}}(\mathbf{x},\mathbf{c}) = \begin{bmatrix} 0.1957571\mathbf{7} & 0.79317606 & 0.0502807\mathbf{8} \\ 0.79317606 & -1.3527074\mathbf{7} & 0.0211838\mathbf{4} \\ 0.0502807\mathbf{8} & 0.02118384 & -1.68553831 \end{bmatrix}, \tag{B.4}$$

$$(H_{\phi_G})_{\texttt{C++}}(\mathbf{x},\mathbf{c}) = \begin{bmatrix} 0.1957571\mathbf{6} & 0.79317606 & 0.0502807\mathbf{6} \\ 0.79317606 & -1.3527074\mathbf{8} & 0.0211838\mathbf{3} \\ 0.0502807\mathbf{6} & 0.02118383 & -1.68553831 \end{bmatrix}. \tag{B.5}$$

It is considered that the Gaussian basis function is verified.

B.1.3. Cubic Basis Function

The evaluation of the cubic basis function at the input $\mathbf{x}$ and center point $\mathbf{c}$, with the ”true value”, computed with `Python`, on the left, and the value obtained with `tudat-learn`, on the right:

$$(\phi_C)_{\texttt{python}}(\mathbf{x},\mathbf{c}) = 0.25670216, \qquad (\phi_C)_{\texttt{C++}}(\mathbf{x},\mathbf{c}) = 0.25670216 \tag{B.6}$$

The evaluation of the gradient of the cubic basis function at the input $\mathbf{x}$ and center point $\mathbf{c}$, with the ”true value”, computed with `Python`, on the left, and the value obtained with `tudat-learn`, on the right:

$$(\nabla\phi_C)_{\texttt{python}}(\mathbf{x},\mathbf{c}) = \begin{bmatrix} 1.11633646 \\ 0.47032472 \\ 0.02981468 \end{bmatrix}, \qquad (\nabla\phi_C)_{\texttt{C++}}(\mathbf{x},\mathbf{c}) = \begin{bmatrix} 1.11633646 \\ 0.47032472 \\ 0.02981468 \end{bmatrix} \tag{B.7}$$

At last, the evaluation of the Hessian matrix of the cubic basis function at the input $\mathbf{x}$ and center point $\mathbf{c}$, with the ”true value”, computed with `Python`, followed by the value obtained with `tudat-learn`:

$$(H_{\phi_C})_{\texttt{python}}(\mathbf{x},\mathbf{c}) = \begin{bmatrix} 3.52484826 & 0.68177668 & 0.04321898 \\ 0.68177668 & 2.1938611\mathbf{9} & 0.0182086\mathbf{3} \\ 0.0432189\mathbf{8} & 0.0182086\mathbf{3} & 1.9077755\mathbf{2} \end{bmatrix} \tag{B.8}$$

$$(H_{\phi_C})_{\texttt{C++}}(\mathbf{x},\mathbf{c}) = \begin{bmatrix} 3.52484826 & 0.68177668 & 0.04321897 \\ 0.68177668 & 2.1938611\mathbf{8} & 0.0182086\mathbf{2} \\ 0.0432189\mathbf{7} & 0.0182086\mathbf{2} & 1.9077755\mathbf{1} \end{bmatrix} \tag{B.9}$$

It is considered that the cubic basis function is verified.

B.2. Generalized Regression Neural Network

As a highly parallelizable regression algorithm, GRNN is implemented in `tudat-learn` as a set of matrix multiplications, benefitting from the vectorization capabilities of the `Eigen` package. This implementation is denoted by the subscript `matrix`, and it is compared to an implementation that uses nested for-loops, by closely following Equation (A.3).

B.2.1. Fitting

A randomly generated training dataset using `Python` is used for the verification. In this case, a dataset of size 10 is used, where the center points or feature vectors are 7-dimensional, and the labels are 2-dimensional. The center points:

$$\mathbf{c} = \begin{bmatrix} 0.548814 & 0.715189 & 0.602763 & 0.544883 & 0.423655 & 0.645894 & 0.437587 \\ 0.891773 & 0.963663 & 0.383442 & 0.791725 & 0.528895 & 0.568045 & 0.925597 \\ 0.071036 & 0.087129 & 0.020218 & 0.832620 & 0.778157 & 0.870012 & 0.978618 \\ 0.799159 & 0.461479 & 0.780529 & 0.118274 & 0.639921 & 0.143353 & 0.944669 \\ 0.521848 & 0.414662 & 0.264556 & 0.774234 & 0.456150 & 0.568434 & 0.018790 \\ 0.617635 & 0.612096 & 0.616934 & 0.943748 & 0.681820 & 0.359508 & 0.437032 \\ 0.697631 & 0.060225 & 0.666767 & 0.670638 & 0.210383 & 0.128926 & 0.315428 \\ 0.363711 & 0.570197 & 0.438602 & 0.988374 & 0.102045 & 0.208877 & 0.161310 \\ 0.653108 & 0.253292 & 0.466311 & 0.244426 & 0.158970 & 0.110375 & 0.656330 \\ 0.138183 & 0.196582 & 0.368725 & 0.820993 & 0.097101 & 0.837945 & 0.096098 \end{bmatrix}, \tag{B.10}$$

and the corresponding labels, line by line:

$$f(\mathbf{c}) = \begin{bmatrix} 0.976459 & 0.468651 \\ 0.976761 & 0.604846 \\ 0.739264 & 0.039188 \\ 0.282807 & 0.120197 \\ 0.296140 & 0.118728 \\ 0.317983 & 0.414263 \\ 0.064147 & 0.692472 \\ 0.566601 & 0.265389 \\ 0.523248 & 0.093941 \\ 0.575946 & 0.929296 \end{bmatrix}. \tag{B.11}$$

The following value for the smoothing parameter σ was also randomly generated and used for verification purposes:

$$\sigma = 0.318569 \tag{B.12}$$

B.2.2. Evaluation

After being fitted to the dataset, the GRNN is evaluates the output of the following input values, also randomly generated:

$$\mathbf{x} = \begin{bmatrix} 0.667410 & 0.131798 & 0.716327 & 0.289406 & 0.183191 & 0.586513 & 0.020108 \\ 0.828940 & 0.004695 & 0.677817 & 0.270008 & 0.735194 & 0.962189 & 0.248753 \\ 0.576157 & 0.592042 & 0.572252 & 0.223082 & 0.952749 & 0.447125 & 0.846409 \end{bmatrix}. \tag{B.13}$$

Below are the values yielded by the GRNNs when trained on the dataset given by Equations (B.10) and (B.11) and evaluated on the inputs present in Equation (B.13). The values obtained with the implementation that uses nested for-loops are followed by the ones obtained with the implementation that uses matrix multiplications:

$$(\widehat{f})_{\texttt{nested}}(\mathbf{x}) = \begin{bmatrix} 0.212657 & 0.569603 \\ 0.712506 & 0.374801 \\ 0.327922 & 0.146603 \end{bmatrix}, \tag{B.14}$$

$$(\widehat{f})_{\texttt{matrix}}(\mathbf{x}) = \begin{bmatrix} 0.212657 & 0.569603 \\ 0.712506 & 0.374801 \\ 0.327922 & 0.146603 \end{bmatrix}. \tag{B.15}$$

It is considered that the GRNN is verified.

B.3. Radial Basis Function Networks

B.3.1. Fitting

Similarly to what happens with the GRNN, the RBFN is fitted to the dataset given by Equations (B.10) and (B.11), and a `nested` implementation is used to verify the `matrix` implemenation. The RBFN is fitted using both a Gaussian and a cubic basis function, with the former using the smoothing parameter in Equation (B.12).

Tables B.1 and B.2 have the coefficients l_i, p_0, and p_k, obtained for each of the output dimensions `out`. The first table concerns the Gaussian basis function, with the second concerning the cubic basis function. In each of those tables, the two columns on the left-hand side correspond to the implementation with nested for-loops, while the two columns on the right-hand side correspond to the implementation with matrix multiplications, used in `tudat-learn`. To solve the linear system inherent to RBFN, the `nested` implementation uses a the solver in the `numpy.linalg` package, while the `matrix` implementation uses `Eigen`'s `householderQr` decomposition:

Table B.1: Gaussian coefficients for both the `nested` and `matrix` implementations.

		nested		matrix	
	out	**1**	**2**	**1**	**2**
l_1		0.164707	-0.080970	0.164707	-0.080970
l_2		-0.043197	0.01682**8**	-0.043197	0.01682**9**
l_3		0.042483	-0.024992	0.042483	-0.024992
l_4		-0.113948	0.037186	-0.113948	0.037186
l_5		-0.02674**5**	0.00471**4**	-0.02674**4**	0.00471**3**
l_6		-0.003996	0.03213**5**	-0.003996	0.03213**6**
l_7		0.061702	-0.042552	0.061702	-0.042552
l_8		-0.010971	-0.010852	-0.010971	-0.010852
l_9		0.045096	0.00650**1**	0.045096	0.00650**0**
l_{10}		-0.115131	0.062003	-0.115131	0.062003
p_0		0.36056**7**	-1.45051**6**	0.36056**6**	-1.45051**2**
p_1		-0.397730	0.85825**8**	-0.397730	0.85825**7**
p_2		0.820341	-0.56787**2**	0.820341	-0.56787**1**
p_3		-0.17150**1**	1.42212**7**	-0.17150**0**	1.42212**4**
p_4		-0.15360**5**	0.88865**2**	-0.15360**4**	0.88865**1**
p_5		-0.674028	-1.052130	-0.674028	-1.052130
p_6		0.51327**3**	1.33752**0**	0.51327**4**	1.33751**9**
p_7		0.513274	0.39812**1**	0.513274	0.39812**0**

Table B.2: Cubic coefficients for both the `nested` and `matrix` implementations.

		nested		matrix	
	out	**1**	**2**	**1**	**2**
l_1		0.210345	-0.124317	0.210345	-0.124317
l_2		-0.047379	0.01714**4**	-0.047379	0.01714**5**
l_3		0.061512	-0.046473	0.061512	-0.046473
l_4		-0.112250	0.019956	-0.112250	0.019956
l_5		-0.019254	-0.009396	-0.019254	-0.009396
l_6		-0.058411	0.108840	-0.058411	0.108840
l_7		0.100389	-0.089431	0.100389	-0.089431
l_8		0.014692	-0.048700	0.014692	-0.048700
l_9		0.00693**6**	0.06652**2**	0.00693**5**	0.06652**1**
l_{10}		-0.156580	0.105854	-0.156580	0.105854
p_0		0.543433	-1.46438**4**	0.543433	-1.46438**5**
p_1		-0.472862	0.854898	-0.472862	0.854898
p_2		0.850252	-0.588**60**	0.850252	-0.588**59**
p_3		-0.088981	1.370274	-0.088981	1.370274
p_4		-0.243245	0.940585	-0.243245	0.940585
p_5		-0.722858	-1.0343**96**	-0.722858	-1.0343**95**
p_6		0.615800	1.2754**94**	0.615800	1.2754**93**
p_7		0.524939	0.3966**12**	0.524939	0.3966**13**

B.3.2. Evaluation

Below are the results yielded by the `nested` implementation of RBFN with a Gaussian basis function for the inputs points in Equation (B.13), followed by the results yielded by its `matrix` counterpart:

$$(\widehat{f_G})_{\texttt{nested}}(\mathbf{x}) = \begin{bmatrix} 0.224686 & 0.922516 \\ 0.003029 & 1.074597 \\ 0.502017 & -0.346183 \end{bmatrix}, \tag{B.16}$$

$$(\widehat{f_G})_{\texttt{matrix}}(\mathbf{x}) = \begin{bmatrix} 0.224686 & 0.922516 \\ 0.003029 & 1.074596 \\ 0.502017 & -0.346182 \end{bmatrix}. \tag{B.17}$$

And now, the results yielded by the `nested` implementation of RBFN with a cubic basis function for the inputs points in Equation (B.13), followed by the results yielded by its `matrix` counterpart:

$$(\widehat{f_C})_{\texttt{nested}}(\mathbf{x}) = \begin{bmatrix} 0.337597 & 0.844558 \\ 0.071872 & 1.007463 \\ 0.486969 & -0.348277 \end{bmatrix} \tag{B.18}$$

$$(\widehat{f_C})_{\texttt{nested}}(\mathbf{x}) = \begin{bmatrix} 0.337597 & 0.844557 \\ 0.071872 & 1.007463 \\ 0.486969 & -0.348276 \end{bmatrix} \tag{B.19}$$

B.3.3. Gradient

The gradients yielded by the `nested` implementation of RBFN with a Gaussian basis function evaluated at the points in Equation (B.13), followed by the results yielded by its `matrix` counterpart:

$$\nabla(\widehat{f_G})_{\texttt{nested}}(\mathbf{x}) = \begin{bmatrix} -0.397016 & 0.821603 & -0.171936 & -0.148307 & -0.672812 & 0.506353 & 0.519774 \\ 0.857843 & -0.568336 & 1.422509 & 0.884785 & -1.052923 & 1.341997 & 0.394420 \end{bmatrix}, \tag{B.20}$$

$$\nabla(\widehat{f_G})_{\texttt{matrix}}(\mathbf{x}) = \begin{bmatrix} -0.397016 & 0.821603 & -0.171936 & -0.148307 & -0.672812 & 0.506353 & 0.519774 \\ 0.857843 & -0.568336 & 1.422509 & 0.884785 & -1.052923 & 1.341997 & 0.394420 \end{bmatrix}. \tag{B.21}$$

And with a cubic basis function:

$$\nabla(\widehat{f_C})_{\texttt{nested}}(\mathbf{x}) = \begin{bmatrix} 0.365133 & 0.986004 & -0.024654 & -0.244258 & -0.697604 & 0.498648 & 0.617562 \\ 0.785769 & -0.653578 & 1.315828 & 0.927130 & -1.035478 & 1.340054 & 0.402368 \end{bmatrix} \tag{B.22}$$

$$\nabla(\widehat{f_C})_{\texttt{matrix}}(\mathbf{x}) = \begin{bmatrix} -0.365133 & 0.986004 & -0.024654 & -0.244258 & -0.697604 & 0.498648 & 0.617562 \\ 0.785769 & -0.653578 & 1.315828 & 0.927131 & -1.035478 & 1.340054 & 0.402368 \end{bmatrix} \tag{B.23}$$

B.3.4. Hessians

Finally, the Hessians are presented, evaluated at the same points, in Equation (B.13). Naturally, one Hessian matrix exists for each output dimension, there being two in this case. The output dimension is denoted by a 1 or 2 superscript. First, for the Gaussian basis function:

$$\left(H^1_{\widehat{f_G}}(\mathbf{x})\right)_{\texttt{nested}} = \begin{bmatrix} -0.022846 & -0.004908 & -0.005201 & 0.009077 & -0.002424 & -0.001289 & -0.001047 \\ -0.004908 & 0.012587 & -0.001683 & 0.001212 & 0.010069 & 0.011126 & 0.016673 \\ -0.005201 & -0.001683 & -0.022130 & 0.002080 & -0.000436 & 0.009541 & -0.010190 \\ 0.009077 & 0.001212 & 0.002080 & 0.016986 & 0.006265 & -0.050305 & 0.039650 \\ -0.002424 & 0.010069 & -0.000436 & 0.006265 & -0.013757 & -0.001410 & 0.011036 \\ -0.001289 & 0.011126 & 0.009541 & -0.050305 & -0.001410 & 0.044094 & -0.041008 \\ -0.001047 & 0.016673 & -0.010190 & 0.039650 & 0.011036 & -0.041008 & 0.029044 \end{bmatrix} \tag{B.24}$$

$$\left(H^1_{\hat{f}_G}(\mathbf{x})\right)_{\texttt{matrix}} = \begin{bmatrix} -0.022846 & -0.004908 & -0.005201 & 0.009077 & -0.002424 & -0.001289 & -0.001047 \\ -0.004908 & 0.012587 & -0.001683 & 0.001212 & 0.010069 & 0.011126 & 0.016673 \\ -0.005201 & -0.001683 & -0.022130 & 0.002080 & -0.000436 & 0.009541 & -0.010190 \\ 0.009077 & 0.001212 & 0.002080 & 0.016986 & 0.006265 & -0.050305 & 0.039650 \\ -0.002424 & 0.010069 & -0.000436 & 0.006265 & -0.013757 & -0.001410 & 0.011036 \\ -0.001289 & 0.011126 & 0.009541 & -0.050305 & -0.001410 & 0.044094 & -0.041008 \\ -0.001047 & 0.016673 & -0.010190 & 0.039650 & 0.011036 & -0.041008 & 0.029044 \end{bmatrix} \tag{B.25}$$

$$\left(H^2_{\hat{f}_G}(\mathbf{x})\right)_{\texttt{nested}} = \begin{bmatrix} 0.013963 & 0.002824 & 0.002359 & -0.004822 & 0.001493 & 0.001272 & -0.000113 \\ 0.002824 & -0.004298 & 0.001301 & -0.000177 & -0.005423 & -0.007785 & -0.006225 \\ 0.002359 & 0.001301 & 0.012581 & 0.001649 & 0.001311 & -0.004944 & 0.004198 \\ -0.004822 & -0.000177 & 0.001649 & -0.015315 & -0.004644 & 0.034923 & -0.025982 \\ 0.001493 & -0.005423 & 0.001311 & -0.004644 & 0.008853 & 0.001491 & -0.006001 \\ 0.001272 & -0.007785 & -0.004944 & 0.034923 & 0.001491 & -0.028368 & 0.024666 \\ -0.000113 & -0.006225 & 0.004198 & -0.025982 & -0.006001 & 0.024666 & -0.012396 \end{bmatrix} \tag{B.26}$$

$$\left(H^2_{\hat{f}_G}(\mathbf{x})\right)_{\texttt{matrix}} = \begin{bmatrix} 0.013963 & 0.002824 & 0.002359 & -0.004822 & 0.001493 & 0.001272 & -0.000113 \\ 0.002824 & -0.004298 & 0.001301 & -0.000177 & -0.005423 & -0.007785 & -0.006225 \\ 0.002359 & 0.001301 & 0.012581 & 0.001649 & 0.001311 & -0.004944 & 0.004198 \\ -0.004822 & -0.000177 & 0.001649 & -0.015315 & -0.004644 & 0.034923 & -0.025982 \\ 0.001493 & -0.005423 & 0.001311 & -0.004644 & 0.008853 & 0.001491 & -0.006001 \\ 0.001272 & -0.007785 & -0.004944 & 0.034923 & 0.001491 & -0.028368 & 0.024666 \\ -0.000113 & -0.006225 & 0.004198 & -0.025982 & -0.006001 & 0.024666 & -0.012396 \end{bmatrix} \tag{B.27}$$

And finally, for the cubic basis function:

$$\left(H^1_{\hat{f}_C}(\mathbf{x})\right)_{\texttt{nested}} = \begin{bmatrix} -0.281995 & -0.062577 & -0.037437 & 0.091527 & -0.104265 & 0.060146 & -0.132788 \\ -0.062577 & -0.047900 & -0.001114 & 0.007576 & -0.010361 & 0.083593 & -0.018292 \\ -0.037437 & -0.001114 & -0.197624 & 0.062519 & -0.064718 & 0.039502 & -0.092705 \\ 0.091527 & 0.007576 & 0.062519 & -0.277696 & 0.053671 & -0.126865 & 0.130489 \\ -0.104265 & -0.010361 & -0.064718 & 0.053671 & -0.210858 & 0.115168 & -0.034071 \\ 0.060146 & 0.083593 & 0.039502 & -0.126865 & 0.115168 & -0.156658 & 0.103230 \\ -0.132788 & -0.018292 & -0.092705 & 0.130489 & -0.034071 & 0.103230 & -0.239243 \end{bmatrix} \tag{B.28}$$

$$\left(H^1_{\hat{f}_C}(\mathbf{x})\right)_{\texttt{matrix}} = \begin{bmatrix} -0.281995 & -0.062577 & -0.037437 & 0.091527 & -0.104265 & 0.060146 & -0.132788 \\ -0.062577 & -0.047900 & -0.001114 & 0.007576 & -0.010361 & 0.083593 & -0.018292 \\ -0.037437 & -0.001114 & -0.197624 & 0.062519 & -0.064718 & 0.039502 & -0.092705 \\ 0.091527 & 0.007576 & 0.062519 & -0.277696 & 0.053671 & -0.126865 & 0.130489 \\ -0.104265 & -0.010361 & -0.064718 & 0.053671 & -0.210858 & 0.115168 & -0.034071 \\ 0.060146 & 0.083593 & 0.039502 & -0.126865 & 0.115168 & -0.156658 & 0.103230 \\ -0.132788 & -0.018292 & -0.092705 & 0.130489 & -0.034071 & 0.103230 & -0.239243 \end{bmatrix} \tag{B.29}$$

$$\left(H^2_{\hat{f}_C}(\mathbf{x})\right)_{\texttt{nested}} = \begin{bmatrix} 0.118457 & 0.039558 & 0.010134 & -0.037972 & 0.056766 & -0.039384 & 0.066058 \\ 0.039558 & -0.016906 & 0.009447 & 0.015074 & 0.029756 & -0.052556 & 0.015745 \\ 0.010134 & 0.009447 & 0.063250 & -0.000725 & 0.045274 & -0.001931 & 0.024348 \\ -0.037972 & 0.015074 & -0.000725 & 0.116810 & 0.028232 & 0.099444 & -0.054338 \\ 0.056766 & 0.029756 & 0.045274 & 0.028232 & 0.097426 & -0.070777 & -0.007832 \\ -0.039384 & -0.052556 & -0.001931 & 0.099444 & -0.070777 & 0.054743 & -0.088420 \\ 0.066058 & 0.015745 & 0.024348 & -0.054338 & -0.007832 & -0.088420 & 0.090468 \end{bmatrix} \tag{B.30}$$

$$\left(H^2_{\hat{f}_C}(\mathbf{x})\right)_{\texttt{matrix}} = \begin{bmatrix} 0.118457 & 0.039558 & 0.010134 & -0.037972 & 0.056766 & -0.039384 & 0.066058 \\ 0.039558 & -0.016906 & 0.009447 & 0.015074 & 0.029756 & -0.052556 & 0.015745 \\ 0.010134 & 0.009447 & 0.063249 & -0.000725 & 0.045274 & -0.001931 & 0.024348 \\ -0.037972 & 0.015074 & -0.000725 & 0.116810 & 0.028232 & 0.099444 & -0.054338 \\ 0.056766 & 0.029756 & 0.045274 & 0.028232 & 0.097426 & -0.070777 & -0.007831 \\ -0.039384 & -0.052556 & -0.001931 & 0.099444 & -0.070777 & 0.054743 & -0.088420 \\ 0.066058 & 0.015745 & 0.024348 & -0.054338 & -0.007831 & -0.088420 & 0.090468 \end{bmatrix} \tag{B.31}$$

It is considered that the RBFN is verified.

B.4. Linear Scaler

A linear scaler was also implemented in `tudat-learn` and tested with the help of the `numpy` package. The `C++` and `python` is used here again, and scaling is applied to the center points from the dataset in Equation (B.10), scaling every feature dimension between 0 and 1. Below are the results obtained with the `Python` version:

$$(\mathbf{c}_{\text{scaled}})_{\texttt{python}} = \begin{bmatrix} 0.582133 & 0.724968 & 0.766193 & 0.490299 & 0.479482 & 0.704967 & 0.436325 \\ 1.000000 & 1.000000 & 0.477731 & 0.773993 & 0.634007 & 0.602485 & 0.944760 \\ 0.000000 & 0.029780 & 0.000000 & 0.820993 & 1.000000 & 1.000000 & 1.000000 \\ 0.88715\mathbf{8} & 0.444141 & 1.000000 & 0.000000 & 0.797027 & 0.043413 & 0.964630 \\ 0.549277 & 0.392320 & 0.321366 & 0.753890 & 0.52719\mathbf{5} & 0.602997 & 0.000000 \\ 0.665986 & 0.610857 & 0.784831 & 0.948712 & 0.858548 & 0.327963 & 0.435747 \\ 0.763454 & 0.000000 & 0.850374 & 0.634828 & 0.166333 & 0.024421 & 0.309053 \\ 0.356600 & 0.564479 & 0.550280 & 1.000000 & 0.007259 & 0.129670 & 0.148485 \\ 0.70920\mathbf{6} & 0.213703 & 0.586724 & 0.144986 & 0.090843 & 0.000000 & 0.664223 \\ 0.081813 & 0.150931 & 0.458374 & 0.807630 & 0.000000 & 0.957786 & 0.080544 \end{bmatrix}, \tag{B.32}$$

and now the results obtained with `tudat-learn`'s implementation.

$$(\mathbf{c}_{\text{scaled}})_{\texttt{C++}} = \begin{bmatrix} 0.582133 & 0.724968 & 0.766193 & 0.490299 & 0.479482 & 0.704967 & 0.436325 \\ 1.000000 & 1.000000 & 0.477731 & 0.773993 & 0.634007 & 0.602485 & 0.944760 \\ 0.000000 & 0.029780 & 0.000000 & 0.820993 & 1.000000 & 1.000000 & 1.000000 \\ 0.88715\mathbf{7} & 0.444141 & 1.000000 & 0.000000 & 0.797027 & 0.043413 & 0.964630 \\ 0.549277 & 0.392320 & 0.321366 & 0.753890 & 0.52719\mathbf{4} & 0.602997 & 0.000000 \\ 0.665986 & 0.610857 & 0.784831 & 0.948712 & 0.858548 & 0.327963 & 0.435747 \\ 0.763454 & 0.000000 & 0.850374 & 0.634828 & 0.166333 & 0.024421 & 0.309053 \\ 0.356600 & 0.564479 & 0.550280 & 1.000000 & 0.007259 & 0.129670 & 0.148485 \\ 0.70920\mathbf{7} & 0.213703 & 0.586724 & 0.144986 & 0.090843 & 0.000000 & 0.664223 \\ 0.081813 & 0.150931 & 0.458374 & 0.807630 & 0.000000 & 0.957786 & 0.080544 \end{bmatrix} \tag{B.33}$$

It is considered that the linear scaler is verified.

B.5. Latin Hypercube Sampling

To verify the Latin hypercube sampler implemented in `tudat-learn`, three Latin hypercubes were generated for each of three, five, and ten samples, with the respective results being presented in Figures B.1, B.2, and B.3.

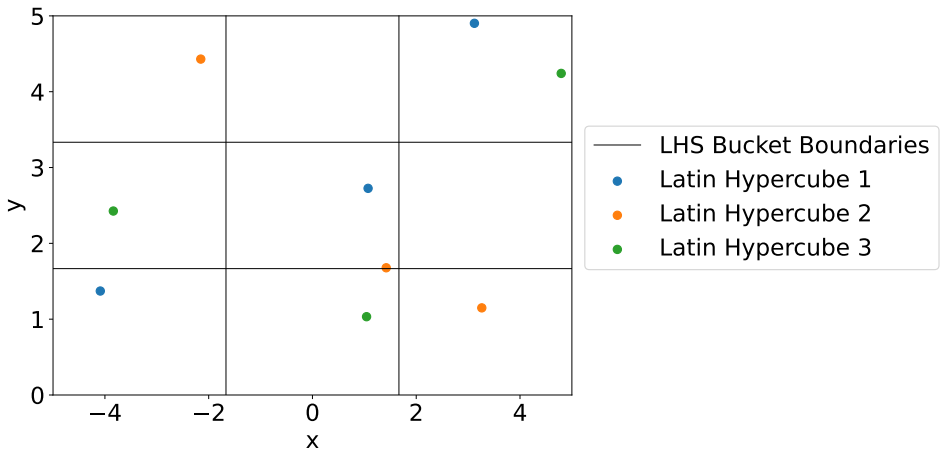


Figure B.1: Three Latin Hypercubes with three samples each.

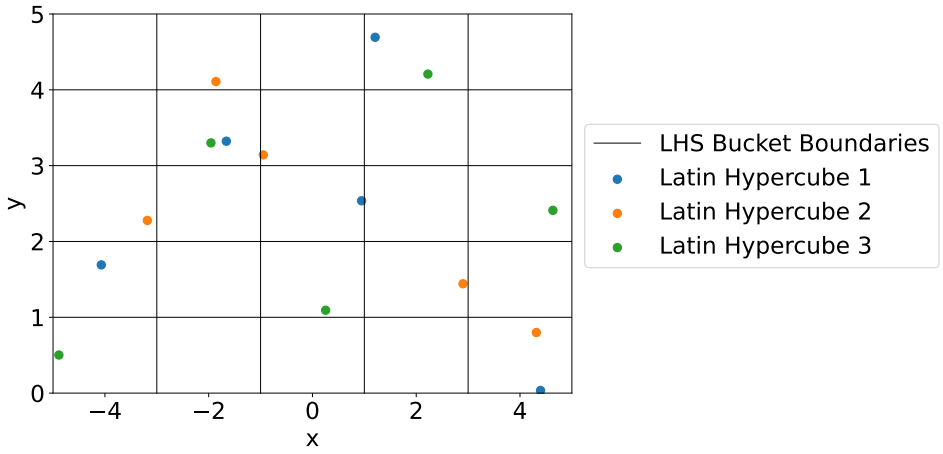


Figure B.2: Three Latin Hypercubes with five samples each.

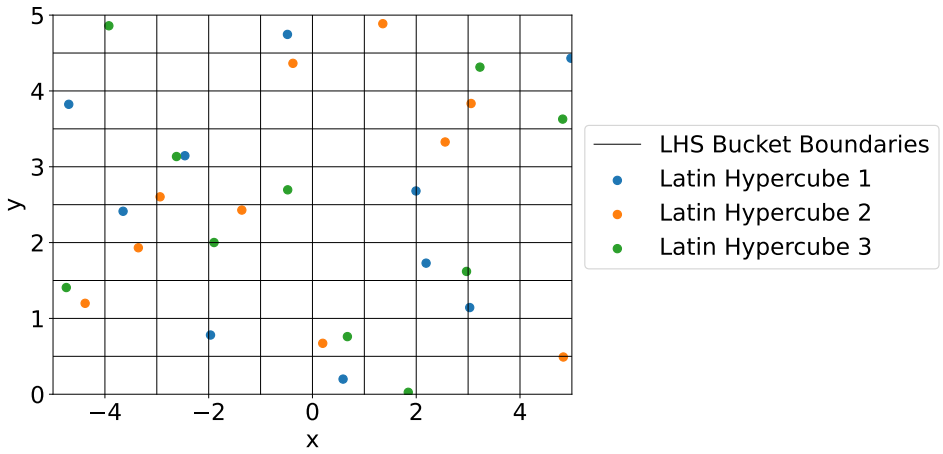


Figure B.3: Three Latin Hypercubes with ten samples each.

Since the sampled points form Latin hypercubes, that is, they do not violate the bucket boundaries according to the rules established in Section A.4, it is considered that the Latin hypercube sampler is verified.

B.6. Dynamical Model, Monotonic Basin Hopping, and Differential Evolution Operators

The dynamical model, as well as the queries for the states of the celestial bodies are taken from `Tudat` and either wrapped or used directly. As `Tudat` has a test suite, the tools are considered verified an no further verification was conducted.

Regarding the MBH algorithm and the DE operators that were taken from `pagmo`, they were not used directly or wrapped, essentially their code was copied and slightly changed to match what was needed in the optimization scripts. As `pagmo` also has a comprehensive test suite, and the changes were deemed small enough, it is also considered that these algorithms are verified.

C

Validation

As stated by Sargent [45], while model verification is defined as ”ensuring that the computer program of the computerized model and its implementation are correct”, model validation corresponds to ”substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model”. With that in mind, some words on the validity of the chosen models can be found in this chapter.

C.1. Dynamical Model

The Sims-Flanagan model, combined with an impulsive orbit insertion model and with the approximation that the spheres of influence of the departure and arrival bodies are negligible at the scale of an interplanetary trajectory is very common in literature [11, 13–15, 22]. The fact that said literature concerns preliminary mission design, which is what this work addresses, leads to the conclusion that the dynamical model is valid for the use it was given throughout this report.

C.2. Surrogate Functions

The cross-validation plots in Appendix B of the paper, namely the ones concerning the sorting accuracy make it clear that the chosen surrogate functions can distinguish good from bad candidate solutions for some hyperparameter combinations. Even though the error yielded by these approximations is not the lowest, their objective is to guide the optimization process in the direction of the optimal solutions, which they are able to do, as validated by the results of the paper. Therefore, the choice of the surrogate functions and the functions themselves are considered validated.

C.3. Optimization Algorithms

The triad of MBH, SNOPT, and DE is commonly used in literature with positive results [11, 13–15, 22, 46–49], which leads to the three of them being considered validated from the start.

The surrogate-assisted optimization proposed by Wang et al. [27], which was modified and used in this paper, is also considered verified due to the results obtained in this work.

D

tudat-learn

In this appendix, simplified versions of `tudat-learn`'s class diagrams are presented for better understanding of the library.

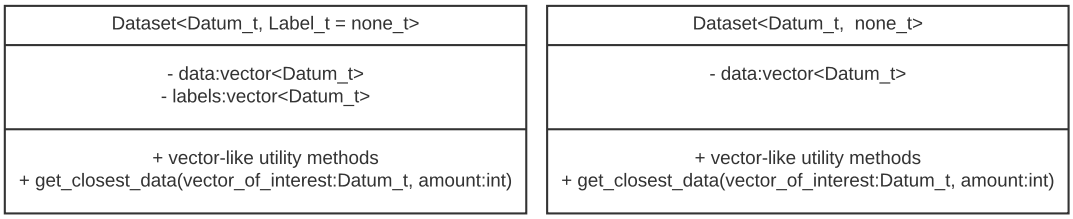


Figure D.1: Class diagram for the `Dataset` class in `tudat-learn`.

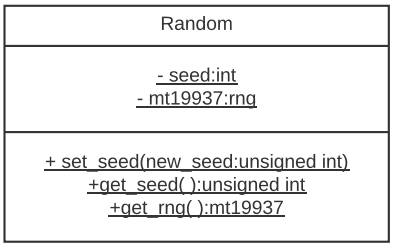


Figure D.2: Class diagram for the `Random` class in `tudat-learn`.

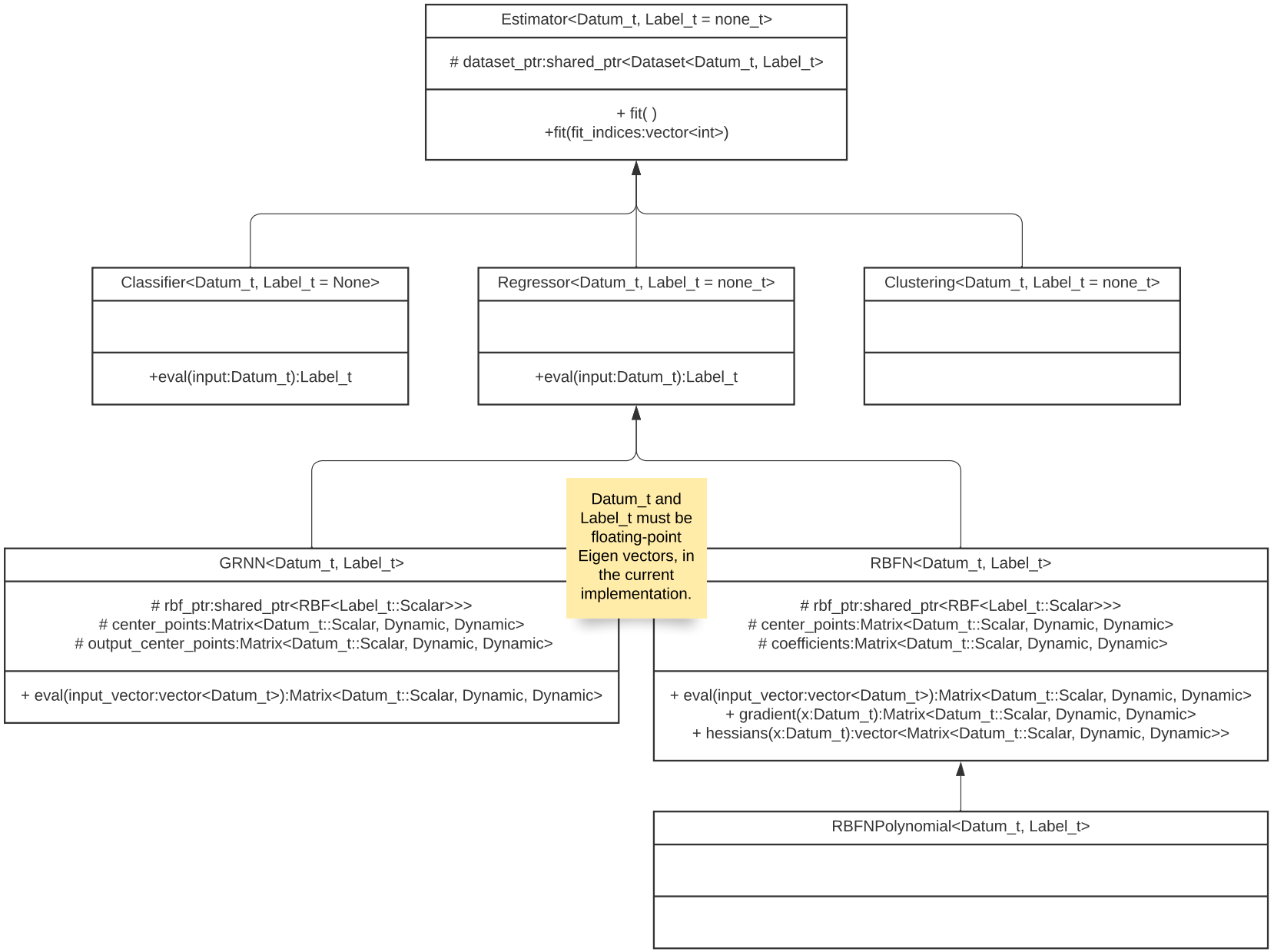


Figure D.3: Class diagram for the `Estimator` class and the classes that derive from it in `tudat-learn`.

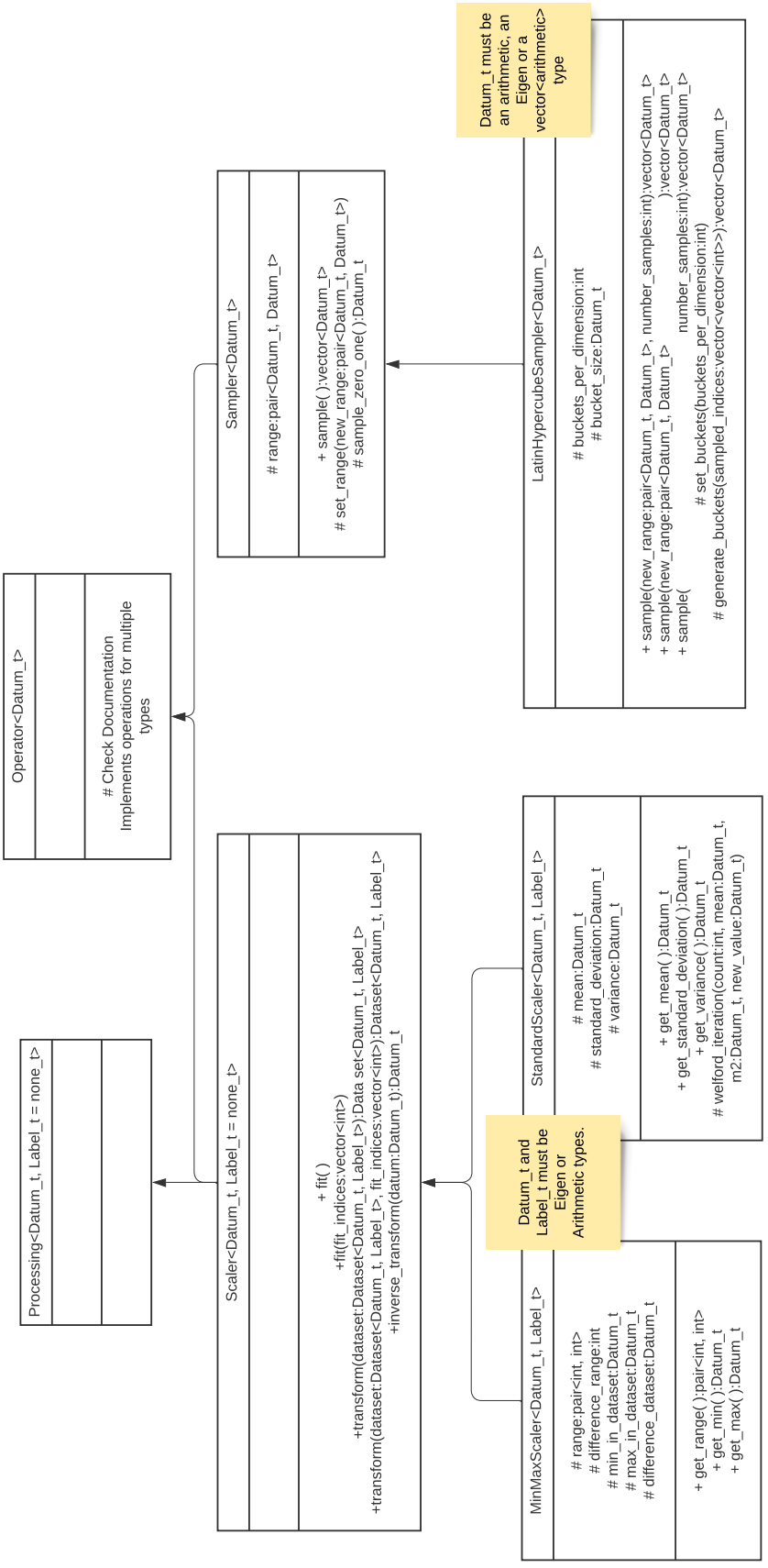


Figure D.4: Class diagram for the `Processing` and `Operator` classes and the classes that derive from them in `tudat-learn`.

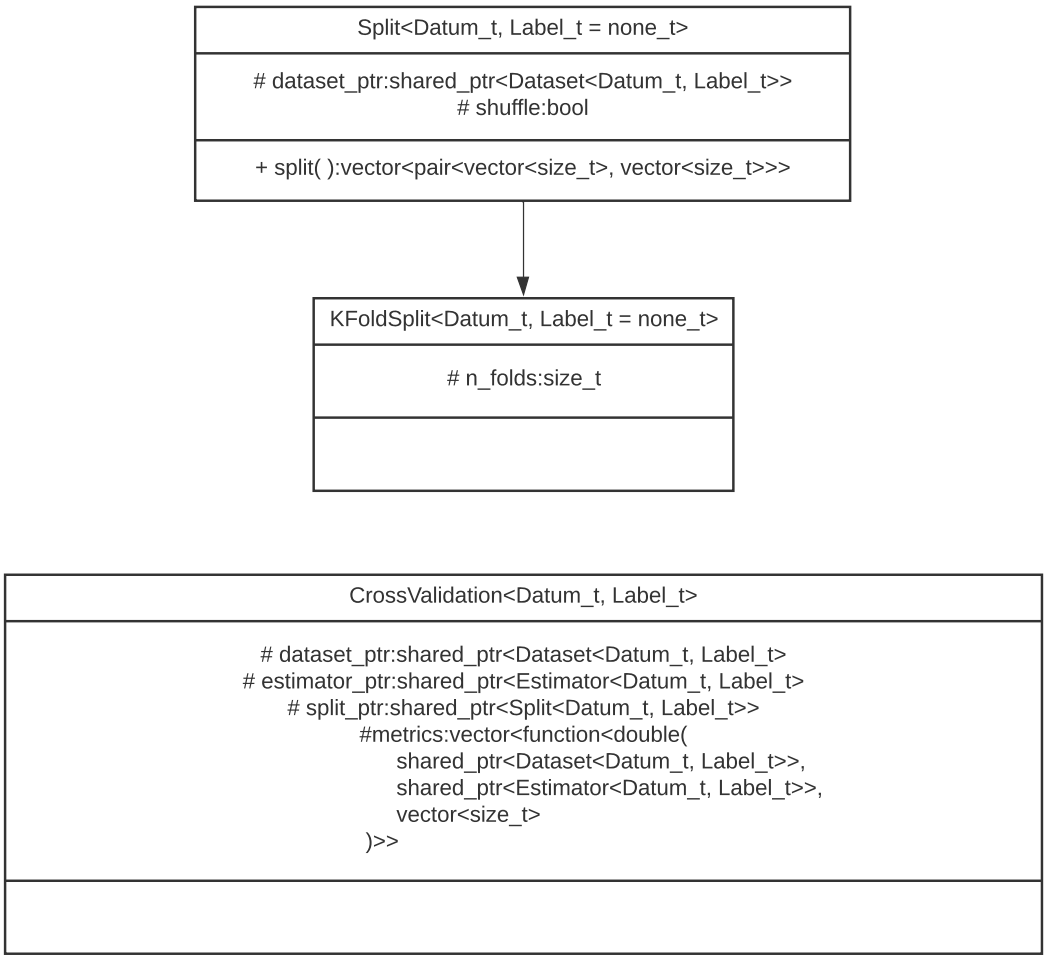


Figure D.5: Class diagram for the `Split` and `CrossValidation` classes and the classes that derive from them in `tudat-learn`.

List of References

[1] Marc D. Rayman, Pamela A. Chadbourne, Jeffery S. Culwell, and Steven N. Williams. Mission design for deep space 1: A low-thrust technology validation mission. *Acta Astronautica*, 45(4):381–388, 1999. ISSN 0094-5765. doi: https://doi.org/10.1016/S0094-5765(99)00157-5. URL https://www.sciencedirect.com/science/article/pii/S0094576599001575. Third IAA International Conference on Low-Cost Planetary Missions.

[2] Hitoshi Kuninaka. Microwave discharge ion engines onboard hayabusa asteroid explorer. *AIP Conference Proceedings*, 997:572–581, 04 2008. doi: 10.1063/1.2931929.

[3] Marc D Rayman, Thomas C Fraschetti, Carol A Raymond, and Christopher T Russell. Dawn: A mission in development for exploration of main belt asteroids vesta and ceres. *Acta Astronautica*, 58(11):605–616, 2006.

[4] Johannes Benkhoff, Jan Van Casteren, Hajime Hayakawa, Masaki Fujimoto, Harri Laakso, Mauro Novara, Paolo Ferri, Helen R Middleton, and Ruth Ziethe. Bepicolombo—comprehensive exploration of mercury: Mission overview and science goals. *Planetary and Space Science*, 58(1-2): 2–20, 2010.

[5] Onur Çelik, Diogene Alessandro Dei Tos, Takayuki Yamamoto, Naoya Ozaki, Yasuhiro Kawakatsu, and Chit Hong Yam. Multiple-Target Low-Thrust Interplanetary Trajectory of DESTINY+. *Journal of Spacecraft and Rockets*, pages 1–18, 2021.

[6] D J Gondelach. A hodographic-shaping method for low-thrust trajectory design. Master's thesis, Delft University of Technology, 2012.

[7] Anastassios E Petropoulos and James M Longuski. Shape-based algorithm for the automated design of low-thrust, gravity assist trajectories. *Journal of Spacecraft and Rockets*, 41(5):787–796, 2004.

[8] Anil V Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1):497–528, 2009.

[9] Harold W Kuhn and Albert W Tucker. Nonlinear programming. In *Traces and emergence of nonlinear programming*, pages 247–258. Springer, 2014.

[10] J Sims and S Flanagan. Preliminary design of low-thrust interplanetary missions. In *AAS/AIAA Astrodynamics Specialist Conference, AAS paper 99-338*, Girdwood, Alaska, August 1999.

[11] Donald H Ellison, Jacob A Englander, and Bruce A Conway. Robust global optimzation of low-thrust, multiple-flyby trajectories. In *AAS/AIAA Astrodynamics Specialist Conference, Hilton Head, SC*, 2013.

[12] Donald H Ellison, Jacob A Englander, Martin T Ozimek, and Bruce A Conway. Analytical partial derivative calculation of the sims-flanagan transcription match point constraints. In *AAS/AIAA Space-Flight Mechanics Meeting, Santa Fe, NM*, 2014.

[13] Jacob A Englander and Bruce A Conway. Automated solution of the low-thrust interplanetary trajectory problem. *Journal of Guidance, Control, and Dynamics*, 40(1):15–27, 2017.

[14] Chit Hong Yam, Francesco Biscani, and Dario Izzo. Global optimization of low-thrust trajectories via impulsive delta-v transcription. In *27th International Symposium on Space Technology and Science*, 2009.

[15] C H Yam, D D Lorenzo, and D Izzo. Low-thrust trajectory design as a constrained global optimization problem. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 225(11):1243–1251, 2011. doi: 10.1177/0954410011401686. URL https://doi.org/10.1177/0954410011401686.

[16] Martin T Ozimek, Jack F Riley, and Juan Arrieta. The low-thrust interplanetary explorer: A medium-fidelity algorithm for multi-gravity assist low-thrust trajectory optimization. In *AAS Space Flight Mechanics Conference, No. AAS*, pages 19–348, 2019.

[17] T McConaghy and James Longuski. Parameterization effects on convergence when optimizing a low-thrust trajectory with gravity assists. In *AIAA/AAs Astrodynamics Specialist Conference and Exhibit*, page 5403, 2004.

[18] Eugina D Mendez Ramos, Pranay Mishra, Stephen Edwards, and Dimitri Mavris. Response surface regressions for low-thrust interplanetary mission design. In *AIAA SPACE 2016*, page 5651. 2016.

[19] Christos Ampatzis and Dario Izzo. Machine learning techniques for approximation of objective functions in trajectory optimisation. In *Proceedings of the ijcai-09 workshop on artificial intelligence in space*, pages 1–6. Citeseer, 2009.

[20] Haiyang Li, Shiyu Chen, Dario Izzo, and Hexi Baoyin. Deep networks as approximators of optimal low-thrust and multi-impulse cost in multitarget missions. *Acta Astronautica*, 166:469–481, 2020.

[21] L Stubbig. Investigating the use of neural network surrogate models in the evolutionary optimization of interplanetary low-thrust trajectories. Master's thesis, Delft University of Technology, 2019.

[22] Veronica Saz Ulibarrena and Kevin Cowan. Low-thrust interplanetary trajectory optimization using pre-trained artificial neural network surrogates. AAS/AIAA Astrodynamics Specialist Conference, AAS Preprint, 2021.

[23] Daniel Hennes, Dario Izzo, and Damon Landau. Fast approximators for optimal low-thrust hops between main belt asteroids. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE, 2016.

[24] Alessio Mereta, Dario Izzo, and Alexander Wittig. Machine learning of optimal low-thrust transfers between near-earth objects. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 543–553. Springer, 2017.

[25] Bo Liu, Qingfu Zhang, and Georges GE Gielen. A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(2):180–192, 2013.

[26] Guodong Chen, Yong Li, Kai Zhang, Xiaoming Xue, Jian Wang, Qin Luo, Chuanjin Yao, and Jun Yao. Efficient hierarchical surrogate-assisted differential evolution for high-dimensional expensive optimization. *Information Sciences*, 542:228–246, 2021.

[27] Yong Wang, Da-Qing Yin, Shengxiang Yang, and Guangyong Sun. Global and local surrogate-assisted differential evolution for expensive constrained optimization problems with inequality constraints. *IEEE transactions on cybernetics*, 49(5):1642–1656, 2018.

[28] Chaoli Sun, Yaochu Jin, Jianchao Zeng, and Yang Yu. A two-layer surrogate-assisted particle swarm optimization algorithm. *Soft computing*, 19(6):1461–1475, 2015.

[29] Qunfeng Liu, Xunfeng Wu, Qiuzhen Lin, Junkai Ji, and Ka-Chun Wong. A novel surrogate-assisted evolutionary algorithm with an uncertainty grouping based infill criterion. *Swarm and Evolutionary Computation*, 60:100787, 2021.

[30] Mariana-Edith Miranda-Varela and Efrén Mezura-Montes. Constraint-handling techniques in surrogate-assisted evolutionary optimization. an empirical study. *Applied Soft Computing*, 73: 215–229, 2018.

[31] Xiwen Cai, Liang Gao, Xinyu Li, and Haobo Qiu. Surrogate-guided differential evolution algorithm for high dimensional expensive problems. *Swarm and Evolutionary Computation*, 48:288–311, 2019.

[32] Alan Díaz-Manríquez, Gregorio Toscano, and Carlos A Coello Coello. Comparison of metamodeling techniques in evolutionary algorithms. *Soft Computing*, 21(19):5647–5663, 2017.

[33] C Acton, N Bachman, J Diaz Del Rio, B Semenov, E Wright, and Y Yamamoto. Spice: A means for determining observation geometry. In *EPSC–DPS Joint Meeting*, volume 553, 2011.

[34] Donald F Specht. A general regression neural network. *IEEE transactions on neural networks*, 2 (6):568–576, 1991.

[35] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[36] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. *Encyclopedia of database systems*, 5:532–538, 2009.

[37] Junheung Park and Kyoung-Yun Kim. Meta-modeling using generalized regression neural network and particle swarm optimization. *Applied Soft Computing*, 51:354–369, 2017.

[38] Huachao Dong, Baowei Song, Zuomin Dong, and Peng Wang. Scgosr: Surrogate-based constrained global optimization using space reduction. *Applied Soft Computing*, 65:462–477, 2018.

[39] Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

[40] Francisco Andrade Castanheira. Online Surrogate Models for the Optimization of Interplanetary Low-Thrust Trajectories: Literature Study . Delft University of Technology, 2020.

[41] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[42] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[43] Jacinto Carrasco, Salvador García, MM Rueda, S Das, and Francisco Herrera. Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review. *Swarm and Evolutionary Computation*, 54:100665, 2020.

[44] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*, pages 2–145. MIT press, 2009.

[45] Robert G Sargent. Verification and validation of simulation models. In *Proceedings of the 2010 winter simulation conference*, pages 166–183. IEEE, 2010.

[46] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[47] David J Wales and Jonathan PK Doye. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, 1997.

[48] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2010.

[49] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.