Low-Thrust Spacecraft Transfers Using Physics-Informed Neural Networks

Tesi di Laurea Magistrale in
Space Engineering - Ingegneria Spaziale

Author: **Giuseppe Edoardo Addario**

Student ID: 964948
Advisor: Prof. Francesco Topputo
Co-advisors: Christian Hofmann, Alessandra Mannocchi,
Andrea Carlo Morelli, Mattia Pugliatti
Academic Year: 2022-23

To my grandmother Lella,
who inspired me to be a better person

To my father Mario,
an extraordinary fighter

Abstract

The objective of the research is to investigate recent developments in the field of deep learning to perform real-time guidance of low-thrust spacecraft. The methods based on deep learning meet the requirements for designing optimal trajectories on-board. In particular, the methods are suitable for the implementation on spacecraft computers given the limited computing power available. Thus, the ultimate goal of the research is to take a small step forward in the development of spacecraft capable of exploring the Solar System autonomously.

The research is based on physics-informed neural networks, a particular architecture developed to solve partial differential equations. The purpose of physics-informed neural networks is to increase the reliability and to overcome the data dependence of standard networks by exploiting the physics behind the problem in the training process of the networks.

The dissertation focuses on how the Hamilton-Jacobi-Bellman theory of optimal control can be exploited to build physics-informed neural networks. The end goal is to design a neurocontroller, i.e. a neural network-based controller, capable to guide the spacecraft to the final destination with reliability while minimizing specific performance indices, such as the mass of propellant required.

The work analyzes in detail an efficient method for creating databases of optimal trajectories used to train neural networks and in particular it extends the method to specific interplanetary transfers. The performance of physics-informed networks is then compared with that of standard networks in the context of Earth-Venus and Earth-Mars transfers, providing insights on why some architectures turn out to be more successful. In general, physics-informed networks prove to be more effective than standard networks, particularly in the context of fuel-optimal control problems. Nevertheless, the networks still lead to relatively large final position and velocity errors of the spacecraft with respect to the target, therefore, this issue needs to be further investigated in future research.

Keywords: physics-informed neural networks, real-time guidance, low-thrust spacecraft, interplanetary trajectories optimization

Sommario

L'obiettivo della ricerca è sfruttare i recenti sviluppi nel campo dell'apprendimento profondo per eseguire la guida in tempo reale di veicoli spaziali a bassa spinta. I metodi basati sull'apprendimento profondo soddisfano i requisiti necessari per progettare traiettorie ottimali a bordo. In particolare, i metodi sono adatti all'implementazione sui computer dei veicoli spaziali, data la limitata potenza di calcolo disponibile. L'obiettivo finale della ricerca è quindi quello di fare un piccolo passo avanti nello sviluppo di navicelle spaziali in grado di esplorare autonomamente il Sistema Solare.

La ricerca si basa sulle reti neurali informate dalla fisica, una particolare architettura sviluppata per risolvere equazioni differenziali parziali. L'obiettivo di queste reti neurali è quello di aumentare l'affidabilità e superare la dipendenza dai dati delle reti standard, sfruttando la fisica alla base del problema nel processo di addestramento delle reti.

La dissertazione si concentra su come la teoria Hamilton-Jacobi-Bellman del controllo ottimale possa essere sfruttata per costruire reti neurali informate dalla fisica. L'obiettivo finale è quello di progettare un neurocontrollore, cioè un controllore basato su reti neurali, in grado di guidare il veicolo spaziale verso la destinazione finale con affidabilità minimizzando specifici indici di prestazione, come la massa di propellente richiesta.

Il lavoro analizza in dettaglio un metodo efficiente per ottenere traiettorie ottimali utilizzate per addestrare le reti neurali e in particolare estende il metodo a specifici trasferimenti interplanetari. Le prestazioni delle reti informate dalla fisica vengono poi confrontate con quelle delle reti standard nel contesto dei trasferimenti Terra-Venere e Terra-Marte, analizzando i motivi per cui alcune architetture si rivelano più efficaci. In generale, le reti informate dalla fisica si dimostrano superiori a quelle standard, in particolare nei problemi di minimizzazione del carburante. Tuttavia, le reti si rivelando ancora imprecise in quanto si assiste a errori relativamente grandi di posizione e velocità finali del veicolo rispetto al bersaglio. Questo aspetto deve dunque essere ulteriormente analizzato in ricerche future.

Parole chiave: reti neurali informate dalla fisica, guida in tempo reale, veicoli spaziali a bassa spinta, ottimizzazione di traiettorie interplanetarie

Contents

1 | Introduction

Far out in the uncharted backwaters of the unfashionable end of the western spiral arm of the Galaxy lies a small unregarded yellow sun. [1]

Douglas Adams

1.1. Motivation

In the context of interplanetary low-thrust transfers, the current approach consists of designing the spacecraft trajectory during the mission planning phase. In the operational phase, the ground control center is responsible for tracking the reference trajectory by continuously calculating corrective maneuvers and by communicating the instructions to the spacecraft. This approach suffers from two main drawbacks. The first issue is the dependence of the spacecraft on the ground control center during the entire cruise in order to monitor the transfer. The second problem is the inability of the spacecraft to quickly react to unforeseen events: in these cases the trajectory must be recalculated on-ground and then uploaded to the spacecraft computer. Therefore, the goal of the current research is a paradigm shift toward fully autonomous spacecraft capable of recomputing the trajectories on-board. In this regard, the primary obstacle is the computational cost of the algorithms used to solve trajectory optimization problems. Thus, the studies focus on finding new techniques which can run on current on-board computers and they meet the following additional requirements. First, the algorithm must be able to find optimal, or at least suboptimal, trajectories, i.e. characterized, for example, by the shortest transfer time or by the least mass of propellant consumed. In addition, the algorithm must be reliable and robust, meaning it must converge and produce feasible trajectories in every situation faced by the spacecraft. In this regard, deep learning is a possible solution to the problem presented. Indeed, studies and computer simulations have shown that

[1]Douglas Adams, *The Hitchiker's Guide To The Galaxy* [2].

neural networks can be even more effective than traditional approaches in specific space-related problems [18, 42]. Despite the encouraging results, deep-learning-based methods still struggle to impose themselves. The main cause is the reliability of neural networks, that is, the impossibility of knowing a priori with certainty how networks behave when subjected to external inputs, especially if these inputs come from an environment that is particularly different from the training environment of the network. Indeed, reliability is a key concern in space and thus the issue must be overcome in order to promote the adoption of these techniques. Therefore, the question that prompted the present research is:

How can a reliable algorithm based on neural networks be developed to design low-thrust transfers of spacecraft?

The reliability problem can be addressed within the framework of physics-informed neural networks [37]. This recent paradigm has been applied to solve ordinary and partial differential equations by embedding in a smart way the equations during the training process of the networks. The procedure can be applied also in optimal control problems to leverage the physical laws governing the problem under consideration. The objective of the manuscript is therefore to preliminary investigate physics-informed neural networks in the context of spacecraft guidance. The research aims to initiate a path that leads to a general procedure for effectively addressing the complex problem of autonomous spacecraft guidance based on the physical description of the problem. The work specifically focuses on searching for answers to the following research question:

How can physics be exploited to build physics-informed neural networks for on-board guidance of spacecraft?

The work then intends to take a step forward in a preliminary attempt to evaluate the performance of physics-informed networks to understand the main strengths and limitations of these architectures. In this regard, the following research question arises naturally:

What is the performance of physics-informed neural networks compared with standard networks in the context of spacecraft guidance?

1.2. Neural Networks in Spacecraft Guidance

In this section, the objective is to indicate to the reader the most relevant works related to the application of neural networks to perform spacecraft guidance. To begin the overview, the work by Dachwald [11] is a successful example of deep learning applied to interplanetary trajectory optimization. In the study, neural networks and evolutionary algorithms are combined into an evolutionary neurocontroller to guide solar sails spacecraft by optimizing the total transfer time and the propellant mass consumed. In the research by Sánchez-Sánchez and Izzo [40], on the other hand, neural networks are exploited to produce optimal controls in real-time in the context of landing problems of quadcopters, spacecraft and rockets. In this study, the authors demonstrate that neural networks are able to replicate the simulated optimal landings. To continue, the work by Cheng et al. [8] focuses on minimum-time two-dimensional orbit transfer problems and in particular, the authors introduce an interesting methodology to enhance the performance of classical trajectory optimization solvers by relying on trained neural networks. In addition, the work presents an interesting cooperation strategy among multiple neural networks to improve the terminal guidance accuracy of interplanetary transfers. This technique is applied also in the context of fuel-optimal transfers by Yin et al. [46]. The work of Rubinsztejn et al. [38], instead, analyzes in depth the performance of neural networks in the context of fuel and energy-optimal transfers subjected to missed thrust events. The authors demonstrate that neural networks are particularly robust against these unexpected events. In contrast, the limitations of deep learning are particularly highlighted by Li et al. [27] who show that neural networks are not able to guide the spacecraft to the target orbits, in time-optimal problems, with accuracy. It must be emphasized that in all these works, standard neural networks are considered to solve the related problems. In fact, the earliest example of application of physics-informed neural networks is the work by Izzo et Öztürk [17] where neural networks are trained to approximate the value function for real-time guidance of spacecraft in fuel-optimal problems. It should be noted, however that the authors did not explicitly use the term physics-informed neural networks to refer to the architectures presented. Indeed, they claimed that the methodology was developed independently from the emerging field of physics-informed neural networks introduced by Raissi et al. [37] by expanding the original idea presented by Lagaris et al. [24]. In this context, Izzo et Öztürk have applied the same methodology also for time-optimal transfers of spacecraft subjected to constant acceleration [16]. To conclude the overview, physics-informed neural networks are exploited also in the work by Schiassi et al. [41] to address optimal planar orbit transfers. It is worth noting that the study follows a completely different direction with respect to the work of Izzo et Öztürk and in particular it relies on the theory of functional connections formulated by Mortari [34].

1.3. Manuscript Structure

The manuscript is organized as follows:

- **Chapter** 2 presents the mathematical formulation of neural networks and describes some of the algorithms used in the context of supervised learning. In addition, the chapter discusses and explores through an example the new paradigm of physics-informed neural networks.
- **Chapter** 3 introduces the mathematical theory of optimal control problems and summarizes the main theoretical results of Pontryagin's and Hamilton-Jacobi-Bellman principles of optimal control.
- **Chapter** 4 formalizes the structure of the optimal low-thrust transfer problem analyzed in the work and presents some practical techniques exploited to solve it.
- **Chapter** 5 addresses the problem of creating a database of optimal trajectories to train the neural networks and introduces the concept of trust band used to discuss the reliability of neural networks. In addition, the chapter describes the models of standard and physics-informed neural networks, with emphasis on the relevant differences between them.
- **Chapter** 6 presents the general methodology adopted to investigate the case studies and analyzes the results obtained in the context of an Earth to Venus and an Earth to Mars transfers. In conclusion, the chapter provides insights into the numerical implementation of the algorithms used in the study.
- **Chapter** 7 concludes the work by summarizing the main results and by presenting possible directions for future research.

2 | Physics-Informed Machine Learning

Deep Blue was only intelligent the way your programmable alarm clock is intelligent. Not that losing to a $10 million alarm clock made me feel any better. [1]

Garry Kasparov

2.1. Feedforward Neural Networks

Artificial Neural Networks or simply *Neural Networks* are essentially function approximators [12, 14]. Given a generic function $f := \mathbb{R}^m \to \mathbb{R}^n$, a neural network defines a function on the same domain and codomain, i.e. $\mathcal{N}(\cdot\,|\,\theta) := \mathbb{R}^m \to \mathbb{R}^n$, and learns the parameters $\theta \in \mathbb{R}^p$ which result in the best function approximation of $f(\cdot)$. Neural networks generally consist of an input layer, a number of hidden layers and an output layer. In feedforward neural networks the information flows from the input layer to the output layer without cycling. The layers are represented by maps of the type:

$$g^{[i]}(\cdot\,|\,\theta^{[i]}) := \mathbb{R}^{m_i} \to \mathbb{R}^{n_i} \qquad i = 1, ..., l-1 \tag{2.1}$$

where l denotes the total number of layers composing the network and $\theta^{[i]} \in \mathbb{R}^{p_i}$ are the trainable parameters associated with each layer. The neural network function is defined from the layer maps as follows:

$$\mathcal{N}(x\,|\,\theta) = (g^{[l-1]} \circ ... \circ g^{[1]})(x) \tag{2.2}$$

[1]In 1997, *IBM Deep Blue* beat world chess champion Garry Kasparov in a series of six games. It was the first time a computer proved to be superior to a chess grandmaster.

where $\theta = [\theta^{[1]}, ..., \theta^{[l-1]}]^T$ collects in a single vector all layers parameters. Note that the layers must have the correct number of inputs and outputs for the previous relationship to be valid. The layers are composed of computational units called neurons which receive an input and return a response which is governed by the neuron activation process. The activation process, from a mathematical perspective, is the evaluation of a nonlinear function, the neuron activation function, on the input of the neuron. The output of a layer $L^{[i]}$ composed of m neurons, $y^{[i]} \in \mathbb{R}^m$, is defined as the collection of the responses of all neurons in the layer:

$$y^{[i]} = [y_1^{[i]}, .., y_m^{[i]}]^T \tag{2.3}$$

where $y_j^{[i]} = \sigma_j^{[i]}(x_j^{[i]})$ is the output of the j-th neuron, $x_j^{[i]}$ is the input and $\sigma_j^{[i]} : \mathbb{R} \to \mathbb{R}$ denotes the neuron activation function. Let us consider the layers $L^{[i]}$ and $L^{[i+1]}$, composed of m and n neurons, respectively. The j-th neuron of $L^{[i+1]}$ receives as input a linear combination of the output of $L^{[i]}$:

$$x_j^{[i+1]} = \sum_{k=1}^{m} w_{jk}^{[i]} y_k^{[i]} + b_j^{[i]}, \qquad j = 1, .., n \tag{2.4}$$

where the parameters $w_{jk}^{[i]} \in \mathbb{R}$ are referred as weights and are associated with the connections between $L^{[i]}$ and the j-th neuron and the parameter $b_j^{[i]} \in \mathbb{R}$ is called bias and it is associated with the j-th neuron. A scheme of a feedforward neural network with a detailed view on the working principle of a neuron is reported in Figure 2.1.

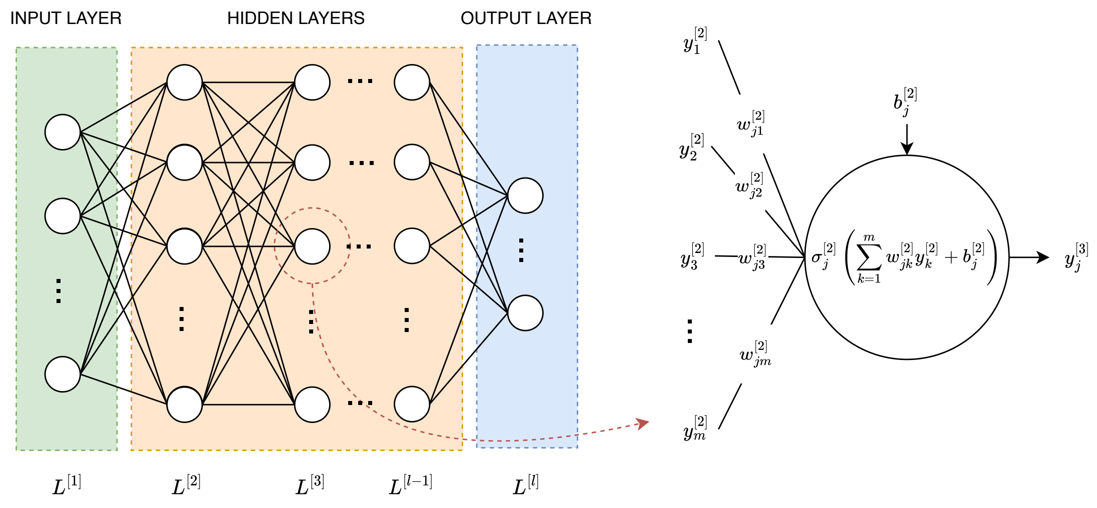


Figure 2.1: Schematic representation of a feedforward neural network.

The response of the layer $L^{[i+1]}$ can be compactly expressed in matrix form as follows:

$$y^{[i+1]} = \sigma^{[i]}(W^{[i]}y^{[i]} + b^{[i]}) \tag{2.5}$$

where $W^{[i]} \in \mathbb{R}^{n\times m}$ is a matrix whose rows are the weights associated with each neuron in the layer $L^{[i+1]}$ and $b^{[i]} \in \mathbb{R}^n$ is a vector whose components are the biases associated with each neuron in the layer $L^{[i+1]}$. The layer activation function, $\sigma^{[i]} := \mathbb{R}^n \to \mathbb{R}^n$, is simply:

$$\sigma^{[i]}(x) = [\sigma_1^{[i]}(x), ..., \sigma_n^{[i]}(x)]^T \tag{2.6}$$

where the same nonlinear function is usually used as the neuron activation function for all neurons in the same layer. The maps of the layer are thus defined as:

$$g^{[i]}(x \,|\, \theta^{[i]}) = \sigma^{[i]}(W^{[i]}x + b^{[i]}) \qquad i = 1, ..., l-1 \tag{2.7}$$

where the vector of layer parameters, $\theta^{[i]} \in \mathbb{R}^{(nm+n)}$, consists of the entries of the weights matrix, rearranged in vector form, and the components of the biases vector.

2.2. Supervised Learning

In *Supervised Learning*, neural networks are trained by exploiting a labelled dataset, i.e a collection of inputs and the corresponding correct outputs. In this regard, the training can be decomposed in a sequence of phases. In the first phase, the inputs contained in the dataset are fed to the network which in turn predicts the corresponding outputs. In the second phase, the predicted outputs are compared with the correct outputs trough an evaluation metric called the loss function. Finally, the network adjusts the parameters to minimize the loss function, that is, to find the best fit of the data. Let us consider a labeled dataset $\mathcal{D} := \{(x^{(j)}, y^{(j)})\}_{j=1}^d$ and let us define the following quantities:

$$X := [x^{(1)}, ..., x^{(d)}],\ Y := [y^{(1)}, ..., y^{(d)}],\ \hat{Y} := [\hat{y}^{(1)}, ..., \hat{y}^{(d)}] \tag{2.8}$$

where $X \in \mathbb{R}^{m\times d}$ collects all the inputs of the dataset, $Y \in \mathbb{R}^{n\times d}$ collects all the correct outputs of the dataset, $\hat{Y} := \mathcal{N}(X \,|\, \theta) \in \mathbb{R}^{n\times d}$ collects all the predicted outputs of the network and m and n denote the input and the output size of the network. The loss function, $l(\cdot \,|\, \mathcal{D}) := \mathbb{R}^p \to \mathbb{R}$, is defined as follows:

$$l(\theta \,|\, \mathcal{D}) = \text{metric}(\mathrm{Y}, \hat{\mathrm{Y}}) \tag{2.9}$$

Note that the metric in the previous expression can be defined in different ways, for example, it is possible to consider the Mean Square Error (MSE) or the Mean Absolute Error (MAE) as follows:

$$MSE(Y,\hat{Y}) = \frac{1}{n\,d}\sum_{i=1}^{d} \|y^{(i)} - \hat{y}^{(i)}\|_2^2 \tag{2.10}$$

$$MAE(Y,\hat{Y}) = \frac{1}{n\,d}\sum_{i=1}^{d} \|y^{(i)} - \hat{y}^{(i)}\|_1 \tag{2.11}$$

2.2.1. Gradient Descents Methods

The most popular algorithm to update the neural networks parameters during the training process is *Gradient Descent* [39]. In gradient descent, the loss function is minimized by taking repeated steps in the direction opposite to the gradient of the loss function. In fact, the aim is to update the parameters so that the loss function decreases, so the following must hold:

$$l(\bar{\theta} + hd) < l(\bar{\theta}) \tag{2.12}$$

where $h \in \mathbb{R}$ is a coefficient, $d \in \mathbb{R}^p$ is the direction of parameters update and $\bar{\theta} \in \mathbb{R}^p$ denotes the current parameters. This condition can be re-expressed by rearranging the terms and by taking the following limit:

$$\lim_{h\to 0} \frac{l(\bar{\theta} + hd) - l(\bar{\theta})}{h} < 0 \tag{2.13}$$

It can be noticed that the left-hand side of the expression is the directional derivative of the loss function, thus it must be $\nabla l(\bar{\theta})^T d < 0$. To satisfy this condition, the simplest choice is to select the direction of parameters update as follows:

$$d = -\nabla l(\bar{\theta}) \tag{2.14}$$

Therefore, in gradient descent, the rule to update the parameters can be expressed as:

$$\theta_{k+1} = \theta_k - \eta \nabla l(\theta_k) \tag{2.15}$$

where $\eta \in \mathbb{R}_{>0}$ is a coefficient which governs the size of step in the opposite direction of the gradient and it is called learning rate. It should be pointed out that a small learning rate leads to slow convergence, while a too large learning rate can cause oscillations around minima. It must be noticed that in this naive version of gradient descent all the parameters are updated with the same learning rate. The most popular variant of gradient descent is the ADAptive Moment estimation (ADAM) algorithm [22]. ADAM computes adaptive learning rates for each parameter and to this aim it considers exponentially decaying averages of past gradients as follows:

$$m_k = \beta_1 m_{k-1} + (1-\beta_1)\nabla l(\theta_k) \tag{2.16}$$

$$v_k = \beta_2 v_{k-1} + (1-\beta_2)\nabla l(\theta_k)^2 \tag{2.17}$$

where m_k and v_k are the biased estimates of the first and the second moments of the gradient and $\beta_1, \beta_2 \in [0,1)$ are tunable parameters. The estimates of the moments are then bias-corrected:

$$\hat{m}_k = \frac{m_k}{1-\beta_1^k} \tag{2.18}$$

$$\hat{v}_k = \frac{v_k}{1-\beta_2^k} \tag{2.19}$$

so that the ADAM parameters update rule become:

$$\theta_{k+1} = \theta_k - \frac{\eta}{\sqrt{\hat{v}_k}+\epsilon}\hat{m}_k \tag{2.20}$$

where ϵ is a small value used to avoid division by zero. The authors of the ADAM algorithm suggests the default values $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. An example of application of gradient descent is reported in Figure 2.2, where a loss function depending on two parameters is considered. It must be noticed that the direction of parameters update is orthogonal to the contour lines of the function, as per definition of gradient.

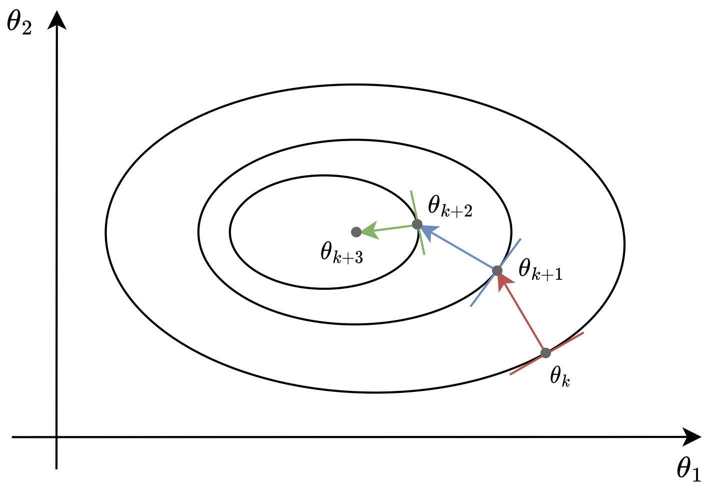


Figure 2.2: Gradient descent for a two-dimensional function.

2.2.2. Quasi-Newton Methods

In Newton's method, a second order Taylor expansion of the function is considered [5]:

$$l(\bar{\theta} + d) \approx l(\bar{\theta}) + \nabla l(\bar{\theta})^T d + \frac{1}{2} d^T H_l(\bar{\theta}) d \tag{2.21}$$

Let us introduce the following auxiliary function, $g := \mathbb{R}^p \to \mathbb{R}$, which is defined as the right-hand side of the previous expression:

$$g(d) = l(\bar{\theta}) + \nabla l(\bar{\theta})^T d + \frac{1}{2} d^T H_l(\bar{\theta}) d \tag{2.22}$$

The gradient of this function is:

$$\nabla g(d) = \nabla l(\bar{\theta}) + H_l(\bar{\theta}) d \tag{2.23}$$

In order to minimize the auxiliary function, the following necessary condition must hold $\nabla g(d) = 0$, and thus the following relation can be obtained:

$$d = -H_l(\bar{\theta})^{-1} \nabla l(\bar{\theta}) \tag{2.24}$$

In Newton's method, the rule to update the parameters can then be expressed as follows:

$$\theta_{k+1} = \theta_k - \alpha H_l(\theta_k)^{-1} \nabla l(\theta_k) \tag{2.25}$$

where $\alpha \in \mathbb{R}_{>0}$ is a coefficient which governs the step size. It should be noted that Newton's method is a second-order method, which means that the algorithm needs the Hessian of the function, whereas in gradient descent, a first-order method, only the gradient is required. From a computational point of view, the Hessian can be expensive to evaluate, hence the emergence of quasi-Newton methods. In quasi-Newton methods, the Hessian is approximated to speed up computations. The most popular quasi-Newton variant is the BFGS method from the the names of the creators: Broyden, Fletcher, Goldfarb and Shanno and its limited computer memory version L-BFGS. An example of application of Newton's method is reported in Figure 2.3 where a loss function depending on a single parameter is considered. It can be noticed that the function is first locally approximated by polynomials of second degree and then these polynomials are sequentially minimized.

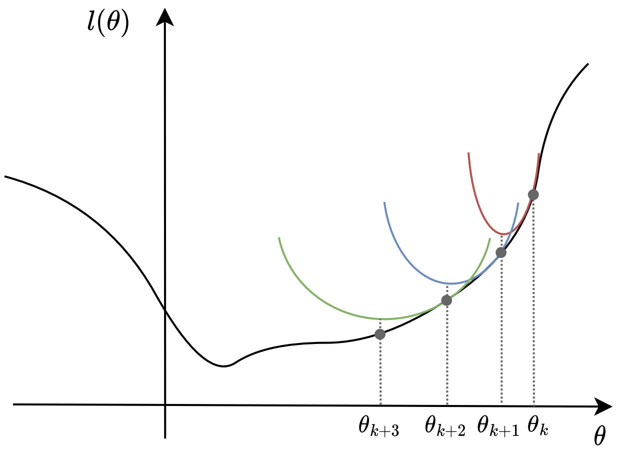


Figure 2.3: Newton's method for a scalar function with $\alpha = 1$.

2.2.3. Adaptive Activation Functions

The neural activation functions play a fundamental role in neural networks. Note in this regard that if the activation functions are linear then the neural network becomes a linear regression model. Therefore, the role of the activation function is to introduce nonlinearities so that the network is able to replicate complex tasks. In addition, it should be noted that the activation function is directly involved in the evaluation of the derivatives of the loss function with respect to the parameters in the minimization process. Therefore, the training outcome of neural networks is highly dependent on the selection of the activation function. It is not an easy task to select the activation function because it is closely related to the problem at hand, and a trial and error procedure is usually carried out. This procedure requires computational resources, so it is generally not possible to find the best option for a specific problem. To enhance the representation capabilities of the activation functions a trainable parameter is introduced in the mathematical formulation of the function [19, 20]. Let us consider a generic activation function, $\sigma := \mathbb{R} \to \mathbb{R}$, then its adaptive version, $\sigma_{adaptive} := \mathbb{R} \to \mathbb{R}$, can be expressed as follows:

$$\sigma_{adaptive}(x) = \sigma(\rho\,\kappa\,x) \tag{2.26}$$

where the parameter $\kappa \in \mathbb{R}$ is referred as slope and is adjusted during training, exactly like the other parameters, while $\rho \in [1, +\infty)$ is a fixed scale factor which is used to accelerates the convergence towards the global minima. Note that the slope parameter and the scale factor may be different for each neuron in the network. Following the work of the authors, the slope is initialized such that $\kappa\rho = 1$ regardless of the value of the scale factor.

2.2.4. Loss Landscape

The minimization of the loss function is a difficult task. The outcome of the training process depends strongly on the architecture of the neural network and on the selection of the parameters of the optimizer. The process of design choices can be imagined as locks picking, it is a very delicate process and the exact combination of parameters must be found. The effects of the various choices can be understood with loss function display techniques. Nevertheless loss function are high-dimensional so dimensionality reduction techniques must be applied in order to visualize these functions [26]. The easiest technique consists of selecting two vectors of parameters, $\theta_1, \theta_2 \in \mathbb{R}^p$ and plot the values of the loss function along the line which connect these $p-$dimensional vectors. This line can be parameterized to define the following auxiliary function $f \coloneqq [0,1] \rightarrow \mathbb{R}$:

$$f(\alpha) = l((1-\alpha)\theta_1 + \alpha\theta_2) \tag{2.27}$$

where $l(\cdot)$ denotes the loss function. The plot of the auxiliary function is a representation of the so called loss landscape. This technique can be extended to the $2-$dimensional case as follows. The first step is to select a vector of parameters $\hat{\theta} \in \mathbb{R}^p$ and two random vectors $\delta, \eta \in \mathbb{R}^p$. Then, the following auxiliary function, $f \coloneqq \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{R}$, can be defined:

$$f(\alpha, \beta) = l(\hat{\theta} + \alpha\delta + \beta\eta) \tag{2.28}$$

where the interval $\mathcal{I} \subset \mathbb{R}$ can be selected, for example, as $\mathcal{I} = [-1, 1]$. If the parameters vector is selected as the result of the optimization process it is possible to have a representation of the loss landscape around the optimizer convergence point. It must be noticed that to have a meaningful representation the following filter normalization technique, must be applied [26]. Let us denote the collection of the layers parameters as $\hat{\theta} = [\hat{\theta}^{[i]}, ..., \hat{\theta}^{[l-1]}]$ where l denotes the total number of layer in the neural network. The same partition can be applied to the random vectors, i.e. $\delta = [\delta^{[i]}, ..., \delta^{[l-1]}]$, $\eta = [\eta^{[i]}, ..., \eta^{[l-1]}]$. The following replacements must be made in the evaluation of the function defined in Equation (2.28):

$$\delta^{[i]} \leftarrow \delta^{[i]} \frac{||\hat{\theta}^{[i]}||_2}{||\delta^{[i]}||_2} \qquad i = 1, ..., l-1 \tag{2.29}$$

$$\eta^{[i]} \leftarrow \eta^{[i]} \frac{||\hat{\theta}^{[i]}||_2}{||\eta^{[i]}||_2} \qquad i = 1, ..., l-1 \tag{2.30}$$

2.3. Physics-Informed Neural Networks

The prediction ability of neural networks depends heavily on the data exploited during the training process. In the naive setting of supervised learning, it is not possible to know a priori how the trained networks behave on all possible meaningful inputs not considered during training. This unpredictable nature could be catastrophic in physical problems solving. Thus, the main limitation of neural networks is the reliability. Let us consider, for example, the following one-dimensional thermodynamic problem. A plane wall is subjected to an internal heat source. The wall is thermally insulated on one side and it exchanges heat trough convection on the other side. It is well know that the steady state distribution of the temperature inside the wall is the solution of the heat equation:

$$\begin{cases} D^2 T(x) + \dfrac{q}{\lambda} = 0 & x \in [0, L] & \text{(2.31a)} \\ D\,T(x) = 0 & x = 0 & \text{(2.31b)} \\ T(x) = T_\infty + \dfrac{qL}{h} & x = L & \text{(2.31c)} \end{cases}$$

where λ and L denote the thermal conductivity and the thickness of the wall, respectively, q is the heat generated per unit volume per unit second, T_∞ is the temperature of the air surrounding the wall and h is the convective heat transfer coefficient. Note that Equations (2.31b) and (2.31c) express the boundary conditions of the problem. The temperature distribution can be approximated with a neural network, i.e. $T(\cdot) \approx \mathcal{N}(\cdot \,|\, \theta)$. To train the network the temperature can be measured at a finite number of points and a labeled dataset consisting of pairs of positions and corresponding temperatures can be created, i.e $\{(x_{data}^{(i)}, T_{data}^{(i)})\}_{i=1}^{n_{data}}$. The loss function is then formulated as follows:

$$l_{data}(\theta) = \frac{1}{n_{data}} \sum_{i=1}^{n_{data}} [T_{data}^{(i)} - \mathcal{N}(x_{data}^{(i)} \,|\, \theta)]^2 \tag{2.32}$$

The trained network should be able to predict the correct temperature at any point on the wall, but there is no guarantee that the network will not predict a negative temperature, i.e below the absolute zero, at a given point. This is obviously not possible from a physical point of view, but no one has taught the network that this is not acceptable. The basic paradigm of supervised learning may thus fail, but it should be noted that the knowledge of the physical laws governing heat transfers is not used at all. To this aim, let us expand the loss function as follows:

$$l(\theta) = l_{ode}(\theta) + l_{bc1}(\theta) + l_{bc2}(\theta) + l_{data}(\theta) \tag{2.33}$$

where the individual terms can be expressed as:

$$l_{ode}(\theta) = \frac{1}{n_{ode}} \sum_{i=1}^{n_{ode}} \left[D^2 \mathcal{N}(x_{ode}^{(i)} \,|\, \theta) + \frac{q}{\lambda} \right] \tag{2.34a}$$

$$l_{bc1}(\theta) = [D\, \mathcal{N}(0 \,|\, \theta)]^2 \tag{2.34b}$$

$$l_{bc2}(\theta) = \left[\mathcal{N}(L \,|\, \theta) - T_\infty + \frac{qL}{h} \right]^2 \tag{2.34c}$$

It can now be seen that the structure of the heat equation is imposed on a finite number of collocation points $x_{ode}^{(i)}$ for $i = 1, ..., n_{ode}$ by the loss term in Equation (2.34a). In addition, the reaming loss terms ensure that the boundary conditions of the problem are satisfied. It should be pointed out that the derivatives of the neural network function with respect to the inputs can be efficiently evaluated by exploiting *automatic differentiation*, namely the methodology already used by minimization algorithms to evaluate the derivatives of the loss function with respect to the parameters of the network. The physical constraints imposed on the training process amplify the content information of the observed data. Therefore, the neural networks become able to generalize even if few examples are available for the training process. In addition, note that the the loss term related to the data, Equation (2.32), is not strictly necessary, therefore, if process measurements are not available, it is still possible to solve the problem. The procedure presented can be even extended to solve system identification problems. Suppose that the convective heat transfer coefficient is unknown in Equation (2.31a). This coefficient can be included in the list of trainable parameters of the network and the loss function can be defined as $l := \mathbb{R}^p \times \mathbb{R} \to \mathbb{R} \,|\, (\theta, h) \mapsto l(\theta, h)$. Thus, the heat transfer coefficient is estimated as a by-product of the minimization process. The variants of neural networks presented are called *Physics-Informed Neural Networks* [37]. Physics-informed neural networks are thus function approximators that exploit the knowledge of any physical laws, which govern a given problem, in the training of the network. To a more general extend, physics-informed neural networks can be exploited to approximate the solutions of partial differential equations. Let us consider for example a generic partial differential problem on the unknown function $u := \mathbb{R}^n \to \mathbb{R}$:

$$\begin{cases} \mathcal{F}[u(x)] = h(x) \quad x \in \Omega & (2.35a) \\ \mathcal{B}[u(x)] = g(x) \quad x \in \partial\Omega & (2.35b) \end{cases}$$

where $\Omega \subset \mathbb{R}^n$, $\mathcal{F}[\cdot]$ is the nonlinear differential operator, $\mathcal{B}[\cdot]$ is the operator defining the boundary condition and $h := \mathbb{R}^n \to \mathbb{R}$ and $g := \mathbb{R}^n \to \mathbb{R}$ are problem-specific functions.

The unknown solution of the problem can be approximated by a neural network:

$$u(\cdot) \approx \mathcal{N}(\cdot \,|\, \theta) \tag{2.36}$$

The network can then be trained to minimize the following loss function:

$$l(\theta) = \omega_{\mathcal{F}}\, l_{\mathcal{F}}(\theta) + \omega_{\mathcal{B}}\, l_{\mathcal{B}}(\theta) + l_{data}(\theta) \tag{2.37}$$

where $\omega_{\mathcal{F}}, \omega_{\mathcal{B}} \in \mathbb{R}$ are the weights introduced to balance the different loss terms:

$$l_{\mathcal{F}}(\theta) = \frac{1}{n_{\mathcal{F}}} \sum_{j=1}^{n_{\mathcal{F}}} \mathcal{F}[\mathcal{N}(x_{\mathcal{F}}^{[i]} \,|\, \theta)]^2 \tag{2.38a}$$

$$l_{\mathcal{B}}(\theta) = \frac{1}{n_{\mathcal{B}}} \sum_{j=1}^{n_{\mathcal{B}}} \mathcal{B}[\mathcal{N}(x_{\mathcal{B}}^{(j)} \,|\, \theta)]^2 \tag{2.38b}$$

$$l_{data}(\theta) = \frac{1}{n_{data}} \sum_{j=1}^{n_{data}} [\mathcal{N}(x_{data}^{(j)} \,|\, \theta) - u(x_{data}^{(j)})]^2 \tag{2.38c}$$

where $\{(x_{\mathcal{F}}^{(j)})\}_{j=1}^{n_{\mathcal{F}}}$ are the datapoints where the structure of the partial differential equation is imposed and similarly $\{(x_{\mathcal{B}}^{(j)})\}_{j=1}^{n_{\mathcal{B}}}$ are the datapoints where the boundary condition is imposed, finally, $\{(x_{data}^{(j)})\}_{j=1}^{n_{data}}$ are the datapoints where the unknown function of the problem is known, for example because of a measurement process.

3 | Optimal Control Problems

Tanquam ex ungue leonem
[I recognise the lion by its claw]. [1]

Johann Bernoulli

3.1. Mathematical Formulation

Let $x(t) \in \mathbb{R}^n$ be the state of a dynamical system at a generic time instant and similarly, let $c(t) \in C \subset \mathbb{R}^m$ be the external control applied to the system, where the subset C represents the vector space of admissible controls. Suppose that the dynamical evolution of the system is described by the function $f : \mathbb{R} \times \mathbb{R}^n \times C \to \mathbb{R}^n$, then a generic optimal control problem can be formally stated as follows [29]:

$$\begin{aligned}
&\min_{c(\cdot)\in\mathcal{C}} && J(c(\cdot)) = \phi(t_f, x(t_f)) + \int_{t_i}^{t_f} L(t, x(t), c(t))dt && \text{(3.1a)}\\
&\text{subject to} && \dot{x}(t) = f(t, x(t), c(t)), && \text{(3.1b)}\\
& && x(t_i) = x_i, && \text{(3.1c)}\\
& && \psi(t_f, x(t_f)) = 0. && \text{(3.1d)}
\end{aligned}$$

where $t_i, t_f \in \mathbb{R}$ denote the initial time and the final time of the problem, respectively. Note that the final time could be unknown, for example in time-optimal problems. The initial state of the system is $x_i \in \mathbb{R}^n$ while the function $\psi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^l$ with $l \leq n$ describes the terminal constraints imposed on the state of the system. The objective is to find the control function $c := [t_i, t_f] \to C$, i.e. the optimal control, which minimizes the cost functional of the problem $J : \mathcal{C} \to \mathbb{R}$, where clearly $\mathcal{C} := \{c : [t_i, t_f] \to C\}$. In particular, note that the cost functional is composed of the following terms: the terminal cost function $\phi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ and the running cost function $L : \mathbb{R} \times \mathbb{R}^n \times C \to \mathbb{R}$.

[1] Johann Bernoulli made this remark after receiving an anonymous solution to the *Brachistochrone Problem* and recognizing that the solution was the work of Isaac Newton.

3.2. Pontryagin's Maximum Principle

Let us introduce the Hamiltonian of the system $H : \mathbb{R} \times \mathbb{R}^n \times C \times \mathbb{R}^n \to \mathbb{R}$:

$$H(p, q, r, s) := L(p, q, r) + \langle s, f(p, q, r)\rangle \tag{3.2}$$

The following necessary conditions, referred as Euler-Lagrange equations, must be met for a minimum of the cost functional [5]:

$$\begin{cases} \dot{x}(t) = \partial_s H(t, x(t), c(t), \lambda(t)) & (3.3a) \\ \dot{\lambda}(t) = -\partial_q H(t, x(t), c(t), \lambda(t)) & (3.3b) \\ 0 = \partial_r H(t, x(t), c(t), \lambda(t)) & (3.3c) \end{cases}$$

where $\lambda(t) \in \mathbb{R}^n$ is the costate of the dynamical system. Pontryagin's Maximum Principle states that Equation (3.3c) can be replaced with the following expression to evaluate the optimal control function:

$$c_*(t) = \arg\min_{\bar{c} \in C} H(t, x(t), \bar{c}, \lambda(t)) \tag{3.4}$$

The system of Equations (3.3) can then be simplified as follows:

$$\begin{cases} \dot{x}(t) = \partial_s H(t, x(t), c_*(t), \lambda(t)) \\ \dot{\lambda}(t) = -\partial_q H(t, x(t), c_*(t), \lambda(t)) \end{cases} \tag{3.5}$$

where it is implied that the control is evaluated with Equation (3.4). To solve this system of $2n$ ordinary differential equations, a total number of $2n$ boundary conditions are needed [30]. In the case where the final time is unknown, the conditions become $2n + 1$ and this is the case considered in the remainder of the section. The initial conditions of the problem, Equation (3.1c), provide n boundary conditions and the terminal constraints, Equation (3.1d), provide l equations on the final state of the system, so there are still $n + 1 - l$ missing boundary conditions. Let us explicit the independent variables in the terminal constraints function $\psi : (v, w) \mapsto \psi(v, w)$ and in the terminal cost function $\phi : (v, w) \mapsto \phi(v, w)$. The state of the system at the final time must respect the terminal constraints and thus the following equations must hold:

$$d\psi_i(t_f, x(t_f)) = \partial_v \psi_i(t_f, x(t_f))dv + \langle \partial_w \psi_i(t_f, x(t_f)), dw\rangle = 0, \qquad i = 1, .., l \tag{3.6}$$

The previous equations can be solved to obtain l differentials as a function of the remaining $n+1-l$ differentials. The l differentials obtained can be substituted in the following transversality condition:

$$[H(t_f, x(t_f), c(t_f), \lambda(t_f)) - \partial_v \, \phi(t_f, x(t_f))] \, dv - \langle \lambda(t_f) + \partial_w \, \phi(t_f, x(t_f)) \rangle, dw\rangle = 0 \quad (3.7)$$

to get an expression which depends only on $n+1-l$ independent differentials. To satisfy the expression obtained, the coefficients multiplying the remaining differentials must be set to zero, thus obtaining a system of $n+1-l$ equations:

$$N_i(t_f, x(t_f), \lambda(t_f)) = 0, \quad i = 1, ..., n+1-l \quad (3.8)$$

The problem consisting of the system of differential equations, Equations (3.5), and the boundary conditions, Equation (3.1d) and Equations (3.8), can be reformulated as a two-point boundary value problem and solved with shooting methodologies [30]. In this regard, note that the only unknowns of the problem are the initial costate, $\lambda_i \in \mathbb{R}^n$, such that $\lambda(t_i) = \lambda_i$, the costate normalization factor and the final time. If these variables are know, the dynamic evolution of the system can be obtained by integrating Equations (3.5). Let us introduce the shooting function $\gamma : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n+1}$:

$$\gamma(\lambda_i, t_f) = \begin{bmatrix} \psi(t_f, x(t_f)) \\ N_j(t_f, x(t_f), \lambda(t_f)) \\ \vdots \\ N_{n+1-l}(t_f, x(t_f), \lambda(t_f)) \end{bmatrix} \quad (3.9)$$

The shooting problem can be then formalized as follows:

$$\begin{aligned} &\text{find } (\lambda_i, t_f) && \text{such that } \gamma(\lambda_i, t_f) = 0 && (3.10a)\\ &\text{subject to} && \dot{x}(t) = \partial_s H(t, x(t), c_*(t), \lambda(t)), && (3.10b)\\ &&& \dot{\lambda}(t) = -\partial_q \, H(t, x(t), c_*(t), \lambda(t)), && (3.10c)\\ &&& x(t_i) = x_i, && (3.10d)\\ &&& \lambda(t_i) = \lambda_i. && (3.10e) \end{aligned}$$

To solve the problem, it is sufficient to find the zeros of the shooting function, and thus root-finding algorithms can be exploited. Note that the algorithms require an initial guess which is difficult to provide because the costate has no physical meaning.

3.3. Hamilton-Jacobi-Bellman Principle

The cost functional in Equation (3.1a) can be interpreted in a more general sense as a function of the initial time and the initial state as well i.e. $J : \mathbb{R} \times \mathbb{R}^n \times \mathcal{C} \to \mathbb{R}$:

$$J(t_i, x_i, c(\cdot)) = \phi(t_f, x(t_f)) + \int_{t_i}^{t_f} L(t, x(t), c(t))dt \tag{3.11}$$

It is possible to introduce the value function, $\nu : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$, representing the minimum value of the cost functional for an arbitrary initial time and initial state [6]:

$$\nu(T, X) \coloneqq \min_{c(\cdot)\in\mathcal{C}} J(T, X, c(\cdot)) \tag{3.12}$$

The Hamilton-Jacobi-Bellman principle states that the value function should satisfy the following partial differential equation with the associated boundary condition:

$$\begin{cases} \partial_T \, \nu(T, X) = -\min_{\bar{c}\in C} H\left(T, X, \bar{c}, \partial_X \, \nu(T, X)\right) & \forall\, X \in \mathbb{R}^n, T \in [t_i, t_f] \qquad (3.13a) \\ \nu(t_f, X) = \phi(t_f, X) & \forall\, X \in \mathbb{R}^n : \psi(t_f, X) = 0 \qquad (3.13b) \end{cases}$$

It must be noticed that if the value function is known then the optimal control can be evaluated along an optimal trajectory as follows:

$$c_*(t) = \arg\min_{\bar{c}\in C} H\left(t, x(t), \bar{c}, \partial_X \, \nu(t, x(t))\right) \tag{3.14}$$

Therefore, considering Equation 3.4, the following must hold along an optimal trajectory:

$$\lambda(t) = \partial_X \, \nu(t, x(t)) \tag{3.15}$$

Thus, a link is established between the costate introduced in the Euler-Lagrange equations and the value function defined in the Hamilton-Jacobi-Bellman principle.

4 | Problem Statement

We can't solve problems by using the same kind of thinking we used when we created them.

Albert Einstein

4.1. Problem Definition

The problem considered is a fuel-optimal low-thrust transfer between two planets in the Solar System. The orbits of the planets are assumed to be circular and coplanar. The spacecraft is modeled as a point mass and is subject to the gravitational attraction of the Sun and to the thrust force determined by its propulsion system. Let us introduce the state of the spacecraft at a given time instant $x(t) \in \mathbb{R}^5$. The state of the spacecraft can be expressed in polar coordinates as follows:

$$x(t) = [r(t), \phi(t), v_r(t), v_\phi(t), m(t)]^T \tag{4.1}$$

where $r(t) \in \mathbb{R}_{>0}$ is the radial position, $\phi(t) \in \mathbb{R}$ is the angular position, $v_r(t), v_\phi(t) \in \mathbb{R}$ are respectively the radial and tangential components of the velocity and $m(t) \in \mathbb{R}_{>0}$ is the mass. Let $u(t) \in [0, 1]$ be the thrust throttle factor and $\alpha(t) \in A \subset \mathbb{R}^2$ be the unit vector representing the thrust direction, where clearly $A := \{v \in \mathbb{R}^2 \,|\, \|v\|_2 = 1\}$ and the following notation is used $\alpha(t) = [\alpha_r(t), \alpha_\phi(t)]^T$. Note that it is also possible to indicate the direction of the thrust through the thrust angle, $\beta(t) \in (-\pi/2, \pi/2)$, defined as:

$$\beta(t) = \arctan\left(\frac{\alpha_r(t)}{\alpha_\phi(t)}\right) \tag{4.2}$$

A schematic representation of the problem is reported in Figure 4.1.

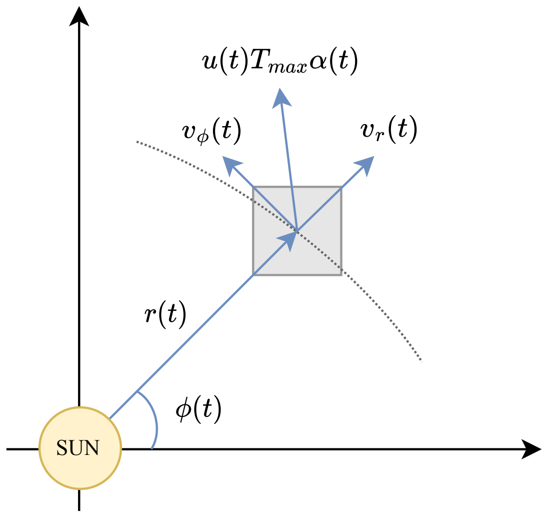


Figure 4.1: Model of the low-thrust transfer problem.

Let us introduce the following dimensionless coefficients:

$$a_1 = \frac{T_{max}\,T_C^2}{m_i\,L_C}, \quad a_2 = \frac{T_{max}\,T_C}{I_{sp}\,g_0\,m_i}, \quad a_3 = \mu\frac{T_C^2}{L_C^3} \tag{4.3}$$

where μ is the Sun gravitational parameter, L_C and T_C are the characteristic length and time of the problem, T_{max}, I_{sp} and m_i are the maximum thrust, the specific impulse and the initial mass of the spacecraft and g_0 is the standard gravitational acceleration. The motion of the spacecraft, in dimensionless form, can be described by the following system of ordinary differential equations:

$$\begin{cases} \dot{r}(t) = v_r(t) \\ \dot{\phi}(t) = \dfrac{v_\phi(t)}{r(t)} \\ \dot{v}_r(t) = \dfrac{v_\phi^2(t)}{r(t)} - a_3\dfrac{1}{r^2(t)} + a_1\dfrac{u(t)\,\alpha_r(t)}{m(t)} \\ \dot{v}_\phi(t) = -\dfrac{v_r(t)\,v_\phi(t)}{r(t)} + a_1\dfrac{u(t)\,\alpha_\phi(t)}{m(t)} \\ \dot{m}(t) = -a_2\,u(t) \end{cases} \tag{4.4}$$

The control will also be referred as $c(t) \in C \subset \mathbb{R}^3$, where $c(t) = [u(t), \alpha_r(t), \alpha_\phi(t)]^T$ and it must be $C \coloneqq \{(u, v, w) \in \mathbb{R}^3 \,|\, u \in [0, 1] \wedge (v, w) \in A\}$. Therefore, the dynamics of the spacecraft can be expressed compactly as $\dot{x}(t) = f(x(t), c(t))$, with $f \coloneqq \mathbb{R}^5 \times C \to \mathbb{R}^5$.

The initial conditions of the spacecraft can be stated as:

$$x(t_i) = \left[r_i,\, \phi_i,\, 0,\, \sqrt{\frac{a_3}{r_i}},\, m_i \right]^T \tag{4.5}$$

where $t_i \in \mathbb{R}$ denotes the initial time while $r_i \in \mathbb{R}_{>0}$ and $\phi_i \in \mathbb{R}$ are respectively the orbital radius and the angular position of the departure planet. The final state of the spacecraft must respect the condition $\psi(t_f) = 0$, with $\psi := \mathbb{R}^5 \to \mathbb{R}^4$, defined as:

$$\psi(x(t_f)) := \begin{bmatrix} r(t_f) - r_f \\ \phi(t_f) - \phi_f \\ v_r(t_f) \\ v_\phi(t_f) - \sqrt{\frac{a_3}{r_f}} \end{bmatrix} = 0 \tag{4.6}$$

where $t_f \in \mathbb{R}$ denotes the final time while $r_f \in \mathbb{R}_{>0}$ and $\phi_f \in \mathbb{R}$ are respectively the orbital radius and the angular position of the arrival planet. In summary, the final radial and angular positions and final radial and tangential velocities of the spacecraft are fixed, while the final mass is free. In addition, note that the final time is left free and must be obtained by solving the optimal control problem, so the problem can be classified as a free-time partially fixed endpoint control problem.

4.2. Optimal Control Problem Formulation

The fuel-optimal control problem can be formally stated as follows:

$$\begin{aligned} \min_{c(\cdot) \in \mathcal{C}} \quad & J(c(\cdot)) = a_2 \int_{t_i}^{t_f} L(c(t))\, dt && \text{(4.7a)} \\ \text{subject to} \quad & \dot{x}(t) = f(x(t), c(t)), && \text{(4.7b)} \\ & x(t_i) = x_i && \text{(4.7c)} \\ & \psi(x(t_f)) = 0. && \text{(4.7d)} \end{aligned}$$

where the running cost $L := C \to \mathbb{R}$ takes the form $L(c(t)) := u(t)$ and clearly, based on the previous consideration, $\mathcal{C} := \{c := [t_i, t_f] \to C\}$. It can be noticed that the terminal cost is absent in this formulation. To conclude, note that the following applies:

$$J(c(\cdot)) = a_2 \int_{t_i}^{t_f} u(t)\, dt = -\int_{t_i}^{t_f} \dot{m}(t)\, dt = m(t_i) - m(t_f) \tag{4.8}$$

The cost functional then quantifies the mass transformed in energy to reach the final target, i.e. the propellant mass required for the transfer.

4.2.1. Homotopy Technique

The fuel-optimal problem introduced in the previous section is notoriously difficult to solve with shooting methodologies. The main reason is that the optimal time evolution of the thrust throttle factor presents a discontinuous bang-bang structure. To smooth the problem, the running cost is slightly modified as follows:

$$L(c(t)) := u(t) - \epsilon \log\left[u(t)(1 - u(t))\right] \tag{4.9}$$

where $\epsilon \in \mathbb{R}_{\geq 0}$ is the homotopy coefficient. Note that the homotopy coefficient defines a class of problems and in particular, if $\epsilon = 0$, the original formulation of the cost functional is restored. The homotopy technique can be summarized as follows. The problem defined by $\epsilon = 1$ is solved first because the root-finding algorithms converge more frequently with random initial guesses. The homotopy coefficient is then slightly decreased to define a second optimal problem, and the solution obtained in the previous step is provided to the solver as initial guess. Finally, the same procedure is repeated iteratively until the homotopy coefficient is zero, resulting in the fuel-optimal solution.

4.2.2. Costate Normalization Technique

It must be noticed that, despite the introduction of the homotopy technique it is still complex to provide a good initial guess of the initial costate to the root-finding algorithms. To overcome this difficulty the initial costate is constrained on a unit high-dimensional hypersphere thus reducing the range of possibilities. To this aim, the cost function is modified as follows:

$$J(c(\cdot)) = \lambda_0 a_2 \int_{t_i}^{t_f} u(t) - \epsilon \log\left[u(t)(1 - u(t))\right] dt \tag{4.10}$$

where $\lambda_0 \in \mathbb{R}_{>0}$ is the costate normalization factor. Note that this parameter does not change the nature of the problem. Let us introduce the costate of the spacecraft, $\lambda(t) \in \mathbb{R}^5$, which is denoted:

$$\lambda(t) = [\lambda_r(t), \lambda_\phi(t), \lambda_{v_r}(t), \lambda_{v_\phi}(t), \lambda_m(t)]^T \tag{4.11}$$

The normalization factor is then selected such that the following condition is satisfied:

$$\sqrt{\lambda_r^2(t_i) + \lambda_\phi^2(t_i) + \lambda_{v_r}^2(t_i) + \lambda_{v_\phi}^2(t_i) + \lambda_m^2(t_i) + \lambda_0^2} = 1 \tag{4.12}$$

Note that this additional constraint is added to the definition of the shooting function.

4.2.3. Consequences of Pontryagin's Maximum Principle

The Hamiltonian of the problem, $H := \mathbb{R}^5 \times C \times \mathbb{R}^5 \to \mathbb{R}$, can be expressed as follows:

$$H(p, q, r) = \lambda_0 a_2 L(q) + \langle r, f(p, q) \rangle \tag{4.13}$$

The necessary conditions, i.e. the Euler-Lagrange equations, to minimize the cost functional take the subsequent form:

$$\begin{cases} \dot{x}(t) = \partial_r H(x(t), c(t), \lambda(t)) = f(x(t), c(t)) & \text{(4.14a)} \\ \dot{\lambda}(t) = -\partial_p H(x(t), c(t), \lambda(t)) & \text{(4.14b)} \\ 0 = \partial_q H(x(t), c(t), \lambda(t)) & \text{(4.14c)} \end{cases}$$

In particular, the compact system of Equations (4.14b) can be expanded as:

$$\begin{cases} \dot{\lambda}_r(t) = -2a_3 \dfrac{\lambda_{v_r}(t)}{r^3(t)} + \dfrac{\lambda_\phi(t) v_\phi(t)}{r^2(t)} + \dfrac{\lambda_{v_r}(t) v_\phi^2(t)}{r^2(t)} - \dfrac{\lambda_{v_\phi}(t) v_r(t) v_\phi(t)}{r^2(t)} \\ \dot{\lambda}_\phi(t) = 0 \\ \dot{\lambda}_{v_r}(t) = -\lambda_r(t) + \dfrac{\lambda_{v_\phi}(t) v_\phi(t)}{r(t)} \\ \dot{\lambda}_{v_\phi}(t) = -\dfrac{\lambda_\phi(t)}{r(t)} - 2\dfrac{\lambda_{v_r}(t) v_\phi(t)}{r(t)} + \dfrac{\lambda_{v_\phi}(t) v_r(t)}{r(t)} \\ \dot{\lambda}_m(t) = a_1 \dfrac{\lambda_{v_r}(t) u(t) \alpha_r(t)}{m^2(t)} + a_1 \dfrac{\lambda_{v_\phi}(t) u(t) \alpha_\phi(t)}{m^2(t)} \end{cases} \tag{4.15}$$

The optimal control is evaluated by applying Pontryagin's Maximum Principle instead of considering Equation (4.14c). In this regard, let us introduce the following auxiliary function $Q := \mathbb{R}^5 \times \mathbb{R}^5 \to \mathbb{R}^3$:

$$Q(p, r) = \arg\min_{\bar{c} \in C} H(p, \bar{c}, r) \tag{4.16}$$

The optimal control is then obtained by evaluating the previous expression on the state and the costate of the spacecraft:

$$c_*(t) = Q(x(t), \lambda(t)) \tag{4.17}$$

In detail, for the problem considered the auxiliary function is defined as follows [4]:

$$Q(x(t), \lambda(t)) = \begin{bmatrix} \dfrac{2\epsilon}{2\epsilon + S(x(t), \lambda(t)) + \sqrt{4\epsilon^2 + S(x(t), \lambda(t))^2}} \\ -\dfrac{\lambda_{v_r}(t)}{\|\lambda_v(t)\|} \\ -\dfrac{\lambda_{v_\phi}(t)}{\|\lambda_v(t)\|} \end{bmatrix} \tag{4.18}$$

where the notation $\lambda_v(t) = [\lambda_{v_r}(t), \lambda_{v_\phi}(t)]^T$ is used. Note that the function introduced in the previous expression, $S := \mathbb{R}^5 \times \mathbb{R}^5 \to \mathbb{R}$, is called switching function and if evaluated on the state and costate of the spacecraft takes the form:

$$S(x(t), \lambda(t)) = 1 - \frac{\lambda_m(t)}{\lambda_0} - \frac{a_1}{a_2} \frac{\|\lambda_v(t)\|}{\lambda_0 m(t)} \tag{4.19}$$

To conclude, it is worth noting a property of the Hamiltonian valid for the problem presented. Let us introduce the auxiliary function $g := [t_i, t_f] \to \mathbb{R}$:

$$g(t) = H(x(t), c(t), \lambda(t)) \tag{4.20}$$

Note that the following simplified notation is used:

$$\partial_\star H(x(t), c(t), \lambda(t)) = \partial_\star H(\sim), \qquad \star = (p, q, r) \tag{4.21}$$

The system of Equations (4.14) implies that:

$$\begin{aligned} \dot{g}(t) &= \langle \partial_p H(\sim), \dot{x}(t) \rangle + \partial_q H(\sim) + \langle \partial_r H(\sim), \dot{\lambda}(t) \rangle \\ &= -\cancel{\langle \dot{\lambda}(t), \dot{x}(t) \rangle} + \cancelto{0}{\partial_q H(\sim)} + \cancel{\langle \dot{x}(t), \dot{\lambda}(t) \rangle} \\ &= 0 \end{aligned} \tag{4.22}$$

The conclusion is that the Hamiltonian, evaluated along an optimal trajectory, is constant.

4.2.4. Shooting Problem Formulation

To derive the boundary conditions of the problem, let us consider the constraint function, $\psi := \mathbb{R}^5 \to \mathbb{R}^4 \,|\, w \mapsto \psi(w)$. The following necessary conditions must be met:

$$d\psi_i = dw_i = 0 \qquad i = 1, .., 4 \tag{4.23}$$

Note that, the transversality condition of the problem can be expressed as:

$$H(x(t_f), c(t_f), \lambda(t_f))dv - \langle \lambda(t_f), dw \rangle = 0 \tag{4.24}$$

The differentials obtained in Equation (4.23) can be then substituted in the transversality condition to obtain the following relation:

$$H(x(t_f), c(t_f), \lambda(t_f))dv - \lambda_m(t_f)dw_5 = 0 \tag{4.25}$$

To satisfy this relation, the coefficients multiplying the differentials must be set to zero, so the following boundary conditions are finally obtained:

$$\begin{cases} H(x(t_f), c(t_f), \lambda(t_f)) = 0 & \text{(4.26a)} \\ \lambda_m(t_f) = 0 & \text{(4.26b)} \end{cases}$$

To express the conditions in words, the Hamiltonian and the mass costate evaluated at the final time must be zero. Note that, considering that the Hamiltonian must be constant along an optimal trajectory (cf. Equation (4.22)), it can be concluded that the Hamiltonian it is always zero. The shooting function, $\gamma := \mathbb{R}^5 \times \mathbb{R} \times \mathbb{R}_{>0} \to \mathbb{R}$, is then defined by considering the boundary conditions just obtained and the relation introduced to determine the costate normalization factor, Equation (4.12), as follows:

$$\gamma(\lambda_i, t_f, \lambda_0) = \begin{bmatrix} \psi(x(t_f)) \\ H(x(t_f), c(t_f), \lambda(t_f)) \\ \lambda_m(t_f) \\ \sqrt{\langle \lambda(t_i), \lambda(t_i) \rangle + \lambda_0^2} - 1 \end{bmatrix} \tag{4.27}$$

4.2.5. Consequences of Hamilton-Jacobi-Bellman Principle

To complete the section, let us introduce the value function $\nu : \mathbb{R}^5 \to \mathbb{R}$, which is a function of the initial state of the spacecraft but not of the initial time because of the properties of the problem studied. It should be remembered that the value function is the minimum value of the cost functional of the optimal control problem, and in the context of spacecraft guidance it represents the optimal transfer cost. Specifically, in the fuel-optimal problem, the transfer cost corresponds to the mass of propellant required to reach the target planet. The Hamilton-Jacobi-Bellman equation can be simplified as follows for the problem stated:

$$\begin{cases} \min\limits_{\bar{c} \in C} H(X, \bar{c}, \partial_X \nu(X)) = 0 & X \in \mathbb{R}^5 \quad (4.28a) \\ \nu(X) = 0 & \forall X \in \mathbb{R}^5 : \psi(X) = 0 \quad (4.28b) \end{cases}$$

In particular, from the previous discussion, the minimizer of Equation (4.28a) can be evaluated by considering Equation (4.18), as follows:

$$\bar{c} = Q(X, \partial_X \nu(X)) \tag{4.29}$$

It must be emphasized that the knowledge of the value function is sufficient to evaluate the optimal control given the state of the spacecraft:

$$c_*(t) = Q(x(t), \partial_X \nu(x(t))) \tag{4.30}$$

Therefore, by exploiting this relation, it is clear that the spacecraft can be controlled with a feedback control law. To conclude, by comparing Equation (4.30) and (4.17) it is evident that along an optimal trajectory:

$$\lambda(t) = \partial_X \nu(x(t)) \tag{4.31}$$

In particular, this relation will be exploited in the construction of physics-informed neural networks together with the Hamilton-Jacobi-Bellman equation.

5 | Neurocontroller

The first ultraintelligent machine is the last invention that man need ever make, provided that the machine is docile enough to tell us how to keep it under control.

Nick Bostrom

The objective of the present work is to design a neurocontroller, i.e. a neural network-based controller, capable of steering the spacecraft from different initial states to the final target. The neurocontroller should be able to output the optimal controls given the current state of the spacecraft, such that the spacecraft can be autonomously guided in closed-loop. This general problem can be split in two minor problems. The first consists in constructing a database of optimal trajectories which can be exploited to train neural networks. The second problem encapsulates every aspect related to the creation and the training process of neural networks.

5.1. Database of Optimal Trajectories

The straightforward approach to create a database of optimal trajectories would be to perturb the initial state of the nominal trajectory and to solve the corresponding optimal control problem characterized by the new set of initial conditions, thus generating a new optimal trajectory. This approach scales very poorly because of the computational difficulties associated with two-point boundary values problems with shooting methodologies. Therefore, in order to generate such a database, the backward generation method, introduced by Izzo et Öztürk [17], is exploited. It should be noted that the method was originally presented for a transfer between two orbits and in this work it is extended to the rendezvous problem discussed in the previous chapter.

5.1.1. Backward Generation Method

The database is populated starting from the nominal trajectory without the need of solving additional shooting problems. To this aim, the first step consists in perturbing the components of the final state and costate of the spacecraft of the nominal trajectory. The perturbations cannot be arbitrary because the following conditions must be respected:

- The final radial and angular positions and the final radial and tangential velocities of the spacecraft must coincide with the arrival planet ones.
- The mass costate must be zero due to the transversality condition.

Therefore, these components cannot be perturbed and are kept constant while the remaining free components are randomly perturbed as follows:

$$\beta_i^{new}(t_f) = \beta_i^{old}(t_f) + \delta_i \quad i = 1, ..., n_{free} \tag{5.1}$$

where $\beta_i(t_f)$ represents a generic free component at the final time and $\delta_i \in \mathbb{R}$ is the associated perturbation, defined as:

$$\delta_i = \rho v_i \tag{5.2}$$

where $\rho \in \mathbb{R}_{>0}$ is the perturbation size and v_i is a coefficient sampled from a uniform distribution $\mathcal{U}(-1, 1)$. There is still a constraint which must be considered: the Hamiltonian, evaluated at the final time, must be zero because of the transversality condition. In order to meet this constraint all the free components are perturbed, except one denoted $\beta_j^{old}(t_f)$ and referred as unknown variable, so $\beta_j^{new}(t_f) = \beta_j^{old}(t_f) + \delta_j$ is still to be determined. Let us define an auxiliary function, $g : \mathbb{R} \to \mathbb{R}$, as follows:

$$g(\delta_j) = H(x^{new}(t_f), c_*(t_f), \lambda^{new}(t_f)) \tag{5.3}$$

The perturbation of the unknown variable is then evaluated by solving the root-finding problem: $g(\delta_j) = 0$. The perturbed final state can be numerically propagated backward in time to obtain an optimal trajectory. It must be highlighted that the trajectory obtained does not start where the nominal trajectory starts but it ends where the nominal trajectory ends. Note that in this procedure the costate normalization factor is kept constant, so it is common to all the back-propagated trajectories. In order to populate the database, each optimal trajectory is sampled at random time instants and the sets composed by the states, costates, transfers cost and controls are stored in the database.

5.1.2. Trust Band Definition

The optimal trajectories generated by exploiting the backward generation method could be very different in nature with respect to the nominal trajectory. To design a neurocontroller, whose purpose is to guide the spacecraft from the vicinity of the departure planet to the arrival planet, it is desirable to generate trajectories close to the nominal one. The reason is that the neural networks can focus on learning patterns over a limited region of space, and thus smaller databases and simpler networks can be adopted in the training process. To this aim, the perturbation size in Equation (5.2) is finely tuned so that the initial positions of the back-propagated trajectories lie within a circle centered on the departure planet, the radius of the circle being referred as maximum distance. The result is a beam of trajectories which presents a thickness constantly decreasing going from the departure planet to the arrival planet, indeed it is worth remembering that all the trajectories converge to the same final position. The region defined by the beam of trajectories, from which the database samples are obtained, will be referred as trust band. For the sake of clarity, consider as example the transfer between the fictitious planets *Helicon* and *Terminus*[1]. An example of the effect of the perturbation size is reported in Figure 5.1 where high, medium and low perturbation sizes are used to generated the optimal trajectories. Note that the concept of trust band can be clearly appreciated in the last figure.

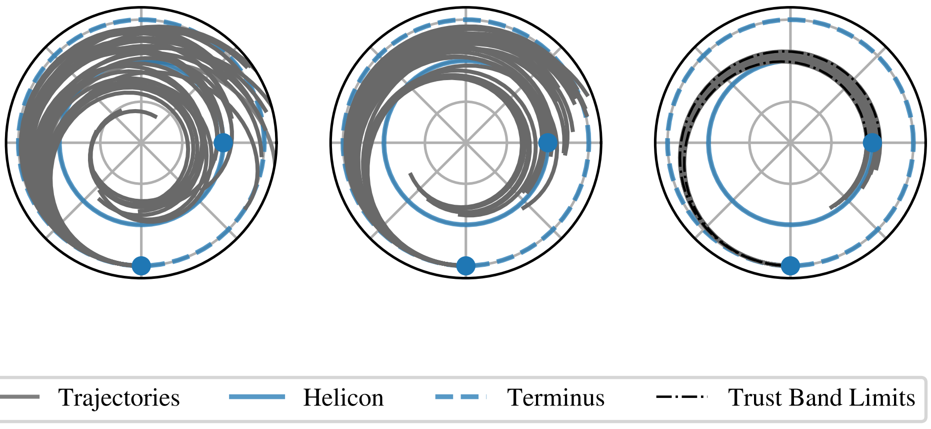


Figure 5.1: Effects of the perturbation size.

It must be noticed that the neural networks are trained on samples belonging to the trust band, so the neurocontroller is expected to be effective in driving the spacecraft from any position (and overall meaningful spacecraft state) within the trust band to the

[1]Isaac Asimov, *Foundation Series* [3].

target planet. Note that, during the guidance, the networks are tested on their ability to generalize, since the spacecraft states are most likely different from those seen during training. Nevertheless, if the spacecraft remains within the trust band the networks are tested on samples similar to those considered during training. Therefore, the goal is to keep the spacecraft within the trust band as much as possible to increase the probability of reaching the target planet. It should be noted, however, that as soon as the spacecraft approaches the arrival planet, the trust band narrows, and thus, if the guidance is not perfect, there is a high probability of leaving this region and thus the neurocontroller may experience a drop in performance because over generalization capabilities are requested to the neural networks.

5.2. Neural Network Models

In this work, standard neural networks and physics-informed neural networks will be implemented within the neurocontroller. The main difference between these networks is the way the costate is predicted from the information about the spacecraft state, the optimal control is then evaluated according to Equation (4.18).

5.2.1. Standard Neural Networks

Standard neural networks directly predict the costate given the state of the spacecraft. Therefore, provided a database composed by pairs of optimal states and optimal costates: $\mathcal{D} := \{(x_*^{(i)}, \lambda_*^{(i)})\}_{i=1}^m$, the training of the networks consists on minimizing the loss function:

$$l(\theta \,|\, \mathcal{D}) = \frac{1}{5m} \sum_{i=1}^{m} \|\lambda_*^{(i)} - \mathcal{N}(x_*^{(i)} \,|\, \theta)\|_2^2 \tag{5.4}$$

where the neural network function is defined as: $\mathcal{N}(\cdot \,|\, \theta) : \mathbb{R}^5 \to \mathbb{R}^5$. The main limitation of this approach is that the network heavily rely on the training data, thus the selection of the database is crucial. In addition there are no constraints, derived from the physics of the problem, imposed during the training process which can increase the reliability of the networks. A scheme of standard neural networks is reported in Figure 5.2.

5.2.2. Physics-Informed Neural Networks

The aim of physics-informed neural networks is to overcome the data-dependent nature of standard neural networks. In particular, the networks try to find the best approximation of the value function, i.e. $\nu(\cdot) \approx \mathcal{N}(\cdot, \theta)$, where $\mathcal{N}(\cdot \,|\, \theta) : \mathbb{R}^5 \to \mathbb{R}$. Therefore, physics-

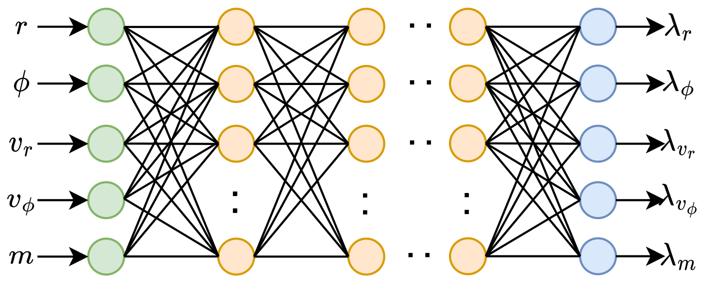


Figure 5.2: Schematic representation of a standard neural network.

informed networks aim to predict the optimal transfer cost from the given spacecraft state. The costate is then obtained as the gradient of the neural network function. It's worth remembering that, for the problem under consideration, the value function is exclusively a function of the state, therefore the network has access to all the information needed to correctly predict the transfer cost. The value function must satisfy the Hamilton-Jacobi-Bellman partial differential equation, formulated in chapter 4, and reported here for simplicity:

$$\begin{cases} \arg\min_{\bar{c}\in C} H(X, \bar{c}, \partial_X \nu(X)) = 0 & X \in \mathbb{R}^5 \quad (5.5a) \\ \nu(X) = 0 & \forall X \in \mathbb{R}^5 : \psi(X) = 0 \quad (5.5b) \end{cases}$$

where the minimizer can be computed as $\bar{c} = Q(X, \partial_X \nu(X))$. In theory, it is possible to construct a physics-informed neural network which is trained to satisfy the previous differential equation. Strictly speaking, a database of optimal trajectories is not required to achieve this objective: fictitious spacecraft states can be randomly sampled and the previous relations can be imposed as soft constraints during training. However, it is not easy to define the sampling process: although some limits on the components of the state can be derived from the physics of the problem, it is not clear how to combine them to form a meaningful spacecraft state, i.e. a condition similar to what the spacecraft encounters during transfer. Therefore, a more straightforward approach is adopted: instead of trying to satisfy Equation (5.5a) on random states, this condition is imposed on the states included in the database. Therefore, given a database of optimal states: $\mathcal{D} := \{(x_*^{(i)})\}_{i=1}^{m}$, the following loss term can be considered during the training process:

$$l_H(\theta \,|\, \mathcal{D}) = \frac{1}{m}\sum_{i=1}^{m}[H(x_*^{(i)}, Q(x_*^{(i)}, \lambda_{\mathcal{N}}^{(i)}), \lambda_{\mathcal{N}}^{(i)})]^2 \quad (5.6)$$

where $\lambda_{\mathcal{N}}^{(i)} = \partial_x \mathcal{N}(x_*^{(i)} \,|\, \theta)$, i.e the costate is computed as the gradient of the neural network function evaluated at the given state belonging to the database. Nevertheless, it is not sufficient to satisfy just Equation (5.5a) because, in order to correctly solve the differential problem, the boundary condition, Equation (5.5b), must be considered. To this aim, the straightforward option would be to sample random states, $\tilde{x}^{(i)} \in \mathbb{R}^5$ such that $\psi(\tilde{x}^{(i)}) = 0$ for $i = 1, ..., m$, to minimize the following loss term:

$$l_{bc}(\theta \,|\, \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} [\mathcal{N}(\tilde{x}^{(i)} \,|\, \theta)]^2 \tag{5.7}$$

Although, this is not the approach followed in this work. Instead, given a database composed by pairs of optimal states and optimal transfers cost: $\mathcal{D} := \{(x_*^{(i)}, \nu_*^{(i)})\}_{i=1}^{m}$ the boundary conditions of the problem are indirectly imposed by considering the loss term:

$$l_{\nu}(\theta \,|\, \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} [\nu_*^{(i)} - \mathcal{N}(x_*^{(i)} \,|\, \theta)]^2 \tag{5.8}$$

Finally, an additional loss term can be considered, this component leverages the fact that the gradient of the value function must correspond to the costate introduced in the Euler-Lagrange equations. Therefore, given a database composed by pairs of optimal states and optimal costates: $\mathcal{D} := \{(x_*^{(i)}, \lambda_*^{(i)})\}_{i=1}^{m}$, the following loss term can be included in the training process:

$$l_{\lambda}(\theta \,|\, \mathcal{D}) = \frac{1}{5m} \sum_{i=1}^{m} \|\lambda_*^{(i)} - \lambda_{\mathcal{N}}^{(i)}\|_2^2 \tag{5.9}$$

where it worth remembering that $\lambda_{\mathcal{N}}^{(i)} = \partial_x \mathcal{N}(x_*^{(i)} \,|\, \theta)$. The loss components introduced can be combined together in different ways to build the loss function and this aspect will be addressed in details in the chapter dedicated to the case studies. A scheme of physics-informed neural networks is reported in Figure 5.3.

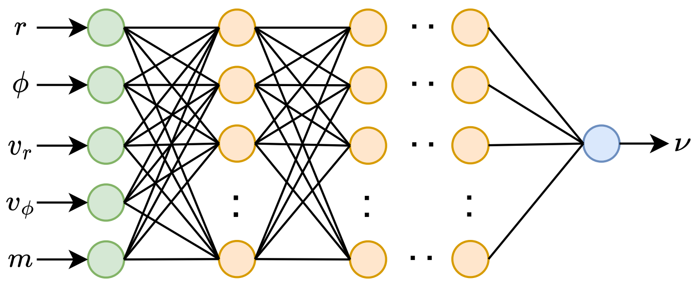


Figure 5.3: Schematic representation of a physics-informed neural network.

6 | Case Studies

Each problem that I solved became a rule which served afterwards to solve other problems.

René Descartes

6.1. Introduction

Th research will analyze two case studies in which the main difference is the value of the homotopy coefficient considered in the definition of the optimal control problem. The first case study is a transfer from Earth to Venus, and for this problem the coefficient is set to $\epsilon = 1$. In the second case study, a transfer from Earth to Mars, the coefficient is selected as $\epsilon = 10^{-5}$. In this case, note that the value of the coefficient is sufficiently small such the problem can be considered a fuel-optimal transfer. The physical constants adopted for the case studies are given in Table 6.1.

Table 6.1: Physical constants.

Quantity	**Value**
Astronomical Unit (AU)	1.496×10^{8} km
Sun Gravitational Parameter	1.327×10^{11} km^3/s^2
Standard Gravitational Acceleration	9.807×10^{-3} km/s^2
Sun-Earth Distance	1.000 AU
Sun-Venus Distance	0.723 AU
Sun-Mars Distance	1.524 AU

To formulate the equations of motion in dimensionless form, the astronomical unit is chosen as characteristic length and the characteristic time is selected so that the Sun's dimensionless gravitational parameter, i.e. the coefficient a_3 in Equations (4.3), is unitary.

6.2. Neural Networks

The numerical implementation of neural networks is based on TensorFlow 2 [1]. It must be noted that the training functions of physics-informed networks are coded from scratch due to the composite nature of the loss functions. In addition, in order to train the neural networks with L-BFGS, an interface is created with the implementation of the method available in the TensorFlow Probability library.

6.2.1. Models

For each case study, four different neural networks models are investigated. The first model is a standard neural network denoted $\mathbb{S}$ while the remaining models are physics-informed networks denoted $\mathbb{P}_1$, $\mathbb{P}_2$ and $\mathbb{P}_3$. The composite loss functions exploited to train physics-informed networks are constructed from the individual loss terms introduced in Chapter 5 as follows:

$$l_{\mathbb{P}_1}(\theta \,|\, \mathcal{D}) = l_\nu(\theta \,|\, \mathcal{D}) + \mu_\lambda \, l_\lambda(\theta \,|\, \mathcal{D}) \tag{6.1}$$

$$l_{\mathbb{P}_2}(\theta \,|\, \mathcal{D}) = l_\nu(\theta \,|\, \mathcal{D}) + \mu_H \, l_H(\theta \,|\, \mathcal{D}) \tag{6.2}$$

$$l_{\mathbb{P}_3}(\theta \,|\, \mathcal{D}) = l_\nu(\theta \,|\, \mathcal{D}) + \mu_\lambda \, l_\lambda(\theta \,|\, \mathcal{D}) + \mu_H \, l_H(\theta \,|\, \mathcal{D}) \tag{6.3}$$

where $\mu_\lambda, \mu_H \in \mathbb{R}$ are the weights introduced to balance the different terms. In this work, the weights are set to $\mu_\lambda = 10$ and $\mu_H = 1$ but the selection process remains a critical open point because of the dramatic impact on the training outcome. Recall that the database, which is common to all neural networks, is defined as $\mathcal{D} = \{(x_*^{(i)}, \lambda_*^{(i)}, \nu_*^{(i)}, u_*^{(i)}, \alpha_*^{(i)})\}_{i=1}^m$, and collects the states, costates, transfers costs and controls sampled from the optimal trajectories.

6.2.2. Training Process

In the analysis, the database is divided into training, validation and test datasets with the proportions $60\% - 20\% - 20\%$. The standard neural networks are trained with ADAM with a learning rate of 1×10^{-4} for a maximum number of iterations set to 20000 while the physics-informed networks are trained for 1000 iterations with ADAM with a learning rate of 5×10^{-4}, and then with L-BFGS for a maximum number of iterations of 2000. The remaining settings of ADAM and L-BFGS are kept at their default values, except for the number of correction pairs to approximate the Hessian matrix, which is set to 50. In training physics-informed neural networks, ADAM is used to properly initialize the search for minima because L-BFGS tends to stop in the early phase on local minima.

In order to avoid the overfitting of the training dataset the training is stopped after the loss function has not decreased by at least 1×10^{-7} in the last 100 iterations on the validation dataset. The adoption of this technique is particularly useful for comparing different neural network models. In fact, the goal is to obtain the best performance for each neural network, and depending on the model, this may require more or fewer iterations. Therefore, the methodology is exploited to avoid setting a predetermined number of iterations while still getting the most out of the training process.

6.2.3. Architectures

For all neural networks models, the hyperbolic tangent function is selected as neuron activation function. The reason is that, at the preliminary stage, the hyperbolic tangent function proved to be superior to other activation functions such as the sigmoid and the softplus. The hyperbolic tangent function, $\tanh := \mathbb{R} \to [-1, 1]$ is defined as:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{6.4}$$

In particular, in the neural networks the adaptive version of the function is implemented with a scale factor of 10 and an initial slope of 1/10. Several architectures, which differ in number of hidden layers and number of neurons per hidden layer, are trained for each model. The best architecture is then selected based on the following metric which is evaluated on the validation dataset:

$$\Delta_{u-\alpha} = \sqrt{\Delta_u \Delta_\alpha} \tag{6.5}$$

where the individual components of this metric are defined as follows:

$$\Delta_u = \frac{1}{m} \sum_{i=1}^{m} |u_*^{(i)} - u_{\mathcal{N}}^{(i)}| \tag{6.6a}$$

$$\Delta_\alpha = \frac{1}{m} \sum_{i=1}^{m} \arccos(\langle \alpha_*^{(i)}, \alpha_{\mathcal{N}}^{(i)} \rangle) \tag{6.6b}$$

and $u_{\mathcal{N}}^{(i)}$ and $\alpha_{\mathcal{N}}^{(i)}$ are the controls predicted by the networks trough Equation (4.18):

$$c_{\mathcal{N}}^{(i)} = \begin{bmatrix} u_{\mathcal{N}}^{(i)} \\ \alpha_{\mathcal{N}}^{(i)} \end{bmatrix} = Q(x_*^{(i)}, \lambda_{\mathcal{N}}^{(i)}) \qquad i = 1, ..., m \tag{6.7}$$

i.e. considering the optimal states included in the databases and the corresponding predictions of the costates by the networks. It is worth remembering that $\lambda_{\mathcal{N}}^{(i)} = N(x_*^{(i)} \,|\, \theta)$ in case of standard neural networks and $\lambda_{\mathcal{N}}^{(i)} = \partial_x N(x_*^{(i)} \,|\, \theta)$ in case of physics-informed networks. Note that the metric defined in Equation (6.5) is only a quantity useful for comparing architectures on the basis of a single number. To conclude, note that $\mathbb{P}_1$, $\mathbb{P}_1$ and $\mathbb{P}_3$ can be tested also on the ability to predict the transfer cost from a given state of the spacecraft, and so the following metric is introduced for this purpose:

$$\Delta_\nu = \frac{1}{m} \sum_{i=1}^{m} |\nu_*^{(i)} - N(x_*^{(i)} \,|\, \theta)| \qquad \text{for } \mathbb{P}_1, \mathbb{P}_2, \mathbb{P}_3 \tag{6.8}$$

6.3. Neurocontroller Performance Evaluation

The neurocontroller is evaluated on the trajectories included in the test dataset, in particular, it is asked to propagate the initial states of these trajectories. The resulting trajectory is referred as neural trajectory for simplicity. Note that the final integration time of the neural trajectory is selected to coincide with the time of flight of the corresponding test trajectory. To evaluate the performance of the neurocontroller the following metrics are introduced:

$$\Delta_r = \frac{1}{n} \sum_{i=1}^{n} \sqrt{[r^{(i)}(t_f)]^2 + r_f^2 - 2 r^{(i)}(t_f)\, r_f \,\cos\left(\phi_f - \phi^{(i)}(t_f)\right)} \tag{6.9a}$$

$$\Delta_v = \frac{1}{n} \sum_{i=1}^{n} \sqrt{(v_r^{(i)}(t_f))^2 + \left(v_\phi^{(i)}(t_f) - \sqrt{\frac{a_3}{r_f}}\right)^2} \tag{6.9b}$$

$$\Delta_J = \frac{1}{n} \sum_{i=1}^{n} |J_{nominal}^{(i)} - J_{controller}^{(i)}| \tag{6.9c}$$

where n denotes the total number of test trajectories while $r^{(i)}(t_f)$, $\phi^{(i)}(t_f)$, $v_r^{(i)}(t_f)$ and $v_\phi^{(i)}(t_f)$ are the final radial ad angular positions and the final radial and tangential velocities of the spacecraft after propagating the trajectory with the neurocontroller. It is worth remembering that r_f and ϕ_f are the radial and angular positions of the target planet. To conclude, $J_{nominal}^{(i)}$ denotes the values of the optimal cost of the transfer while and $J_{controller}^{(i)}$ is the cost of the transfer obtained with the neurocontroller. In other terms, the first two metrics quantify the average final position and velocity errors relative to the target while the last metric is used to evaluate the optimality of the neurocontroller in steering the spacecraft to the final destination.

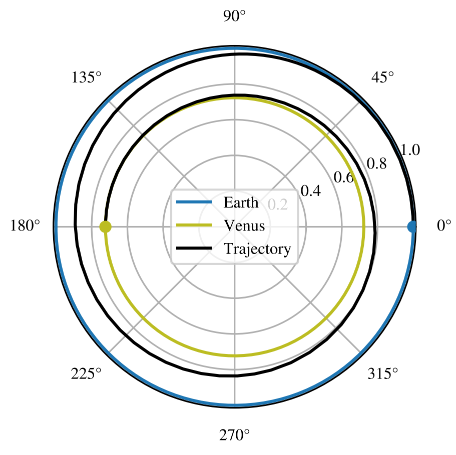
6.4. Earth-Venus Transfer

The first case study considered is the transfer from Earth to Venus. The characteristics of the spacecraft and the data of the problem are reported in Table 6.2.

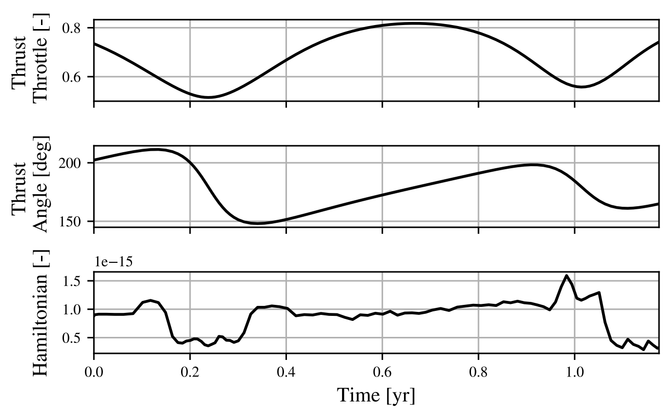
Table 6.2: Earth-Venus: data of the problem.

Quantity	Value
Maximum Thrust	0.30 N
Specific Impulse	3000 s
Initial Mass	1500 kg
Planets Angular Separation	180 deg
Homotopy Coefficient	1

The nominal trajectory is characterized by a total time of flight of $t_f = 1.1755\,\mathrm{yr}$ and a propellant mass consumption of 256.78 kg. The trajectory is shown in Figure 6.1a while the time histories of the optimal control and the Hamiltonian, evaluated along the nominal trajectory, are reported in Figure 6.1b. In particular, it can be noticed that the Hamiltonian is basically zero along the trajectory, as predicted by the theory.



(a)



(b)

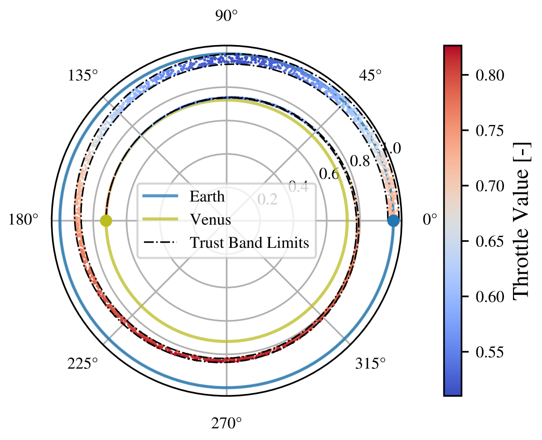
Figure 6.1: Earth-Venus: nominal trajectory.

The characteristics of the database and the parameters selected to generate it from the nominal trajectory are reported in Table 6.3. The small number of trajectories is chosen to speed up the training process but the limited amount of data can be theoretically overcome with physics. In fact, the physical constraints imposed during the training virtually enrich the dataset as neural networks exploit the datapoints to understand the physics of the problem.

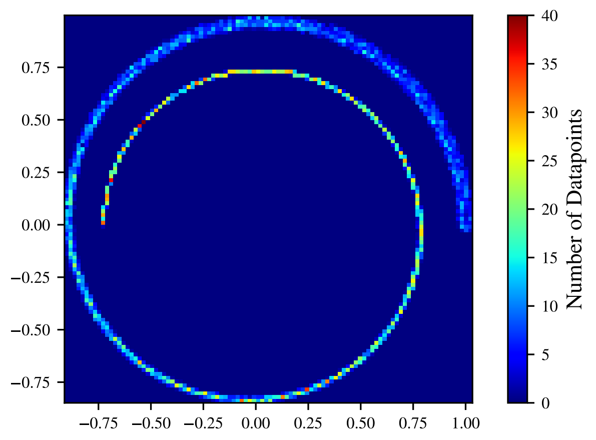
Table 6.3: Earth-Venus: characteristics of the database.

Quantity	**Value**
Maximum Distance	5×10^6 km
Perturbation Size	10^{-3}
Unknown Variable	$m(t_f)$
Number of Trajectories	100
Number of Samples per Trajectory	100

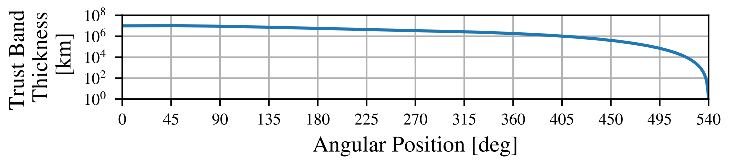
The positions where the datapoints are sampled along the optimal trajectories are displayed in Figure 6.2a, note that only a fraction of the positions is reported. In particular, the positions are highlighted according to the thrust throttle factor value of the corresponding datapoint.



(a)



(b)



(c)

Figure 6.2: Earth-Venus: visualisation of the database.

It is worth noting how the trust band contains within it all the datapoints, as per definition. This band presents a decreasing thickness in the transition from Earth to Venus and in particular, Figure 6.2c shows the evolution of its thickness as a function of the angular position of the spacecraft during the transfer. Finally, in Figure 6.2b the two-dimensional histogram of the datapoints positions, in cartesian coordinates, is reported. With regard to the datapoints, the histograms of the states, costates and transfers costs included in the database are reported in Figure 6.3.

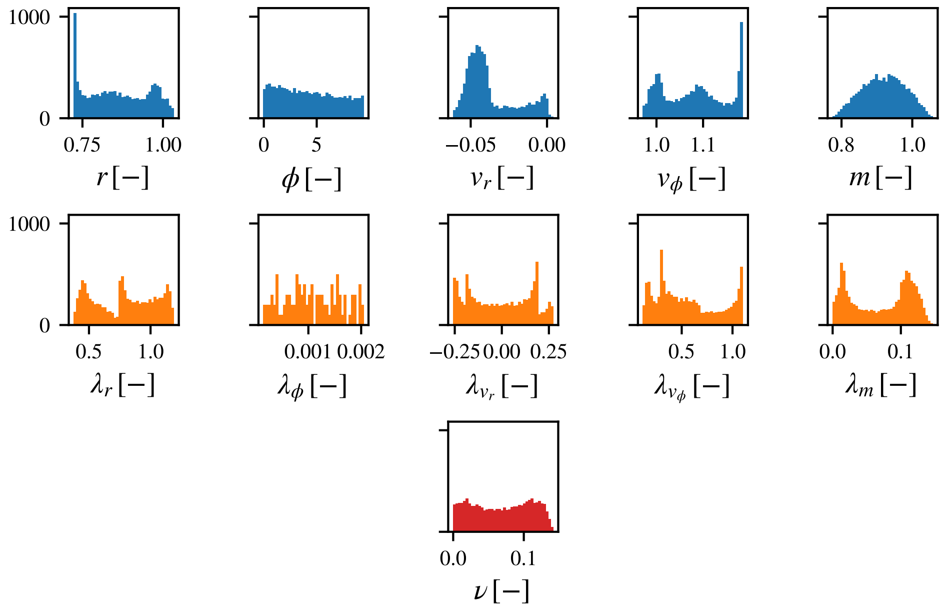


Figure 6.3: Earth-Venus: histograms of the datapoints.

In particular, it can be noticed the peaks in the histograms of the radial position and the tangential velocity of the spacecraft. These peaks are the reflection of datapoint sampling on the last portion of the trajectory where all the trajectories overlap (cf. Figure 6.2b). In addition, it is worth underlying the different range characterizing the entries of the database. To reduce these differences, normalization and standardization techniques of the database elements were attempted, but the performance of the neural networks was reported to be worse, so in the end the raw elements were provided to the networks. In Table 6.4 the metric defined in Equation (6.5) is reported for different architectures of the neural networks models considered.

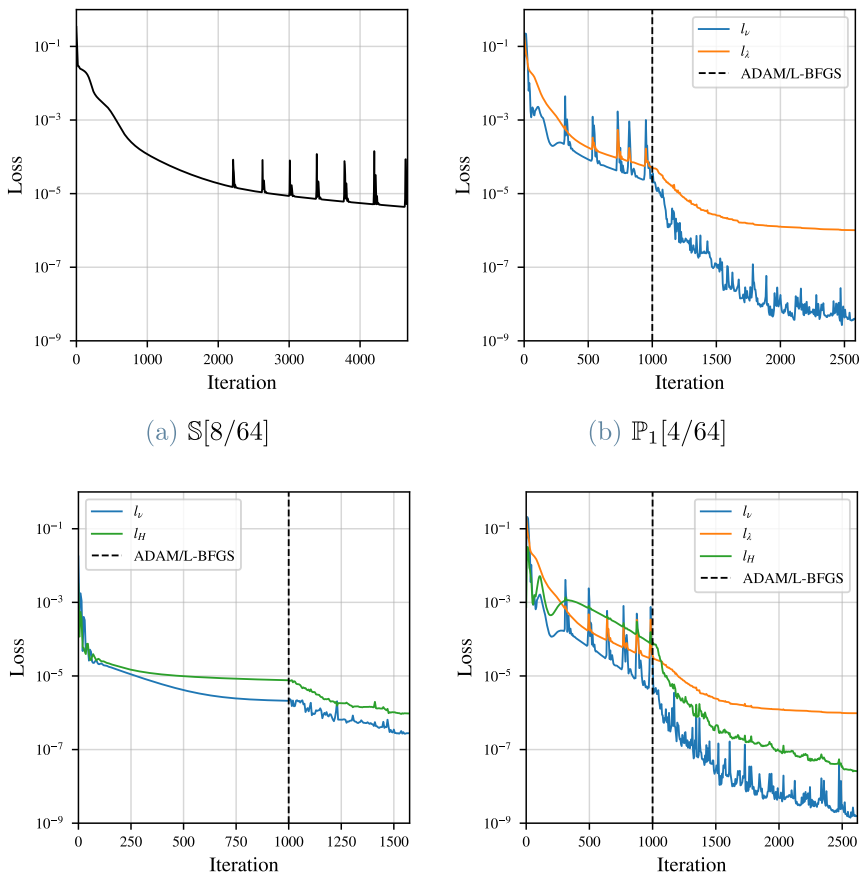
Table 6.4: Earth-Venus: comparison between architectures.

	4/32[*]	4/64	8/32	8/64
$\mathbb{S}$	1.72×10^{-3}	2.07×10^{-3}	1.96×10^{-3}	$\mathbf{1.49 \times 10^{-3}}$[△]
$\mathbb{P}_1$	1.30×10^{-3}	$\mathbf{9.08 \times 10^{-4}}$	1.13×10^{-3}	3.73×10^{-3}
$\mathbb{P}_2$	$\mathbf{1.15 \times 10^{-1}}$	1.66×10^{-1}	1.63×10^{-1}	1.65×10^{-1}
$\mathbb{P}_3$	1.06×10^{-3}	$\mathbf{8.79 \times 10^{-4}}$	1.19×10^{-3}	3.30×10^{-3}

[*] number of hidden layers/number of neurons per hidden layer

[△] bold font indicates the best results for each model

It can be seen that the standard neural network performs better with a larger architecture, whereas this is generally not the case for physics-informed networks. Figure 6.4 shows the evolution of the loss function during the training process for the best architectures of the various models.



(a) $\mathbb{S}[8/64]$ (b) $\mathbb{P}_1[4/64]$ (c) $\mathbb{P}_2[4/32]$ (d) $\mathbb{P}_3[4/64]$

Figure 6.4: Earth-Venus: loss evolution.

The best architectures are evaluated on the ability to predict the optimal controls and the transfers costs from the states included the test dataset. The values of the metrics defined in Equation (6.6) and in Equation (6.8) are reported in Table 6.5. It can be noticed that the neural networks are accurate in the predictions with the only exception of $\mathbb{P}_2$. Note that, according to the split between train, validation and test datasets, the results are based on 2000 datapoints.

Table 6.5: Earth-Venus: prediction errors by the networks.

	Thrust Throttle Error (Δ_u) $[-]$	**Thrust Angle Error** (Δ_α) [deg]	**Transfer Cost Error** (Δ_ν) [kg]
$\mathbb{S}$ [8/64][*]	7.66×10^{-4}	0.230	-
$\mathbb{P}_1$ [4/64]	5.88×10^{-4}	0.140	0.08
$\mathbb{P}_2$ [4/32]	8.55×10^{-2}	16.050	0.61
$\mathbb{P}_3$ [4/64]	5.75×10^{-4}	0.136	0.04

[*] indicates the best model architecture

The best architectures are also implemented in the neurocontroller to guide the spacecraft in closed loop. Table 6.6 shows the position, velocity and optimality errors as defined in Equations (6.9). It is worth underlying that the results are based on 20 neural trajectories.

Table 6.6: Earth-Venus: performance of the neurocontrollers.

	Position Error (Δ_r) [km]	**Velocity Error** (Δ_v) $[\mathrm{km\,s^{-1}}]$	**Optimality Error** (Δ_J) [kg]
$\mathbb{S}$ [8/64][*]	1.615×10^5	1.692×10^{-2}	0.05
$\mathbb{P}_1$ [4/64]	1.465×10^5	1.586×10^{-2}	0.04
$\mathbb{P}_2$ [4/32]	1.109×10^7	2.643	13.31
$\mathbb{P}_3$ [4/64]	1.571×10^5	1.604×10^{-2}	0.04

[*] indicates the best model architecture

Clearly, the final errors are not negligible and a corrective maneuver would be needed to close the gap to rendezvous with the target planet. Despite the good prediction capabilities of the controls by the networks (cf. Table 6.5), it is evident that the accumulation of these small inaccuracies leads to significant final position and velocity errors by controlling the spacecraft with the neurocontroller. In this regard, it must be emphasized that the optimality error does not give a complete picture because the cost of the additional maneuver should be added to the cost of the transfer performed with the neurocontroller. It can be noticed that physics-informed networks perform slightly better than the standard network. The neurocontroller based on $\mathbb{P}_2[4/32]$ presents the worst performance: it seems that the network is unable to replicate the optimal controls. The problem is not necessarily due to the way the loss function is conceptually conceived, but it could be the way the function is mathematically formulated. In other words, the issue may be the minimization of this type of loss function. To try to shed light on this, the loss landscape around the point of convergence for $\mathbb{P}_2[4/32]$ is compared with that of $\mathbb{P}_3[4/64]$, which is the best network according the metric given in Table 6.4. To cross-check the consistency of the results, three different couples of random directions are considered in the evaluation of the landscape. The results are reported in Figure 6.5 and 6.6.

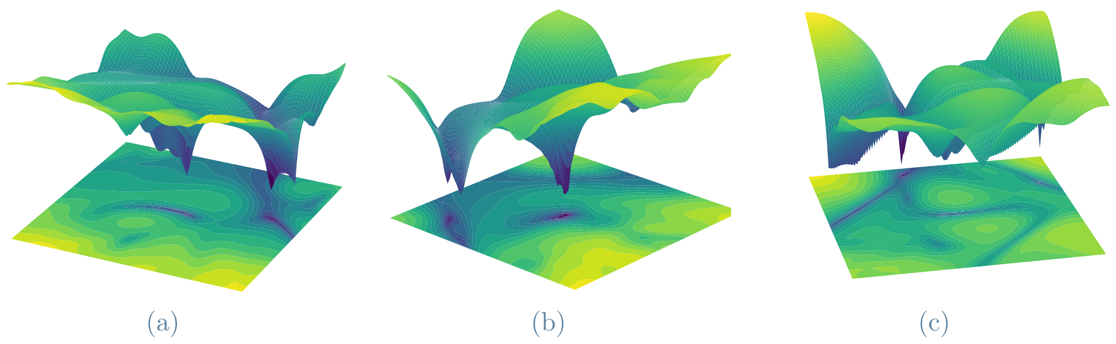


Figure 6.5: Earth-Venus: loss landscapes for $\mathbb{P}_2[4/32]$.

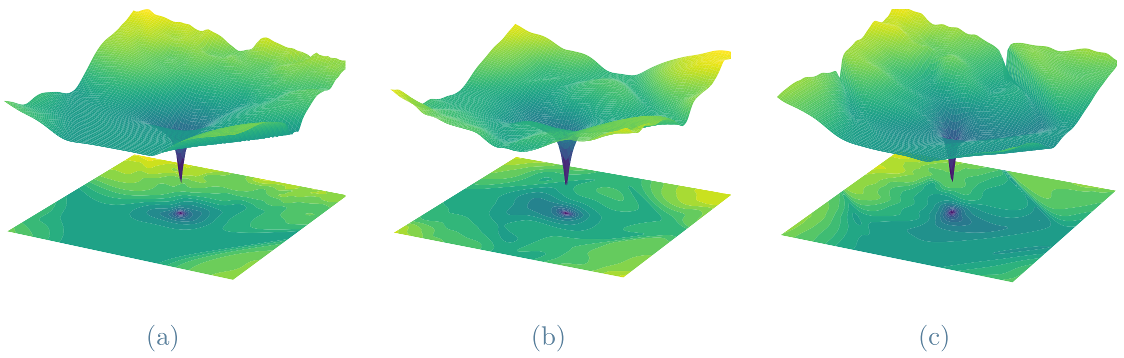
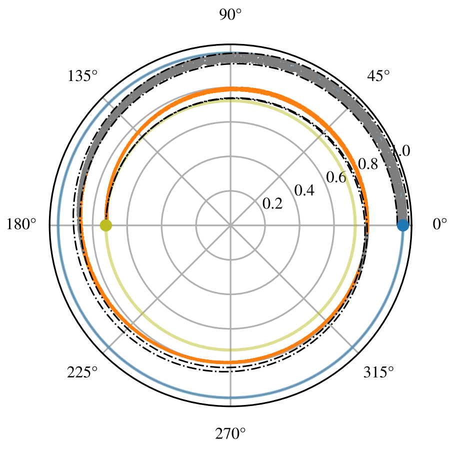
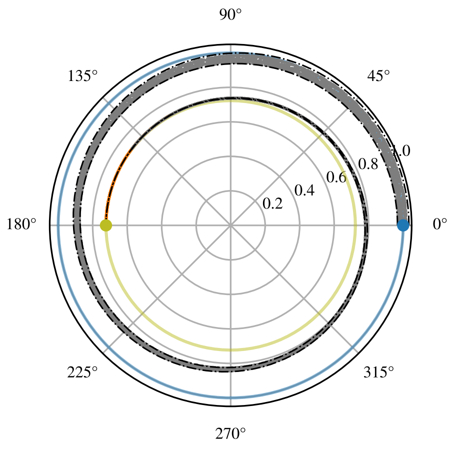


Figure 6.6: Earth-Venus: loss landscapes for $\mathbb{P}_3[4/64]$.

It is evident that the loss landscape of $\mathbb{P}_2[4/32]$ is jagged and presents more than one depression, in contrast to the loss landscape of $\mathbb{P}_3[4/64]$ which appears regular and shows a single minimum. The conformation of the landscape of $\mathbb{P}_2[4/32]$ can be the main obstacle to the minimization of the associated loss function: it is reasonable to assume that the optimizer may get stuck on local minima. To overcome this difficulty, the first possibility might be to modify the architecture of the neural network, for example by considering a more sophisticated model than the feedforward one, to capture the complexity of the Hamilton-Jacobi-Bellman equation. The second alternative could be to change the way the partial differential equation is exploited to formulate the loss function. It can be pointed out in this regard how the term based on the relationship between the two optimality principles integrates well in the construction of the loss function while at the same time increasing the reliability of the network. The trajectories propagated with the neurocontrollers based on $\mathbb{P}_2[4/32]$ and $\mathbb{P}_3[4/64]$ are shown in Figure 6.7.



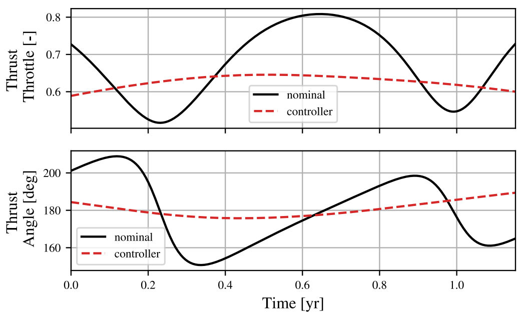
(a) $\mathbb{P}_2[4/32]$



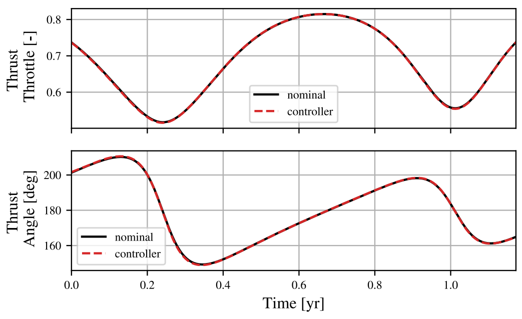
(b) $\mathbb{P}_3[4/64]$

Figure 6.7: Earth-Venus: comparison between neurocontrollers trajectories.

Note that grey and orange colors indicate whether the trajectory in inside or outside the trust band, respectively. It can be seen that the neurocontroller based on $\mathbb{P}_2[4/32]$ takes the spacecraft out of the trust band very early. The main consequence is an increase in the final errors because, it should be remembered, the networks are not trained to predict the optimal controls outside the band. In Figure 6.8a it can be seen that the neurocontroller based on $\mathbb{P}_2[4/32]$ is generally unable to replicate the optimal control profile. Note that for the comparison the trajectories with the highest final position errors are selected among those propagated with the neurocontrollers.



(a) $\mathbb{P}_2[4/32]$



(b) $\mathbb{P}_3[4/64]$

Figure 6.8: Earth-Venus: comparison between controls for the worst neural trajectories.

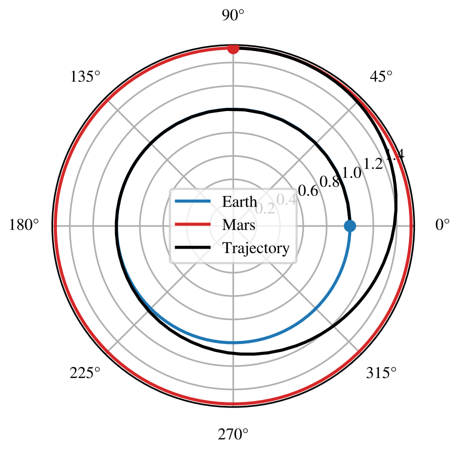
6.5. Earth-Mars Transfer

The second case study is the fuel-optimal transfer from Earth to Mars. Table 6.7 summarizes the data of the problem.

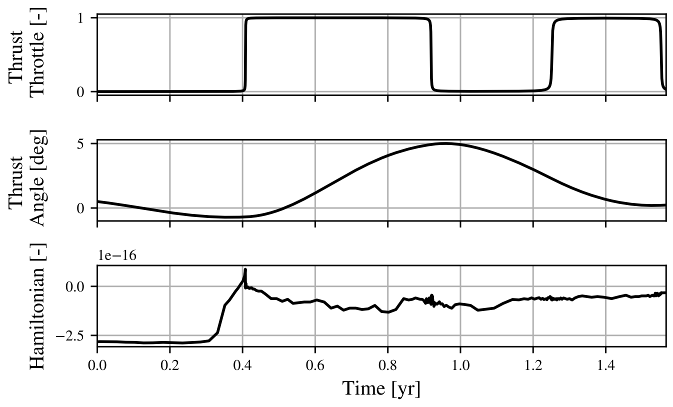
Table 6.7: Earth-Mars: data of the problem.

Quantity	Value
Maximum Thrust	0.30 N
Specific Impulse	3000 s
Initial Mass	1500 kg
Planets Angular Separation	90 deg
Homotopy Coefficient	10^{-5}

The nominal trajectory is characterized by a time of flight of $t_f = 1.5662$ yr with a propellant mass consumption of 260.92 kg. The trajectory and the time histories of the optimal control and the Hamiltonian are reported in Figure 6.9a and Figure 6.9b, respectively.



(a)



(b)

Figure 6.9: Earth-Mars: nominal trajectory.

The typical bang-bang profile of the thrust throttle factor, characteristic of fuel-optimal problems, can be noted. It is worth remembering that in this case the problem is solved by exploiting an homotopy technique. In this regard, Figure 6.10 shows the time histories of thrust throttle factor for different values of the homotopy coefficient.

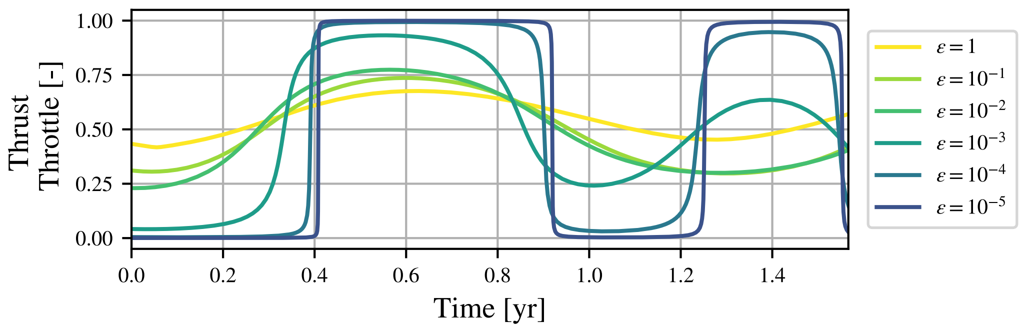


Figure 6.10: Earth-Mars: evolution of profile of the thrust throttle factor.

The parameters defining the database of optimal trajectories are given in Table 6.8. It can be noticed in particular that the unknown variable is different with respect the previous case study.

Table 6.8: Earth-Mars: characteristics of the database.

Quantity	**Value**
Maximum Distance	5×10^6 km
Perturbation Size	10^{-4}
Unknown Variable	$\lambda_\phi(t_f)$
Number of Trajectories	100
Number of Samples per Trajectory	100

In fuel-optimal problems, it turned out to be challenging to control the spread of trajectories, as even small perturbation sizes lead to trajectories very different from the nominal one. The problem could be the structure of the equations in the formulation of the fuel-optimal problem. In addition to the previous issue, it must be underlined the difficulty of achieving a uniform spread of the back-propagated trajectories in the case the maximum distance from the initial position is greater than 5×10^6 km. To clarify this aspect, observe in Figure 6.11 the non-uniform distribution of datapoints obtained by setting the maximum distance to 1×10^7 km and the perturbation size to 1×10^{-3}. In particular, it can be seen that the datapoints are concentrated on the inner part of the beam of trajectories. It can be concluded that the trajectories generated by applying the backward-generation method are for most cases very similar.

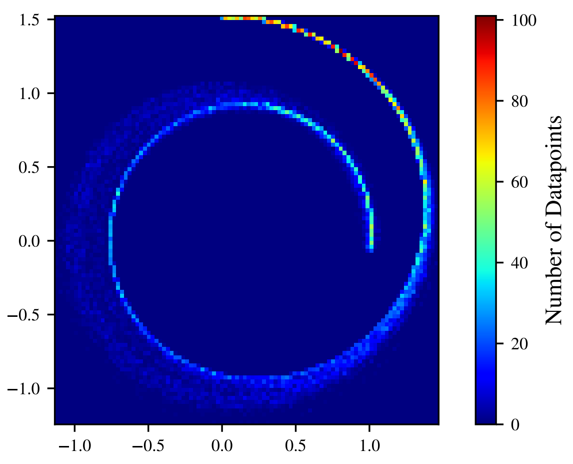


Figure 6.11: Earth-Mars: example of non-uniform distribution of datapoints.

The database can be visualized in Figure 6.12. Note in particular the evolution of the thickness of the trust band, which shows a bulge in the central portion of the transfer. The histograms of the datapoints are reported instead in Figure 6.13.

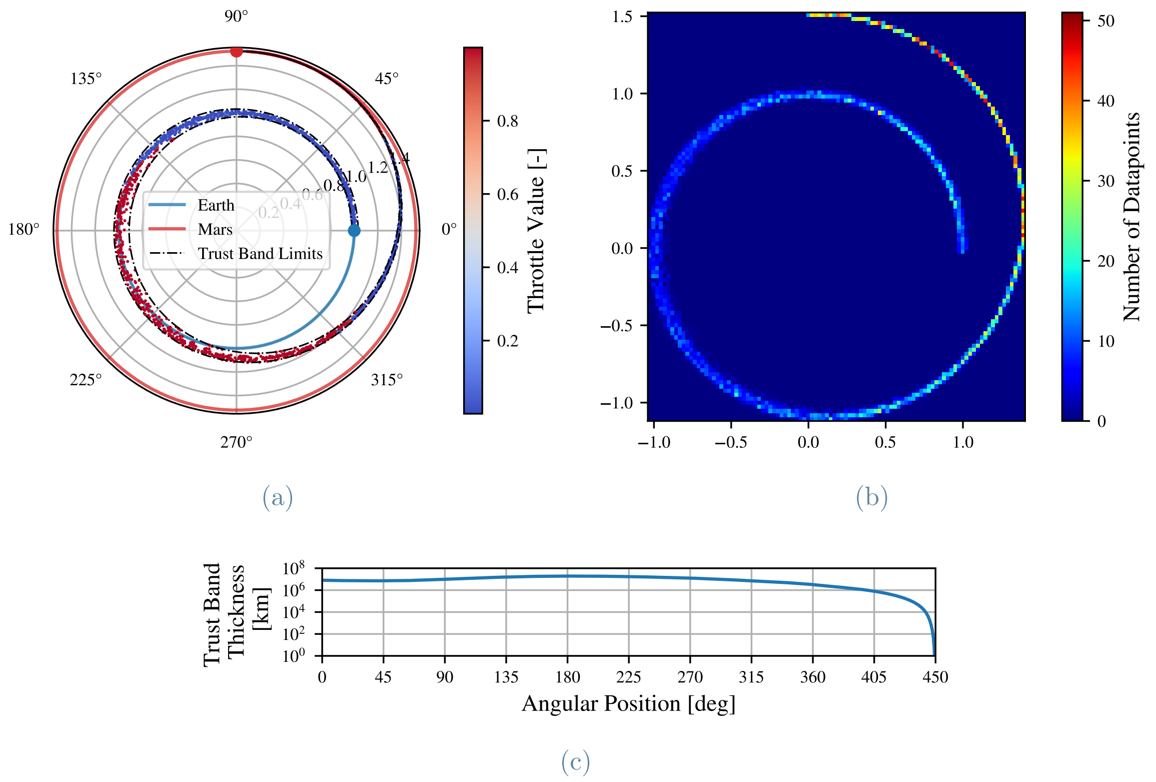


Figure 6.12: Earth-Mars: visualisation of the database.

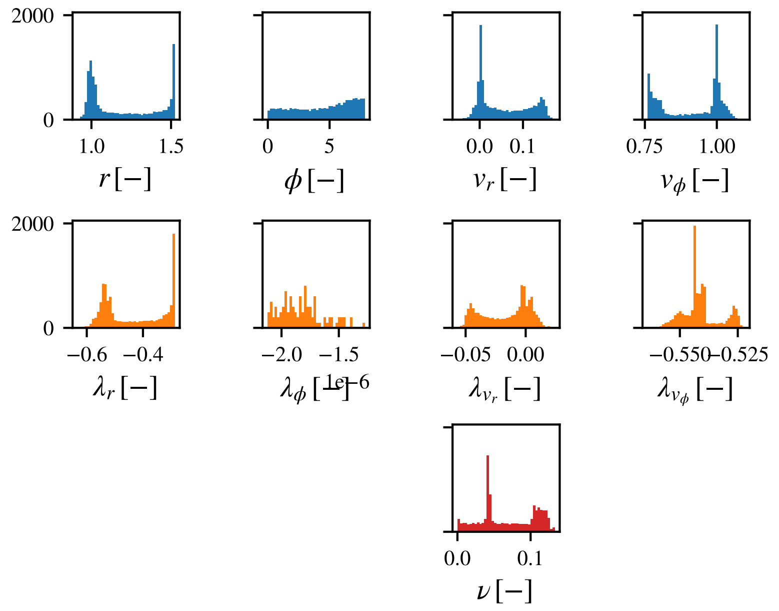


Figure 6.13: Earth-Mars: histograms of the datapoints.

In particular, the presence of a peak can be observed in the histogram of the transfer cost. Note that, for fuel-optimal problems, the transfer cost represents the propellant mass needed. Therefore, the peak is due to sampling in the last portion of the trajectory, from the beginning of the second coasting arc. Indeed, during the coasting arc the thrust is turned off and thus the mass consumed does not increase. After this phase, the trajectories overlap and therefore the propellant mass to get to the final destination is about the same. The metric defined in Equation (6.5), evaluated on the validation dataset, is reported for different architectures of the neural networks models in Table 6.9

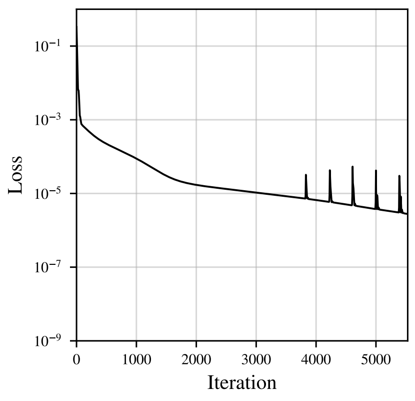
Table 6.9: Earth-Mars: comparison between architectures.

	4/32*	4/64	8/32	8/64
$\mathbb{S}$	2.47×10^{-2}	1.41×10^{-2}	2.39×10^{-2}	$\mathbf{1.24 \times 10^{-2}}^{\triangle}$
$\mathbb{P}_1$	1.17×10^{-2}	$\mathbf{5.44 \times 10^{-3}}$	1.23×10^{-2}	6.71×10^{-3}
$\mathbb{P}_2$	1.82×10^{-1}	2.87×10^{-1}	2.58×10^{-1}	$\mathbf{1.26 \times 10^{-1}}$
$\mathbb{P}_3$	6.82×10^{-3}	$\mathbf{4.55 \times 10^{-3}}$	1.66×10^{-2}	9.76×10^{-3}

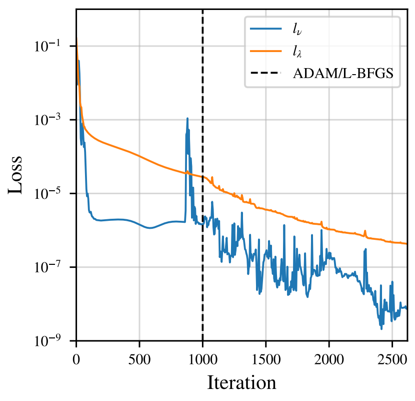
* number of hidden layers/number of neurons per hidden layer

$^{\triangle}$ bold font indicates the best results for each model

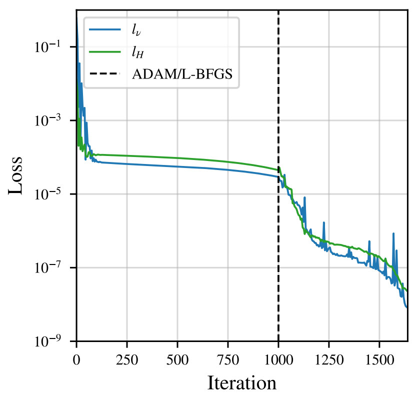
The evolution of the loss during the training process is reported in Figure 6.14.



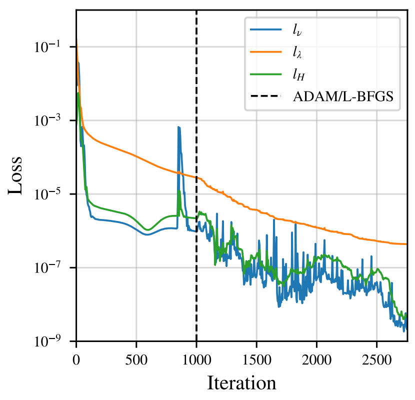
(a) $\mathbb{S}[8/64]$



(b) $\mathbb{P}_1[4/64]$



(c) $\mathbb{P}_2[8/64]$



(d) $\mathbb{P}_3[4/64]$

Figure 6.14: Earth-Mars: loss evolution.

The best architectures are evaluated on the ability to predict the optimal controls and the costs of the transfers on the test dataset. The results are reported in Table 6.10.

Table 6.10: Earth-Mars: prediction errors by the networks.

	Thrust Throttle	**Thrust Angle**	**Transfer Cost Error**
	Error (Δ_u) [$-$]	**Error** (Δ_α) [deg]	**Error** (Δ_ν) [kg]
$\mathbb{S}$ [8/64]*	0.169	0.053	-
$\mathbb{P}_1$ [4/64]	0.038	0.044	0.12
$\mathbb{P}_2$ [8/64]	0.311	2.974	0.09
$\mathbb{P}_3$ [4/64]	0.024	0.045	0.06

* indicates the best model architecture

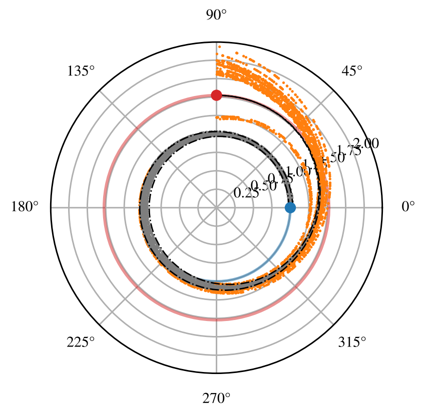
It can be seen that the networks are generally able to predict the optimal controls with small errors, and in addition, physics-informed neural networks can estimate the transfers costs with high accuracy. Therefore, this type of networks may also be useful in the preliminary design of missions to obtain a rough estimate of the required mass associated to different transfer options. With regard to the model $\mathbb{P}_2[8/64]$, the same considerations made for the Earth-Venus problem apply. The best architectures are implemented in the neurocontroller to guide the spacecraft and Table 6.11 shows the position, velocity and optimality errors as defined in Equations (6.9).

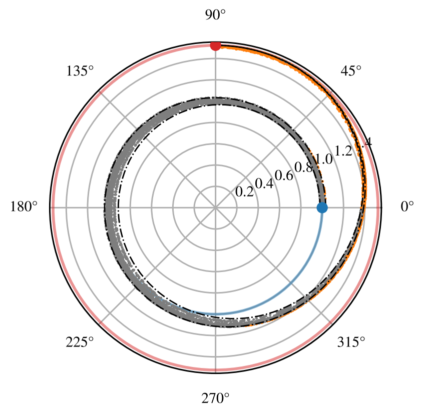
Table 6.11: Earth-Mars: performance of the neurocontrollers.

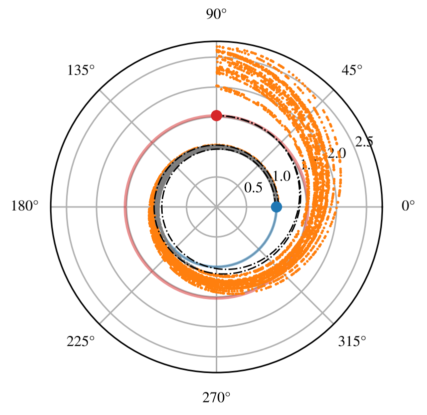
	Position Error	**Velocity Error**	**Optimality Error**
	(Δ_r) [km]	(Δ_v) [km s^{-1}]	(Δ_J) [kg]
$\mathbb{S}$ [8/64]*	6.963×10^7	5.304	78.66
$\mathbb{P}_1$ [4/64]	5.357×10^6	1.984×10^{-1}	3.16
$\mathbb{P}_2$ [8/64]	2.161×10^8	5.882	85.76
$\mathbb{P}_3$ [4/64]	4.591×10^6	1.745×10^{-1}	6.91

* indicates the best model architecture

It can be seen that the final errors are even more pronounced than in the previous case study. Nevertheless, it is interesting to note that the neurocontrollers based on the physics-informed network $\mathbb{P}_1[4/64]$ and $\mathbb{P}_3[4/64]$ perform much better than the one based on the standard neural network. In addition, focusing on the results obtained with the neurocontrollers based on $\mathbb{P}_1[4/64]$ and $\mathbb{P}_3[4/64]$, it is evident that the Hamiltonian-related term considered in the loss function of $\mathbb{P}_3[4/64]$ turn out to be beneficial. The comparison between the trajectories propagated by the neurocontrollers is reported in Figure 6.15.


(a) $\mathbb{S}[8/64]$


(b) $\mathbb{P}_1[4/64]$


(c) $\mathbb{P}_2[8/64]$

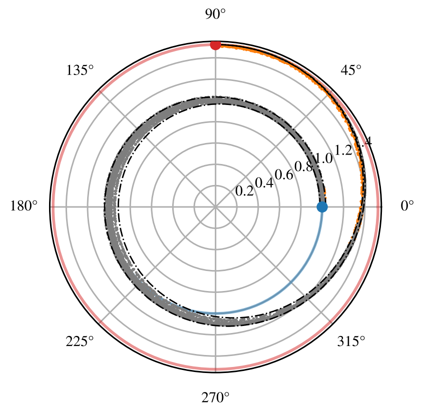
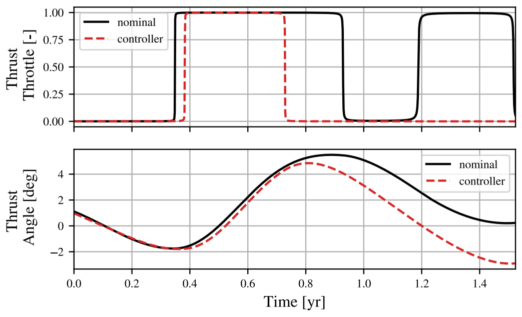
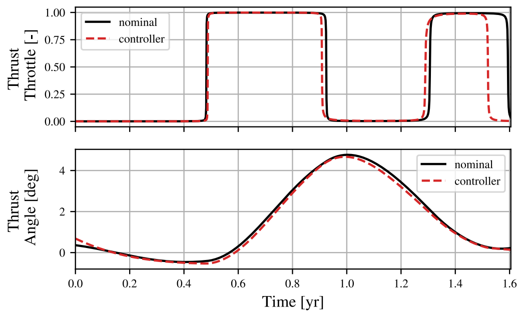

(d) $\mathbb{P}_3[4/64]$

Figure 6.15: Earth-Mars: comparison between neurocontroller trajectories.

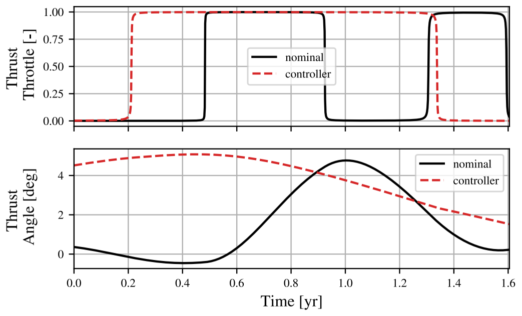
It can be seen that the neurocontrollers based on $\mathbb{S}[8/64]$ and $\mathbb{P}_2[8/64]$ take the spacecraft out of the trust band even before the end of the first burning arc. It can be emphasized again that the sooner the spacecraft leaves the trust band, the larger the final errors. In this case the problem can be clearly appreciated in Figure 6.16. The neurocontrollers based on $\mathbb{S}[8/64]$ and $\mathbb{P}_2[8/64]$ are unable to correctly replicate the second ignition of the engine. Note that the comparison is based on the worst neural trajectories in term of the final position error. Figure 6.17 shows instead the comparison based on the best neural trajectories, where it can be clearly appreciated that $\mathbb{P}_1[4/64]$ and $\mathbb{P}_3[4/64]$ are able to capture the optimal time evolution of the thrust throttle factor.



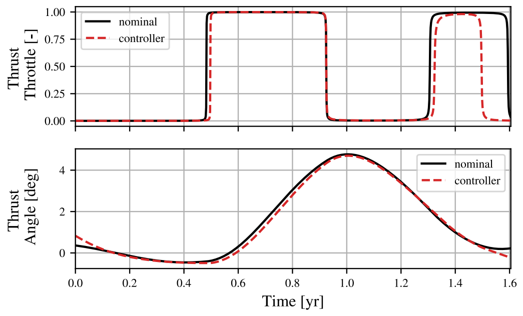
(a) $\mathbb{S}[8/64]$



(b) $\mathbb{P}_1[4/64]$

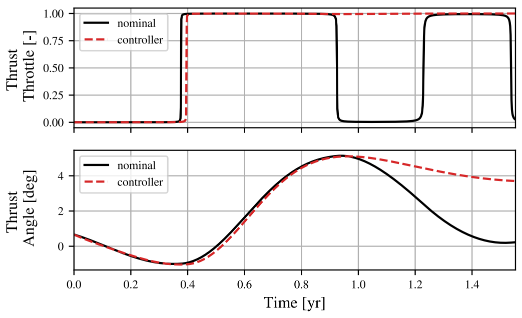


(c) $\mathbb{P}_2[8/64]$

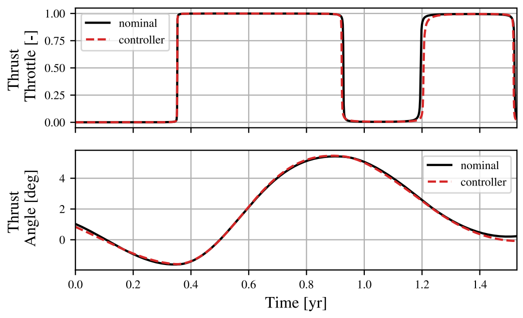


(d) $\mathbb{P}_3[4/64]$

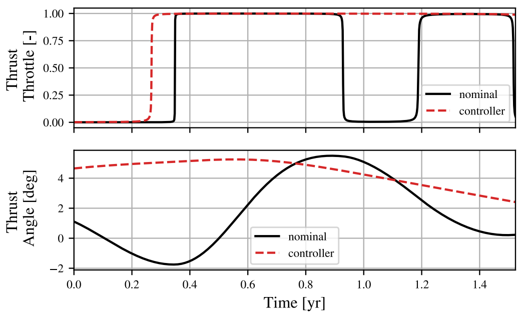
Figure 6.16: Earth-Mars: comparison between controls for the worst neural trajectories.



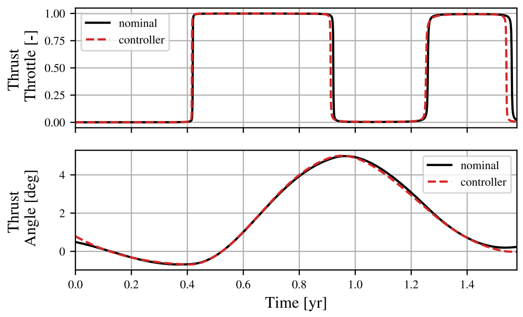
(a) $\mathbb{S}[8/64]$



(b) $\mathbb{P}_1[4/64]$



(c) $\mathbb{P}_2[8/64]$



(d) $\mathbb{P}_3[4/64]$

Figure 6.17: Earth-Mars: comparison between controls for the best neural trajectories.

6.6. Notes on Implementation

To conclude the chapter, the reader is given some details on the numerical implementation of the various steps involved in the reported analysis. In this regard, it is worth underlying that the code is entirely developed in Python.

The numerical integration of the spacecraft trajectory is based on the DOP853 method, an explicit eight-order Runge-Kutta method implemented in the SciPy library [44]. In terms of settings, the absolute tolerance is set to $atol \approx 2.22 \times 10^{-16}$ and the relative tolerance is selected as $rtol = 3 \times 10^{-14}$ in order to accurately integrate the spacecraft dynamics in dimensionless form.

In order to solve the root-finding problems a modification of the Powell hybrid method, as described in MINPACK [33] and implemented in the SciPy library, is exploited. In particular, in case of shooting functions the tolerance is set $xtol = 10^{-12}$ while in the backward-generation method it is selected as $xtol = 10^{-15}$. It is worth mentioning that to initialize the algorithm for the shooting problems the initial components of the costate are sampled from a uniform distribution $\mathcal{U}(-1, 1)$ while the time of flight is sampled from a uniform distribution $\mathcal{U}(1, 2)$ with the bounds expressed in years. Note that the objective of the manuscript is not to find the global optimal solutions of the transfers, indeed sub-optimal solutions are sufficient to acquire a general understanding of the capabilities of neural networks in the context of spacecraft guidance.

To conclude, it is useful to point out that the Numba library [25] is exploited to speed-up the code execution. In this regard, it is interesting to present the computation times associated with all the steps required to obtain a neurocontroller capable of guiding the spacecraft with the performance highlighted in the previous sections. The results are given in Table 6.12, note that the computations are performed on MacBook Air M1 2020.

Table 6.12: Computational times.

Task	**Computational Time**
Solve Shooting Problem	$\mathcal{O}(10^1\,\mathrm{s})$
Create Database of 100 Trajectories	$\mathcal{O}(10^1\,\mathrm{s})$
Train Single Neural Network	$\mathcal{O}(10^2\,\mathrm{s})$

7 | Conclusion

People think of education as
something they can finish.

Isaac Asimov

7.1. Final Remarks

In this work, neural networks are investigated in the context of autonomous guidance of spacecraft. In particular, the research addresses in details the problems related to finding optimal trajectories, creating databases of optimal trajectories and constructing neural networks that can optimally control spacecraft.

In Chapter 5 the backward generation method, originally presented in [17], is extended to the specific case of interplanetary rendezvous with target planets. In this regard, the size of the perturbation used to generate the back-propagated trajectories is finely tuned to introduce the concept of trust band. In particular, the trust band is exploited to specify the range of validity of using neural networks in low-thrust transfers. In addition, this concept proves to be a useful diagnostic tool in the analysis of the trajectories generated by neural network-based controllers. Indeed, in Chapter 6, it is shown that if the spacecraft is driven out of the band, the performance of the neurocontrollers experience a significant drop that is ultimately reflected in larger final position and velocity errors.

Furthermore, in Chapter 5, the reliability issue of standard neural networks is addressed in the framework of physics-informed networks. To this aim, the mathematical structure of the optimality principles is imposed as soft constraint in the training process of the physics-informed networks. In details, the loss functions are constructed by exploiting the residual of the Hamilton-Jacobi-Bellman partial differential equation and the relation between the value function and the costate of the Euler-Lagrange equations.

In Chapter 6 the performances of physics-informed and standard neural networks are compared in the context of two case studies. In the Earth-Venus problem, physics-informed networks show performance comparable to standard ones. In the fuel-optimal problem, instead, the introduction of physics turned out to be extremely advantageous: the networks locally learn the structure of the Hamilton-Jacobi-Bellman equation. This conclusion is also supported by the ability of physics-informed neural networks observed in predicting transfer costs with high accuracy. Therefore, physics-informed networks not only prove to be more effective in guiding spacecraft, but they also present this valuable additional feature which can be exploited at the preliminary stage of mission analysis. The only exception to the previous discussion is a particular physics-informed network which is trained to respect just the the Hamilton-Jacobi-Bellman equation. To further investigate the reason why this network shows poor performance, the loss landscape around the minimizer convergence point is studied. Eventually it is deduced that the cause could be that the optimizer gets stuck on local minima due to the complexity of the loss landscape.

To conclude, it must be underlined that in all the case studies the neural networks-based controllers are unable to guide the spacecraft to the final destination with the accuracy required for operational purposes. Nevertheless, these results are consistent with those found in literature, and therefore the accuracy of terminal guidance should be addressed in future research.

7.2. Future Research

It is clear that still a lot of research needs to be carry out to successfully exploit neural networks in the context of autonomous guidance of spacecraft.

The use of the Hamilton-Jacobi-Bellman principle to increase the reliability of neural networks is a promising methodology. Nevertheless, further studies must be conducted to clarify how to take full advantage of this valuable mathematical knowledge. In particular, it would be interesting to investigate whether it is possible to exploit the mechanisms of physics-informed neural networks to build an architecture that does not have to rely on a database of optimal trajectories in the training process. To this aim, the boundary condition of the Hamilton-Jacobi-Bellman equation must be successfully implemented in the loss function and the differential equation must be solved on a mathematical subset that best represents the possible states of the spacecraft. In this regard, the choice of the subset and the sampling process are critical and it would be of interest to explore the technique proposed in [35]. This paradigm shift would be extremely useful because the database generation phase, which remains a critical problem, can be avoided. In fact, although the backward generation method is a brilliant technique, it has some limitations: in particular it is difficult to control the back-propagated trajectories, especially in fuel-optimal problems. In addition, it is not yet known whether this method is applicable to more complex interplanetary transfers. On the other hand, the alternative to populate the database by solving hundreds or thousands of trajectory optimization problems with classical methodologies presents evident drawbacks form the computational point of view and it introduces an additional layer of complexity.

Another open point is the construction of the composite loss function. As seen in Chapter 6, the mathematical formulation of the loss function can make the minimization process extremely complex. Thus, the visualization techniques of the loss landscape can be exploited to further investigate this aspect. In particular, these techniques can be used to select the weights to balance the different loss terms as mentioned in [23]. Indeed, the weights have a dramatic impact on the outcome of the training process and the choice still relies heavily on a trial and error procedure. It should be noted that in the general context of physics-informed networks, preliminary results in this regard are present in literature [45]. Thus, it would be interesting to consider these findings in the context of spacecraft guidance problems.

As pointed out in Chapter 6, the neural networks-based controllers are still unable to accurately guide spacecraft to the final destinations. Hence the need to develop a methodology to reduce the final errors. In this regard, the study presented in [8] could be a good starting point. Taking inspiration from this work, an interesting option would be to to divide the transfer in many sub-portions and then train separate neural networks to guide the spacecraft along the segments of the trajectory. Therefore, the final transfer would be achieved with a cooperation strategy between multiple networks.

To conclude the overview of possible research directions, it would be interesting to extend the analysis conducted in this work to other types of problem such as time-optimal problems, and in general to interplanetary transfers with more complex boundary conditions. In this context, the case of the time-dependent value function has yet to be addressed and could provide useful knowledge to take a step forward in the study of physics-informed neural networks.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] D. Adams. *The Hitchhiker's Guide to the Galaxy*. Hitchhiker series. Harmony Books, 1980. ISBN 9780517542095. URL `https://books.google.it/books?id=BvjFAAAAIAAJ`.

[3] I. Asimov. *Foundation*. Number v. 1 in Foundation. Random House Worlds, 2004. ISBN 9780553900347. URL `https://books.google.it/books?id=IwywDY4P6gsC`.

[4] R. Bertrand and R. Epenoy. New smoothing techniques for solving bang–bang optimal control problems—numerical results and statistical interpretation. *Optimal Control Applications and Methods*, 23(4):171–197, 2002. doi: https://doi.org/10.1002/oca.709. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/oca.709`.

[5] J. T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming, Third Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2020. doi: 10.1137/1.9781611976199. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611976199`.

[6] A. Bryson and Y. Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. A Halsted Press book. Hemisphere Publishing Corporation, 1975. ISBN 9780470267745. URL `https://books.google.it/books?id=1HWzQgAACAAJ`.

[7] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis. Physics-Informed Neural Networks for Heat Transfer Problems. *Journal of Heat Transfer*, 143(6), 04

2021. ISSN 0022-1481. doi: 10.1115/1.4050542. URL `https://doi.org/10.1115/1.4050542`. 060801.

[8] L. Cheng, Z. Wang, F. Jiang, and C. Zhou. Real-time optimal control for spacecraft orbit transfer via multiscale deep neural networks. *IEEE Transactions on Aerospace and Electronic Systems*, 55(5):2436–2450, 2019. doi: 10.1109/TAES.2018.2889571.

[9] L. Cheng, Z. Wang, Y. Song, and F. Jiang. Real-time optimal control for irregular asteroid landings using deep neural networks. *Acta Astronautica*, 170:66–79, 2020. ISSN 0094-5765. doi: https://doi.org/10.1016/j.actaastro.2019.11.039. URL `https://www.sciencedirect.com/science/article/pii/S0094576520300151`.

[10] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, Jul 2022. ISSN 1573-7691. doi: 10.1007/s10915-022-01939-z. URL `https://doi.org/10.1007/s10915-022-01939-z`.

[11] B. Dachwald. Optimization of very-low-thrust trajectories using evolutionary neurocontrol. *Acta Astronautica*, 57(2):175–185, 2005. ISSN 0094-5765. doi: https://doi.org/10.1016/j.actaastro.2005.03.004. URL `https://www.sciencedirect.com/science/article/pii/S0094576505000627`. Infinite Possibilities Global Realities, Selected Proceedings of the 55th International Astronautical Federation Congress, Vancouver, Canada, 4-8 October 2004.

[12] M. Deisenroth, A. Faisal, and C. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. ISBN 9781108470049. URL `https://books.google.it/books?id=pbONxAEACAAJ`.

[13] R. Furfaro, A. D'Ambrosio, E. Schiassi, and A. Scorsoglio. *Physics-Informed Neural Networks for Closed-Loop Guidance and Control in Aerospace Systems*. doi: 10.2514/6.2022-0361. URL `https://arc.aiaa.org/doi/abs/10.2514/6.2022-0361`.

[14] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Na-*

ture, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[16] D. Izzo and S. Origer. Neural representation of a time optimal, constant acceleration rendezvous. *Acta Astronautica*, 2022. ISSN 0094-5765. doi: https://doi.org/10.1016/j.actaastro.2022.08.045. URL `https://www.sciencedirect.com/science/article/pii/S0094576522004581`.

[17] D. Izzo and E. Öztürk. Real-time guidance for low-thrust transfers using deep neural networks. *Journal of Guidance, Control, and Dynamics*, 44(2):315–327, 2021. doi: 10.2514/1.G005254. URL `https://doi.org/10.2514/1.G005254`.

[18] D. Izzo, M. Märtens, and B. Pan. A survey on artificial intelligence trends in spacecraft guidance dynamics and control. *Astrodynamics*, 3(4):287–299, Dec 2019. ISSN 2522-0098. doi: 10.1007/s42064-018-0053-6. URL `https://doi.org/10.1007/s42064-018-0053-6`.

[19] A. D. Jagtap, K. Kawaguchi, and G. Em Karniadakis. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2239):20200334, 2020. doi: 10.1098/rspa.2020.0334. URL `https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2020.0334`.

[20] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.109136. URL `https://www.sciencedirect.com/science/article/pii/S0021999119308411`.

[21] F. Jiang, H. Baoyin, and J. Li. Practical techniques for low-thrust trajectory optimization with homotopic approach. *Journal of Guidance, Control, and Dynamics*, 35 (1):245–258, 2012. doi: 10.2514/1.52476. URL `https://doi.org/10.2514/1.52476`.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. URL `https://arxiv.org/abs/1412.6980`.

[23] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 26548–26560. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/df438e5206f31600e6ae4af72f2725f1-Paper.pdf`.

[24] I. Lagaris, A. Likas, and D. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5): 987–1000, 1998. doi: 10.1109/72.712178.

[25] S. K. Lam, A. Pitrou, and S. Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[26] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf`.

[27] H. Li, H. Baoyin, and F. Topputo. Neural networks in time-optimal low-thrust interplanetary transfers. *IEEE Access*, 7:156413–156419, 2019. doi: 10.1109/ACCESS.2019.2946657.

[28] H. Li, S. Chen, D. Izzo, and H. Baoyin. Deep networks as approximators of optimal low-thrust and multi-impulse cost in multitarget missions. *Acta Astronautica*, 166:469–481, 2020. ISSN 0094-5765. doi: https://doi.org/10.1016/j.actaastro.2019.09.023. URL `https://www.sciencedirect.com/science/article/pii/S0094576519312901`.

[29] D. Liberzon. *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, 2012. ISBN 9780691151878. URL `https://books.google.it/books?id=mhklgjmACRUC`.

[30] J. Longuski, J. Guzmán, and J. Prussing. *Optimal Control with Aerospace Applications*. Space Technology Library. Springer New York, 2013. ISBN 9781461489450. URL `https://books.google.it/books?id=9y65BAAAQBAJ`.

[31] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2019.112789. URL `https://www.sciencedirect.com/science/article/pii/S0045782519306814`.

[32] S. Markidis. The old and the new: Can physics-informed deep-learning replace traditional linear solvers? *Frontiers in Big Data*, 4, 2021. ISSN 2624-909X. doi: 10.3389/fdata.2021.669097. URL `https://www.frontiersin.org/articles/10.3389/fdata.2021.669097`.

[33] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. User guide for minpack-1. 1980.

[34] D. Mortari. The theory of connections: Connecting points. *Mathematics*, 5(4), 2017. ISSN 2227-7390. doi: 10.3390/math5040057. URL `https://www.mdpi.com/2227-7390/5/4/57`.

[35] M. A. Nabian, R. J. Gladstone, and H. Meidani. Efficient training of physics-informed neural networks via importance sampling. *Computer-Aided Civil and Infrastructure Engineering*, 36(8):962–977, 2021. doi: https://doi.org/10.1111/mice.12685. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12685`.

[36] T. Nakamura-Zimmerer, Q. Gong, and W. Kang. Adaptive deep learning for high-dimensional hamilton–jacobi–bellman equations. *SIAM Journal on Scientific Computing*, 43(2):A1221–A1247, 2021. doi: 10.1137/19M1288802. URL `https://doi.org/10.1137/19M1288802`.

[37] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL `https://www.sciencedirect.com/science/article/pii/S0021999118307125`.

[38] A. Rubinsztejn, R. Sood, and F. E. Laipert. Neural network optimal control in astrodynamics: Application to the missed thrust problem. *Acta Astronautica*, 176:192–203, 2020. ISSN 0094-5765. doi: https://doi.org/10.1016/j.actaastro.2020.05.027. URL `https://www.sciencedirect.com/science/article/pii/S0094576520303106`.

[39] S. Ruder. An overview of gradient descent optimization algorithms, 2016. URL `https://arxiv.org/abs/1609.04747`.

[40] C. Sánchez-Sánchez and D. Izzo. Real-time optimal control via deep neural networks: Study on landing problems. *Journal of Guidance, Control, and Dynamics*, 41(5): 1122–1135, 2018. doi: 10.2514/1.G002357. URL `https://doi.org/10.2514/1.G002357`.

[41] E. Schiassi, A. D'Ambrosio, K. Drozd, F. Curti, and R. Furfaro. Physics-informed neural networks for optimal planar orbit transfers. *Journal of Spacecraft and Rockets*, 59(3):834–849, 2022. doi: 10.2514/1.A35138. URL `https://doi.org/10.2514/1.A35138`.

[42] M. Shirobokov, S. Trofimov, and M. Ovchinnikov. Survey of machine learn-

ing techniques in spacecraft control design. *Acta Astronautica*, 186:87–97, 2021. ISSN 0094-5765. doi: https://doi.org/10.1016/j.actaastro.2021.05.018. URL `https://www.sciencedirect.com/science/article/pii/S0094576521002514`.

[43] E. Taheri, I. Kolmanovsky, and E. Atkins. Enhanced smoothing technique for indirect optimization of minimum-fuel low-thrust trajectories. *Journal of Guidance, Control, and Dynamics*, 39(11):2500–2511, 2016. doi: 10.2514/1.G000379. URL `https://doi.org/10.2514/1.G000379`.

[44] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17: 261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[45] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021. doi: 10.1137/20M1318043. URL `https://doi.org/10.1137/20M1318043`.

[46] S. Yin, J. Li, and L. Cheng. Low-thrust spacecraft trajectory optimization via a dnn-based method. *Advances in Space Research*, 66(7):1635–1646, 2020. ISSN 0273-1177. doi: https://doi.org/10.1016/j.asr.2020.05.046. URL `https://www.sciencedirect.com/science/article/pii/S0273117720303951`.

[47] C. Zhang, F. Topputo, F. Bernelli-Zazzera, and Y.-S. Zhao. Low-thrust minimum-fuel optimization in the circular restricted three-body problem. *Journal of Guidance, Control, and Dynamics*, 38(8):1501–1510, 2015. doi: 10.2514/1.G001080. URL `https://doi.org/10.2514/1.G001080`.

[48] L. Zhu, J. Ma, and S. Wang. Deep neural networks based real-time optimal control for lunar landing. *IOP Conference Series: Materials Science and Engineering*, 608 (1):012045, aug 2019. doi: 10.1088/1757-899X/608/1/012045. URL `https://dx.doi.org/10.1088/1757-899X/608/1/012045`.

List of Figures

List of Tables

Acknowledgements

I would like to thank my advisor, Professor Francesco Topputo, for being first and foremost a source of inspiration for the passion and dedication he puts into his work. Also, I would like to thank him for giving me the opportunity to work on this project and for the professionalism and helpfulness he has shown in guiding me along this path. I would also like to thank Christian Hofmann, Alessandra Mannocchi, Andrea Carlo Morelli, and Mattia Pugliatti for their valuable advice, the continuous support, the availability, and the interest they have shown in my thesis work.

I would like to sincerely thank my family, Mario, Paola and Mattia, for allowing me to pursue my dreams, for supporting me on my path and always believing in my abilities, without your fundamental help I would never have achieved this goal. Together we will be able to overcome any difficulty.

A huge thanks to Nina, for being my companion in every adventure, for patiently listening to all my space talks, for always finding the right words to hearten me in difficult times, and for being my number one supporter, thus transmitting precious energy to me.

Special thanks to my lifelong friends for an infinite number of reasons that I cannot list all here. But thank you in particular for all the discussions that have allowed us to grow together and for all the incredible adventures we have had, it will be a pleasure to get old with you.

An endless thanks to my family and family friends for the countless times they have shown interest in my journey and for never stopping cheering me on.

Thank you to the people who have accompanied me on this journey of master, especially thank you to Nemanja, Giovanni and Enrico for all the moments of comparison, which allowed us to learn together, and for the moments of fun, which were essential for us not to give up. Thanks also to my RASC-Al project friends for making the trip to America first a reality and then an unforgettable experience.

Finally, thanks to all the works on space that have made me passionate about this wonderful world and continue to inspire me day after day, especially thanks to The Martian, the Foundation Series and Outer Wilds.

Ringraziamenti

Vorrei ringraziare il mio relatore, il Professore Francesco Topputo, per essere stato innanzitutto una fonte di ispirazione per la passione e la dedizione con cui si cimenta nel suo lavoro. Inoltre, vorrei ringraziarlo per avermi dato l'opportunità di lavorare a questo progetto e per la professionalità e la disponibilità con cui mi ha accompagnato in questo percorso.

Vorrei ringraziare particolarmente Christian Hofmann, Alessandra Mannocchi, Andrea Carlo Morelli e Mattia Pugliatti per i loro preziosi consigli, il continuo supporto, la disponibilità e l'interesse mostrato per il mio lavoro di tesi.

Ci tengo a ringraziare di cuore la mia famiglia, Mario, Paola e Mattia, per avermi permesso di inseguire i miei sogni, per avermi sostenuto nel mio percorso e per aver sempre creduto nelle mie capacità, senza il vostro fondamentale aiuto non sarebbe stato possibile raggiungere questo traguardo. Insieme riusciremo a superare qualsiasi difficoltà.

Un enorme grazie a Nina, per essere la mia compagna in ogni avventura, per aver pazientemente ascoltato tutti i miei discorsi sullo spazio, per aver sempre trovato le parole giuste per rincuorarmi nei momenti difficili e per essere la mia sostenitrice numero uno, trasmettendomi così un'energia preziosa.

Un ringraziamento speciale ai miei amici di sempre, che se sono amici da sempre è per un'infinità di motivi che non posso elencare tutti qui. Però grazie in particolare per tutti i confronti che ci hanno permesso di crescere insieme e per tutte le incredibili avventure che abbiamo vissuto, sarà un piacere invecchiare con voi.

Un grazie infinito ai miei parenti e agli amici di famiglia per le innumerevoli occasioni in cui si sono interessati al mio percorso e per non aver mai smesso di tifare per me.

Grazie alle persone che mi hanno accompagnato in questo percorso di magistrale, in particolare grazie a Nemanja, Giovanni e Enrico per tutti i momenti di confronto, che ci hanno permesso di imparare insieme, e per i momenti di svago, fondamentali per non arrenderci. Un grazie anche ai miei compagni del progetto RASC-Al per aver reso il viaggio in America prima una realtà e poi un'esperienza indimenticabile.

Infine, grazie a tutte le opere sullo spazio che mi hanno fatto appassionare a questo meraviglioso mondo e che continuano a ispirarmi giorno dopo giorno, in particolare grazie a The Martian, al Ciclo della Fondazioni e ad Outer Wilds.